NEURALSTAGGER: ACCELERATING PHYSICS CONSTRAINED NEURAL PDE SOLVER WITH SPATIAL-TEMPORAL DECOMPOSITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural networks have shown great potential in accelerating the solution of partial differential equations (PDEs). Recently, there has been a growing interest in introducing physics constraints into training neural PDE solvers to reduce the use of costly data and improve the generalization ability. However, these physics constraints, based on certain finite dimensional approximation over the function space, must resolve the smallest scaled physics to ensure the accuracy and stability of the simulation, resulting in heavy computational costs from large input, output, and neural networks. This paper proposes a general acceleration methodology called NeuralStagger by spatially and temporally decomposing the original learning tasks into several coarser-resolution subtasks. We define a coarse-resolution neural solver for each subtask, which requires fewer computational resources, and jointly train them with the vanilla physics constrained loss by simply arranging their outputs to reconstruct the original solution. Due to the perfect parallelism between them, the solution is achieved as fast as a coarse-resolution neural solver. In addition, the trained solvers bring the flexibility for users to simulate with multiple levels of resolution. We demonstrate the successful application of NeuralStagger on various fluid dynamics simulations, which leads to an additional 10 to 100 times speed-up. Moreover, the experiment also shows that the learned model could be well used for optimal control.

1 INTRODUCTION

Partial differential equations (PDEs) are the critical parts of scientific research, describing vast categories of physical and chemical phenomena, e.g. sound, heat, diffusion, electrostatics, electrodynamics, thermodynamics, fluid dynamics, elasticity, and so on. In the era of artificial intelligence, neural PDE solvers, in some works called neural operators, are widely studied as a promising technology to solve PDEs (Guo et al., 2016; Zhu & Zabaras, 2018; Hsieh et al., 2019; Bhatnagar et al., 2019; Bar-Sinai et al., 2019; Berner et al., 2020; Li et al., 2020b;a; Um et al., 2020; Pfaff et al., 2020; Lu et al., 2021b; Wang et al., 2021; Kochkov et al., 2021). Once the neural solver is trained, it can solve unseen PDEs with only an inference step, multiple magnitudes faster than that with traditional numerical solvers. Recently, several works have introduced physics constraints in training the neural PDE solvers in order to reduce the use of costly data and improve the generalization ability. They define the physics constrained loss with certain finite dimensional approximations to transform the PDEs into algebraic equations, which are further used to define the loss function (Zhu et al., 2019; Geneva & Zabaras, 2020; Wandel et al., 2020; Shi et al., 2022). However, to ensure stability and accuracy, they must define the loss in a relatively high resolution to resolve the smallest-scale physics in the PDE, resulting in huge input and output as well as increased neural network size. The solution by the neural network inference might still be slow, but it seems impossible to get further accelerations as the bottleneck comes from the input and output complexity.

In this paper, we propose a simple methodology called NeuralStagger to jump out of the dilemma. The basic idea is to evenly decompose the original physical fields into several coarser-resolution fields. Then we jointly train a lightweight neural network to predict the solution in each coarse-resolution field respectively, which can be naturally a coarse-resolution neural solver to the original PDE. We design the decomposition rules so that the outputs of these lightweight networks can re-

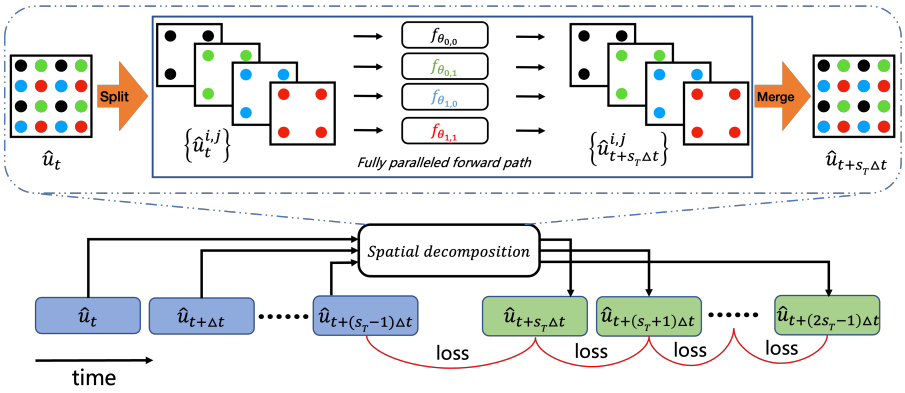

Figure 1: The training pipeline of NeuralStagger. **Top**: the spatial decomposition that splits the field into several pieces of coarse-resolution fields; **Bottom**: the temporal decomposition combined with spatial decomposition to construct the physics constrained loss.

construct the solutions in the original field with simple arrangements. For ease of reading, here and also in most parts of the paper, we illustrate the decomposition methodology in the 2-dimensional example with regular mesh and finite difference approximation. Figure 1 (top) shows the physical field in a 4×4 mesh is decomposed into 4 coarser-resolution fields, each of which is handled by a small neural network. We could also do similar things along the temporal dimension, as is shown in Figure 1 (bottom). The group of coarse-resolution solvers as well as the decomposition and reconstruction operations can be seen as an end-to-end neural PDE solver, which can be trained with the physics constrained loss that resolves small-scale physics in a sufficiently high resolution. Because the neural networks can run in parallel, the original simulation is achieved as fast as a coarse-resolution neural solver. In addition, the trained neural networks can predict the PDE's solution in various levels of resolution, ranging from the resolution of the individual coarse-resolution solver to the resolution of the physics constrained loss by the combination of all these solvers. We believe that such flexibility is vital in balancing the computational resources and the resolution.

We demonstrate the effectiveness of the NeuralStagger in the Navier-Stokes equation with three parametric settings, e.g., periodic boundary conditions with varied initial conditions, lid-driven cavity boundary conditions with varied initial conditions, and the flow around the obstacle with varied obstacles and initial conditions. We find that with NeuralStagger, the learned networks can conduct accurate and stable simulation with 10$\sim$100 times speed-up over SOTA neural PDE solvers. In addition, we demonstrate that they can accurately tackle the optimal control task with auto-differentiation.

Our contributions can be summarized in three parts:

- We propose a general methodology called NeuralStagger to accelerate neural PDE solving by spatially and temporally decomposing the learning task and running a group of coarse-resolution solvers in parallelism.
- The learned network group can provide solutions in multiple resolutions from the coarsest one by a single network to the original resolution, which provides the flexibility to balance the computational resources and the resolution.
- Empirically, we demonstrate that the methodology leads to 10 to 100 times speed-up over SOTA neural PDE solvers as well as the efficient solution on optimal control.

In the following sections, we first briefly summarize the related works in Section 2 and then introduce the preliminaries and the proposed NeuralStagger in Section 3. To showcase the efficiency and accuracy of the proposed method, we present the settings of the experiment and results in Section 4. Finally, we conclude and discuss the future work in Section 5.

2 RELATED WORK

In general, two mainstream approaches have been widely used for solving PDEs. The first is to approximate the PDE's solution function with neural networks (Raissi et al., 2019; 2020; Jin et al., 2021). They have proved to be successful in tackling high-dimensional problems and inverse problems. The second is to learn a PDE solver to solve parametric PDEs. The neural PDE solver can learn the solutions of a class of PDEs, and thus can generalize to PDEs with different parameters. Our work is mainly about the accelerating the second type. Many impressive works have been done to improve the neural solver for parametric PDEs in terms of neural network design, e.g., convolutional neural network (Guo et al., 2016; Tompson et al., 2017; Bhatnagar et al., 2019), graph neural networks (Pfaff et al., 2020), the multipole graph kernel (Li et al., 2020b), Fourier neural operators (Li et al., 2020a; Guibas et al., 2021), the message passing neural network (Brandstetter et al., 2022b), deepOnet (Lu et al., 2021a), Clifford neural networks (Brandstetter et al., 2022a) and so on. After being trained with pre-generated simulated data and labels, they can solve the PDE several magnitudes faster than conventional numerical solvers with competitive accuracy. Recently there are raising concerns about the cost of collecting training data and the generalization ability, so several works have introduced the physics constrained loss for training. For example, (Wang et al., 2021) combined the DeepOnet with a physics-informed way to improve the sample efficiency. Zhu et al. (2019) proposed physics constrained loss for high-dimensional surrogate modeling and (Geneva & Zabaras, 2020) introduced the use of a physics constrained framework to achieve the data-free training in the case of Burgers equations. Wandel et al. (2020; 2021) proposed the physics constrained loss based on the certain approximation of the Navier-Stokes equation to solve fluid-like flow problems. Shi et al. (2022) proposed a general physics constrained loss called mean square residual (MSR) loss as well as a neural network called LordNet for better performance. However, the physics constrained loss by certain approximations require the approximation to be sufficiently close to the continuous version, resulting in a relatively high-resolution discretization. Thus in complex and large-scale problems, the neural solver must be large enough for expressiveness and its inference would still be slow. Although some works (Wang et al., 2021) directly calculate the derivatives via back-propagation through the neural network, they are known to have similar training problems as PINN, e.g., converging to trivial solutions.

Interestingly in the case of regular mesh, the proposed spatial decomposition is the same in the implementation as 'pixel shuffle' from computer vision. There are a huge number of works in this direction, but the most related one might be (Ren et al., 2022) which leverages pixel shuffle and physics constrained loss in the super-resolution task. However, we are fundamentally different in target and solution. For example, we train multiple solvers to work in full parallelism and obtain the solution in multiple levels of resolution without training them again.

We also find similar treatment on meshes in classical numerical methods, e.g., staggered-mesh and leap-frog integration. However, they are also fundamentally different in target and implementation. The numerical methods often place meshes of multiple fields with offsets to get more accurate approximation while NeuralStagger splits the mesh of every single field into multiple sub-meshes for defining the independent subtasks. In addition, they are orthogonal to NeuralStagger, i.e., one can leverage both the staggered-mesh to define the physics constrained loss and NeuralStagger to train multiple coarse-resolution solvers at the same time, as is done in our experiment.

3 METHODOLOGY

3.1 PRELIMINARIES

Consider a connected domain $\Omega \subseteq \mathbb{R}^n$ with boundary $\partial\Omega$, and let $(\mathcal{A}, \mathcal{U}, \mathcal{V})$ be separable Banach spaces. Then the parametric PDEs can be defined as the form

$$\mathcal{S}(\mathbf{u}, \mathbf{a})(\mathbf{x}) = 0, \quad \mathbf{x} \in \Omega \tag{1}$$

where $\mathcal{S} : \mathcal{U} \times \mathcal{A} \to \mathcal{V}$ is a linear or nonlinear differential operator, $\mathbf{a} \in \mathcal{A}$ denotes the parameters under certain distribution μ, such as coefficient functions or boundary/initial conditions, and $\mathbf{u} \in \mathcal{U}$ is the corresponding unknown solution function. Further, we can define the solution operator of the parametric PDE $G : \mathcal{A} \to \mathcal{U}$, which maps two infinite-dimensional function spaces.

A main branch of works in neural PDE solvers approximate the solution operator by discretizing the functions into finite dimensional spaces denoted by $\hat{\mathcal{A}}$ and $\hat{\mathcal{U}}$ and learning the mapping $f_\theta : \hat{\mathcal{A}} \to \hat{\mathcal{U}}$. Correspondingly, we have the discretized version of the PDE's operator $\mathcal{S}$ by certain finite-dimensional approximations such as the finite difference method (FDM) and finite element method (FEM), which is denoted by $\hat{\mathcal{S}}$. We denote the vector of the function values in a mesh with the hat symbol, e.g., $\hat{a}$ is the vector of the PDE's parameter $\mathbf{a} \sim \mu$. Then the physics constrained loss is defined by forcing the predicted solution $\hat{u} \in \hat{\mathcal{U}}$ to satisfy $\hat{\mathcal{S}}$ given $\hat{a} \in \hat{\mathcal{A}}$. For example, LordNet (Shi et al., 2022) proposed the general form with the mean squared error as follows,

$$L(\theta) = \mathbb{E}_{a\sim\mu}||\hat{\mathcal{S}}(f_\theta(\hat{a}), \hat{a})||^2, \tag{2}$$

In this paper, we mainly focus on time-dependent problems as follows,

$$\mathcal{S}(\mathbf{u}, \mathbf{a})(t, \mathbf{x}) = 0, \quad (t, \mathbf{x}) \in [0, T] \times \Omega \tag{3}$$

The temporal dimension is discretized with the timestep Δt and the neural solver solves the PDE in an auto-regressive way,

$$\hat{u}_{t+\Delta t} = f_\theta(\hat{u}_t, \hat{a}) \tag{4}$$

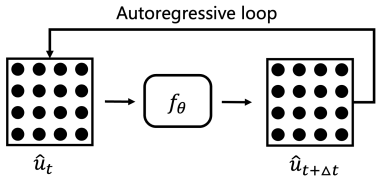

Figure 2: Autoregressive model.

where $\hat{u}_t$ is the corresponding discretized vector of the function $\mathbf{u}$ at time t. Figure 2 shows an example with a 4×4 rectangle mesh. Notice that similar to traditional numerical methods, the resolution of the finite-dimensional approximation in physics constrained loss, either in the spatial dimension or in the temporal dimension, must be sufficiently high, otherwise, the approximation error will be too large to guide the neural PDE solver. This leads to huge input and output as well as large neural networks to ensure expressiveness, whose inference would also be slow.

3.2 NEURALSTAGGER

We propose a general methodology called NeuralStagger to gain further accelerations by exploiting the potential parallelism in the neural PDE solver. NeuralStagger decomposes the original learning task that maps $\hat{u}_t$ to $\hat{u}_{t+\Delta t}$ into several parallelizable subtasks in both spatial and temporal dimensions. The meshes of the subtasks spread evenly in the original field and stagger with each other. Then we can handle each subtask with a computationally cheap neural network. The decomposition strategy is introduced as follows.

Spatial decomposition. The upper part of Figure 1 shows the 2-dimensional example with regular mesh. We first split the grid into patches of the size $s_H \times s_W$ and construct a subgrid by selecting only one point in each patch, resulting in $s_H \times s_W$ subgrids evenly spread in the domain. We denote the functions in each sub-grid as $\hat{u}_t^{i,j}$ and $\hat{a}_t^{i,j}$ where i and j represents the relative position of the sub-grid in horizontal and vertical directions. Then we use $s_H \times s_W$ neural networks to learn to predict the solution at $t + \Delta t$ as follows,

$$\hat{u}_{t+\Delta t}^{i,j} = f_{\theta_{i,j}}(\hat{u}_t^{i,j}, \hat{a}^{i,j}), \tag{5}$$

where $f_{\theta_{i,j}}$ is the neural network for the sub-grid at the position (i, j). The outputs $\hat{u}_{t+\Delta t}^{i,j}$ compose the solution at the original grid. Then the neural networks can be jointly trained with the physics constrained loss defined on the original grid. Notice that the neural networks are independent of each other and can be fully paralleled. As the input and output decrease by $s_H \times s_W$ times, the neural network can be much smaller and faster than the original one to be used for the neural solver. The decomposition rules can be extended to higher-dimensional cases. In addition, the learning tasks at the subgrids are quite close to each other, except for the difference in the boundary of the domain, so we share the parameters of the neural networks $f_{\theta_{i,j}}$ to reduce redundancy and accelerate training. Meanwhile, because there are often tiny differences between the inputs of the subtasks, we encourage the neural network to distinguish them by adding positional information of each grid point as additional input channels.

Temporal decomposition. We can treat the temporal dimension as a 1-dimensional grid with a fixed step Δt. Thus we can also decompose the grid into s_T sub-grids by selecting a point for every

s_T points, where instead of predicting $\hat{u}_{t+\Delta t}$, the neural network predicts $\hat{u}_{t+s_T\Delta t}$,

$$\hat{u}_{t+s_T\Delta t} = f_\theta\left(\hat{u}_t, \hat{a}\right), \tag{6}$$

Given the solution sequence from t to $t+(s_T-1)\Delta t$ denoted by $\hat{u}_{t,s_T}$ for simplicity, we can get the next sequence of the solution $\hat{u}_{t+s_T\Delta t,s_T}$. Then the physics constrained loss is defined on the sequence with timestep Δt, as is shown in the lower part of Figure 1. Once the neural network is trained, we can generate the sequence $\hat{u}_{t+s_T\Delta t,s_T}$ by running the neural network inference of Formula 6 with s_T threads in parallel with inputs $\hat{u}_{t,s_T}$. The non-auto-regressive process can generate the solution in s_T time steps within one inference step, which can be much faster than the original version (Figure 2) with s_T inference steps. Note that though we only need the initial condition for the coarsest-resolution test, we must prepare the first s_T states with numerical solvers for training and the high-resolution test. However, this drawback is neglectful for long-time simulations.

The spatial and temporal decompositions are orthogonal and can be used at the same time. We denote the joint decomposition operator as $D_{\mathbf{s}}$, the transformation operator of the neural networks as F_Θ and the reconstruction operator $E_{\mathbf{s}}$, where $\mathbf{s}$ represents all decomposition factors including s_H, s_W and s_T, Θ represents all parameters of the neural network group. The physics constrained loss with the spatial-temporal decomposition can be written as,

$$L(\Theta) = \mathbb{E}_{\hat{u}_{t,s_T}} ||\hat{\mathcal{S}}\left(E_{\mathbf{s}}\left(F_\Theta\left(D_{\mathbf{s}}\left(\hat{u}_{t,s_T}, \hat{a}\right)\right)\right), \hat{u}_{t,s_T}, \hat{a}\right)||^2. \tag{7}$$

In addition, as the sub-grids spread evenly in the domain of the PDE, each of them can be seen as the down-sampled version of the original problem, where a local patch is reduced to the point at a fixed relative position in the patch. Therefore, the learned neural networks are naturally coarse-resolution solvers to the PDE. Suppose (H, W, T) is the tuple of the original height, width, and time span that the physics constrained loss is conducted on. Then the coarse-resolution solvers are conducted on the resolution $(\frac{H}{s_H}, \frac{W}{s_W}, \frac{T}{s_T})$. Meanwhile, we can infer multiple levels of resolutions ranging from that of coarse-resolution solvers to the original one, all of which can reach the same speed by parallelism.

3.3 Choice of the decomposition factors

Obviously, the acceleration effect by NeuralStagger grows as we use larger s_H, s_W and s_T. However, these decomposition factors cannot be arbitrarily large. We conclude two potential constraints, i.e., the increased complexity of the learning task and the information loss in the input. We would like to leverage the following 2-dimensional diffusion equation with the periodic boundary condition as an example to explain the two constraints,

$$\frac{\partial u(x,y,t)}{\partial t} = \Delta u(x,y,t), \qquad x,y,t \in [0,1], \tag{8}$$

$$u(x,y,0) = f(x,y), \qquad x,y \in [0,1], \tag{9}$$

where u is the density function of diffusing material, Δ is the Laplacian operator and f is the function of the initial condition. We use the regular mesh with d points in total and leverage the central difference scheme with the spatial step Δx and temporal step Δt. Then the PDE is transformed into a matrix equation on the discretized solution at certain time t, denoted by $\hat{u}_t \in \mathbb{R}^d$.

Increased complexity of learning task. For the temporal dimension, we find that the larger decomposition factor might make the mapping from the input to the prediction more complex. For the linear diffusion equation, we can explicitly calculate the transfer matrix from $\hat{u}_i$ to $\hat{u}_{i+\Delta t}$ based on the matrix equation. Suppose the transfer matrix is $T_i \in \mathbb{R}^{d\times d}$. By iterative applying the transfer matrix, we can get the transformation from the initial condition $\hat{u}_0$ to the solution at any time step k as follows,

$$\hat{u}_{k\Delta t} = \hat{u}_0 \prod_0^{k-1} T_i. \tag{10}$$

For notational simplicity, we denote the resulting transfer matrix from $\hat{u}_0$ to $\hat{u}_{k\Delta t}$ as $\mathcal{T}_k$. By certain arrangements, $\mathcal{T}_k$ is a band

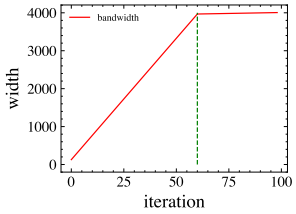

Figure 3: The bandwidth curve.

matrix where the non-zero values are centralized around the diagonal. The bandwidth indicates the sparsity of the matrix as well as how local the points in the mesh entangle with each other. We observe that the bandwidth grows linearly with regard to k. For example, Figure 3 shows the case of $d = 64^2$. When the $k \geq 60$, the matrix is dense and every element in $\hat{u}_{k\Delta t}$ is a weighted summation of almost all the elements in $\hat{u}_t$. This indicates that increasing k may make the entanglements between the grid points more complex, leading to a harder learning task for the neural network.

Information loss. By spatial decomposition, each subgrid only reserves a small part of the original grid. Obviously, it may introduce the problem of information loss if the dropped points are important for the prediction in the subtasks. Here we theoretically characterize the information loss caused by spatial decomposition under the linear model setting, i.e., $f(\hat{u}_t) = \hat{u}_t W^*$. Consider the diffusion equation and the corresponding matrix equation. With some abuse of notation, the superscript i denotes the index of training samples, such as $\hat{u}_t^i$ and the bold symbol without the superscript i denotes the matrix composed of all the samples, such as $\hat{\boldsymbol{u}}_t$. With N training samples, the physics constrained loss aims to learn the parameters W^* of the linear model that satisfies:

$$W^* = \arg\min_W \frac{1}{N} \sum_{i=1}^{N} \|\hat{u}_t^i W - y^i\|^2, \tag{11}$$

where y^i denotes the rest parts of the matrix equation. By applying spatial decomposition, the input and output are equally partitioned into $K = s_H s_W$ subgrids $\{\hat{u}_t^1, \cdots, \hat{u}_t^K\}$ and $\{\hat{u}_{t+1}^1, \cdots, \hat{u}_{t+1}^K\}$. Then according to the physics constrained loss, the optimization goal becomes:

$$W_1^*, \cdots, W_K^* = \arg\min_{W_1, \cdots, W_K} \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \|(\hat{u}_t^{i,k} W_k - y^{i,k})\|^2, \tag{12}$$

where $W_k \in \mathbb{R}^{m \times m}, m = d/K$ for $k = 1, \cdots, K$. The next proposition shows a sufficient condition for equal prediction for Eq.(11) and Eq.(12).

Proposition 1. *If $rank(\hat{\boldsymbol{u}}_t) = rank(\hat{\boldsymbol{u}}_t^k)$, the model $\hat{\boldsymbol{u}}_t W^*$ and $\hat{\boldsymbol{u}}_t^k W_k^*$ will make the same prediction on $\boldsymbol{y}^k$.*

We put the proof in the appendix. In many physical scenarios, the local patches of size $s_H s_W$ do not distribute arbitrarily in the ambient space $\mathbb{R}^{s_H s_W}$, but rather live in some low-dimensional manifold. Hence, there is much information redundancy in $\hat{\boldsymbol{u}}_t$ and with careful settings of s_H and s_W, the rank after the decomposition does not change much, indicating similar predictions on $\boldsymbol{y}^k$. With deep learning models f_θ such as those we use in this paper, we believe that more complex local patterns can be resolved and the spatial factors can be set larger.

4 Experiments

To evaluate the acceleration effect and accuracy of the proposed method, we test three cases of fluid dynamics simulation governed by the Navier-Stokes equation. We first target two benchmark settings, i.e., the periodic boundary condition and the lid-driven cavity boundary condition (Zienkiewicz et al., 2006) In both settings, the initial condition changes, and the neural PDE solver learns to generalize to various initial conditions. Next, we test the more challenging case called flow around obstacles, where several obstacles are placed inside the flow. The neural PDE solver is trained to generalize to different obstacles as well as initial conditions. In addition, the state of the fluid changes quite a lot over time. To ensure the neural solver generalizes to various states, we must maintain a training pool to store states newly predicted during training. At last, we also evaluate the capability to the inverse problem, i.e., the optimal control on the flow-around-obstacles setting.

In general, we consider the 2-dimensional incompressible Navier-Stokes equation as follows:

$$\rho \left(\frac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\vec{v} \right) = -\nabla p + \mu \Delta \vec{v} + \vec{f} \tag{13}$$

$$\nabla \cdot \vec{v} = 0 \tag{14}$$

where $\vec{v}$ is the fluid velocity field, p is the pressure field, μ is the viscosity, and $\vec{f}$ is the external force. In all experiments, we trained neural networks with Adam optimizer and decayed learning rates. The

speed test is done on Nvidia A100 GPUs under the assumption that we have sufficient computational resources for each coarse-resolution solver. See Appendix Section 6.2 for more details.

4.1 PERIODIC AND LID-DRIVEN CAVITY BOUNDARY CONDITION

We first test the Navier-Stokes equation with the periodic boundary condition and the lid-driven cavity boundary condition. In both cases, the physics constrained loss is obtained by discretizing the vorticity-stream equation with the central-difference scheme and the Crank-Nicolson method in the 64×64 regular mesh. The time step Δt is $1\mathrm{e}-2$ and the viscosity ν is $1\mathrm{e}-3$. We use the popular FNO (Li et al., 2020a) to test the accuracy and speed in different settings of decomposition factors. The ground truth is obtained by FDM. We evaluate the accuracy by auto-regressively running the inference of the neural solver across the target length along time L_T and compare the terminal state with that from the ground truth. Note that we compare all the results on the original mesh and thus the spatially decomposed results reconstruct to the 64×64 resolution for evaluation. We measure with the relative error which is calculated by dividing the L2 norm of the error by the L2 norm of the ground truth. The measurement is denoted by Error-k where k is the number of time steps. Following the notations in Section 3.2, the decomposition factors along x dimension, z dimension and the temporal dimension are denoted by s_W, s_H and s_T. In general, NeuralStagger achieves acceleration in both cases without losing much accuracy. As you can see in Figure 5, the coarse-resolution solver is also accurate when applied alone without reconstruction.

In the case of the periodic boundary condition, the target length along time L_T equals 2, which is 200 time steps. The flow is driven by the external force $\vec{f}$, which is introduced in the appendix. As you can see in Figure 4 (left), the relative errors of the learned neural solvers are lower than 0.2% in all settings of spatial and temporal decomposition factors. In terms

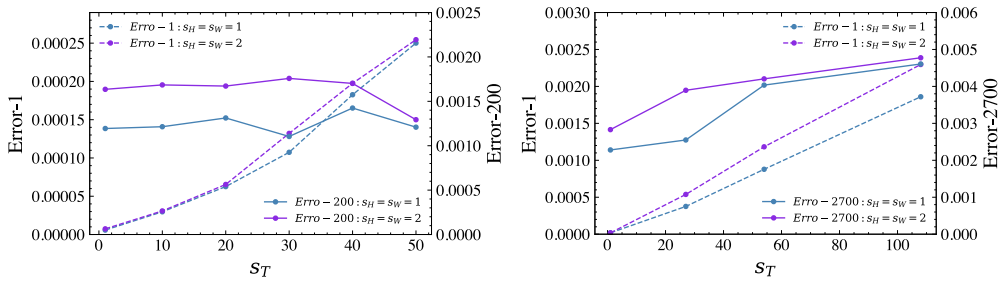

Figure 4: Tests on Navier-Stokes equation with (left) periodic boundary condition and (right) Lid-driven cavity boundary condition.

of speed, with the most aggressive setting $s_T = 40, s_H = s_W = 2$, and full parallelism, the inference time for the 200-time-steps simulation is 0.076 seconds on average. Compared to 0.36 seconds by the baseline without NeuralStagger, there is $47\times$ speed-up. We can also observe some trends in accuracy with regard to the choice of spatial and temporal factors. Error-1 grows like a linear function with the temporal factor s_T in both spatial factor settings. The reason is that the learning task becomes more complex as we discuss in Section 3.3, and with the neural network unchanged, the accuracy drops. Meanwhile, the accumulated errors, i.e., Error-200, almost keep at the same level. This is because the steps in the auto-regressive procedure reduce

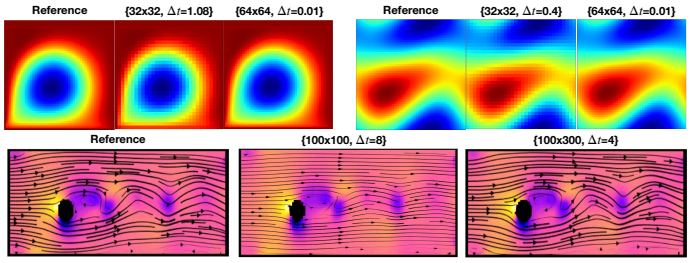

Figure 5: The predictions in two resolutions. **Top**: lid-driven cavity boundary condition (left) and periodic boundary condition (right) and **Bottom**: flow around obstacles.

as s_T grows, e.g., when $s_T = 40$, the neural networks for subtasks only predict $200/40 = 5$ steps ahead. The benefit perfectly neutralizes the detriment of the increased task complexity.

In the case of the lid-driven cavity boundary condition, the fluid acts in a cavity consisting of three rigid walls with no-slip conditions and a lid moving with a steady tangential velocity 1. We set the length of time $L_T = 27$, much larger than that with the periodic boundary, to see if the simulation converges to the right steady state. With larger L_T, we try larger temporal skip factors such as $s_T = 108$. As is shown in Figure 4 (right), the relative errors are all controlled below 0.5% even after 2700 time steps. Again, with the most aggressive setting $s_T = 108, s_H = s_W = 2$ and full parallelism, the neural solver finishes the 2700-time-steps simulation within 0.038 seconds, about $119\times$ faster than the baseline, i.e., 4.49 seconds. Different from the periodic boundary condition, the accuracy drops when we increase s_T. The reason is that the increase of s_T brings more detriments of task complexity than the benefits from the shorter auto-regressive sequence.

4.2 Flow around obstacles

In this section, we evaluate NeuralStagger in a larger and more complex setting called flow around obstacles. The setting is the same as that used in (Wandel et al., 2020), which is also our baseline. The fluid runs through a pipe, where we put different shapes of obstacles to affect the flow, including rotating cylinders and walls constructing a folded pipe. The external forces in Eq. 13 are neglected and set to 0. The neural solver is trained to generalize to different settings of the obstacles, including the shape and the velocity on the surface as well as the inflow/outflow velocities. Then we evaluate the neural solver in 5 randomly sampled configurations in both the cylinder case and the folded pipe case. You may refer to the appendix for more details. We leverage the same configurations as those in (Wandel et al., 2020) including the discretization method, the physics constrained loss, training strategies, the input features, the predicted variables as well as the evaluation metric. Specifically, the rectangular domain is discretized into a 100×300 regular mesh and $\Delta t = 4$. The physics constrained loss is used as the evaluation metric, measuring to what extent the prediction at the next time step satisfies the PDE given the current fluid state and the boundary conditions. As the fields of the fluid change much over time, we maintain a training pool initialized with a set of initial conditions and incrementally enrich it as the training goes. This is achieved because the predictions from the neural network can be seen as new data if the neural network has been well fitted in the current pool. One can refer to (Wandel et al., 2020) for more details.

Wandel et al. (2020) leverages U-net as the neural solver, but to demonstrate the full potential of NeuralStagger, we also try the other two neural network architectures, i.e., FNO and LordNet (Shi et al., 2022) which also leverages the physics constrained loss to train the neural PDE solver. We directly use the trained U-net from the official open-source repository of (Wandel et al., 2020) for evaluation and train FNO and LordNet from scratch. The experiments in Table 1 show that LordNet outperforms the other two neural networks in the baseline setting without NeuralStagger. Therefore, we use LordNet for further experiments on the choice of spatial and temporal factors. We find that in this case, the information from the 100×100 grid ($s_H = 1, s_W = 3$) is sufficient to achieve comparable results to the U-net baseline, while larger spatial steps will introduce too much information loss. In addition, it seems increasing the temporal factors hurts the accuracy more obviously than those in the periodic boundary condition and the lid-driven boundary condition, though the accuracy is still comparable to U-net even with $s_T = 16$. We believe this is because the dataset is incrementally explored by maintaining a training pool and enriching it with the neural network's predictions during training. However, the predictions may not be accurate. As the physics constrained loss is defined on $\hat{u}_{t+(s_T-1)\Delta t}$ and $\hat{u}_{t+s_T\Delta t}$, inaccurate $\hat{u}_{t+(s_T-1)\Delta t}$ may mislead the neural network to the wrong direction. When we increase s_T, more errors will be accumulated along the sequence from $\hat{u}_t$ the $\hat{u}_{t+(s_T-1)\Delta t}$ and the training will be harder. Designing training algorithms to better support NeuralStagger remains unexplored and we leave it for future work.

In terms of speed, the choices of spatial and temporal factors lead to different levels of acceleration, as is shown in Table 1, where GMACs (multiply-accumulate Operations) per card is the average computational load of simulation for 16 timesteps. Specifically, the largest factor configuration to keep the accuracy comparable to the baseline is $s_T = 16, s_H = 1, s_W = 3$, leading to the largest decrease in GMACs per card, i.e., $1/32$ of the baseline U-net and $1/48$ of LordNet without NeuralStagger. Specifically, when tested with A100 cards, it leads to $28\times$ speed-up over U-net and $17\times$ over LordNet without NeuralStagger.

Table 1: Performance of NeuralStagger with different decomposition factors and neural networks in the flow-around-obstacles setting.

Config	Temporal factor	Spatial factors	Folded pipe	Cylinder	Time/step (ms)	GMACs per card
U-net (Wandel et al., 2020)	-	-	1.67 e-4	1.24 e-4	3.386	29.60
FNO (Li et al., 2020a)	-	-	9.12e-4	1.00e-3	1.914	13.52
LordNet (Shi et al., 2022)	-	-	8.97e-6	1.07e-5	2.003	71.04
	1	(1, 3)	1.30e-5	5.01e-5	1.826	19.84
	1	(2, 6)	6.74e-4	8.21e-3	1.758	4.46
	2	(1, 1)	3.42 e-5	4.30 e-5	1.002	35.52
	2	(1, 3)	5.51e-5	1.19e-4	0.912	9.92
	8	(1, 3)	4.67e-5	4.41e-4	0.246	2.48
	16	(1, 3)	4.48e-5	1.51e-4	0.124	1.24

4.3 Application in optimal control

To further showcase the capability of the neural solver with NeuralStagger on the inverse problem, we conduct the optimal control experiment introduced in Wandel et al. (2020). The task is to change the flow speed to control the shedding frequency of a Kármán vortex street behind an obstacle. The shedding frequency is estimated by the frequency spectrum $V(f)$ of the y-component of the velocity field behind the obstacle over 200 time steps, denoted by $E\left[|V(f)|^2\right]$. We define the loss function $L = \left(E\left[|V(f)|^2\right] - \hat{f}\right)^2$, where $\hat{f}$ is the target frequency. Then we compute the gradient of the velocity with regard to the loss by auto-differentiation through the neural solver and leverage Adam optimizer (Paszke et al., 2017; Kingma & Ba, 2014) to update the velocity. We compare the result of the learned model with the setting $s_H = 1, s_W = 3, s_T = 2$ to that shown in Wandel et al. (2020). As is shown in Figure 6, the velocity controlled by LordNet converges to the target velocity with fewer iterations.

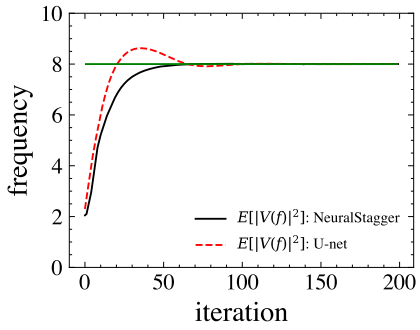

Figure 6: The optimization curve of the frequency control for vortex streets. The U-net converged after almost 72 iterations, while the LordNet using NeuralStagger converged after 55 iterations.

5 Conclusion and Limitation

We present NeuralStagger, a general framework for accelerating the neural PDE solver trained by physics constrained loss. By spatially and temporally decomposing the learning task and training multiple lightweight neural networks, the neural solver is better paralleled and much faster with sufficient computational resources. In addition, each lightweight neural network is naturally a coarse-resolution solver and they bring the flexibility of producing the solutions on multiple levels of resolution, which is important for balancing the resolution and computational resources. We discuss the choice of decomposition factors and empirically test their influence on accuracy and speed. The experiments in fluid dynamics simulation show that NeuralStagger brings an additional 10 to 100× speed-up over SOTA neural PDE solvers with mild sacrifice on accuracy.

There are also several limitations to be tackled in future works. Firstly, the accuracy drops with the growing decomposition factors. A potential solution would be introducing historical states in the neural network input to make up for the information loss. Secondly, we only define the spatial decomposition over regular meshes, while it turns to the non-trivial vertex coloring problem for irregular meshes. Heuristic coloring algorithms would be useful for this problem. Thirdly, our experiments only show the generalization to different initial conditions and boundary conditions. In the future, we would like to explore the generalization to different mesh sizes.

REFERENCES

Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Learning data-driven discretizations for partial differential equations. Proceedings of the National Academy of Sciences, 116(31):15344–15349, 2019.

Peter L Bartlett, Philip M Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. Proceedings of the National Academy of Sciences, 117(48):30063–30070, 2020.

Julius Berner, Markus Dablander, and Philipp Grohs. Numerically solving parametric families of high-dimensional kolmogorov partial differential equations via deep learning. arXiv preprint arXiv:2011.04602, 2020.

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

Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K Gupta. Clifford neural layers for pde modeling. arXiv preprint arXiv:2209.04934, 2022a.

Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. arXiv preprint arXiv:2202.03376, 2022b.

Nicholas Geneva and Nicholas Zabaras. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. Journal of Computational Physics, 403:109056, 2 2020. ISSN 00219991. doi: 10.1016/j.jcp.2019.109056. URL https://linkinghub.elsevier.com/retrieve/pii/S0021999119307612.

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. arXiv preprint arXiv:2111.13587, 2021.

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

Jun-Ting Hsieh, Shengjia Zhao, Stephan Eismann, Lucia Mirabella, and Stefano Ermon. Learning neural pde solvers with convergence guarantees. In International Conference on Learning Representations, 2019.

Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. Journal of Computational Physics, 426:109951, 2021.

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

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning accelerated computational fluid dynamics. arXiv preprint arXiv:2102.01010, 2021.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations, 2020a. URL http://arxiv.org/abs/2010.08895. arXiv: 2010.08895.

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

Lu Lu, Pengzhan Jin, and George Em Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. Nature Machine Intelligence, 3(3):218–229, March 2021a. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL http://arxiv.org/abs/1910.03193. arXiv:1910.03193 [cs, stat].

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

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. NIPS 2017 Workshop Autodiff, 2017.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. arXiv preprint arXiv:2010.03409, 2020.

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

Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science, 367(6481):1026–1030, 2020.

Pu Ren, Chengping Rao, Yang Liu, Zihan Ma, Qi Wang, Jian-Xun Wang, and Hao Sun. Physics-informed Deep Super-resolution for Spatiotemporal Data, August 2022. URL http://arxiv.org/abs/2208.01462. arXiv:2208.01462 [physics].

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

Wenlei Shi, Xinquan Huang, Xiaotian Gao, Xinran Wei, Jia Zhang, Jiang Bian, Mao Yang, and Tie-Yan Liu. LordNet: Learning to Solve Parametric Partial Differential Equations without Simulated Data, 2022. URL https://arxiv.org/abs/2206.09418. ArXiv: 2206.09418.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In International Conference on Machine Learning, pp. 3424–3433. PMLR, 2017.

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. In Advances in Neural Information Processing Systems, volume 33, pp. 6111–6122, 2020.

Nils Wandel, Michael Weinmann, and Reinhard Klein. Learning Incompressible Fluid Dynamics from Scratch – Towards Fast, Differentiable Fluid Models that Generalize, 2020. URL http://arxiv.org/abs/2006.08762. arXiv: 2006.08762.

Nils Wandel, Michael Weinmann, Michael Neidlin, and Reinhard Klein. Spline-PINN: Approaching PDEs without Data using Fast, Physics-Informed Hermite-Spline CNNs. arXiv preprint, 2021. URL http://arxiv.org/abs/2109.07143.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. arXiv preprint arXiv:2103.10974, 2021.

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

Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. Journal of Computational Physics, 394:56–81, October 2019. ISSN 00219991. doi: 10.1016/j.jcp.2019.05.024. URL https://linkinghub.elsevier.com/retrieve/pii/S0021999119303559.

O.C. Zienkiewicz, R.L. Taylor, and P. Nithiarasu. The finite element method for fluid dynamics. 6th edition. Elsevier, 2006.

6 APPENDIX

6.1 INFORMATION LOSS CAUSED BY SPATIAL DECOMPOSITION

In this section, we provide the proof to proposition 1 in the linear model setting. In this section, we will theoretically characterize the information loss caused by spatial decomposition under the linear model setting. Note that the proof is done on the 1-dimensional diffusion equation with the explicit method for ease of understanding, but as we will see, the conclusion is the same in the case with 2 dimensions or the implicit method.

We consider a simple 1d partial differential equation with Dirichlet boundary condition:

$$\partial_t u = \Delta u, x \in \Omega \tag{15}$$
$$u_t(x) = f_t(x), x \in \partial\Omega \tag{16}$$

Discretizing the function u on grid $(x_1, \cdots, x_d)$, we denote $\hat{u}^j = u(x_j)$. We consider the finite difference discretization:

$$\frac{\hat{u}_{t+1}^j - \hat{u}_t^j}{\delta t} = \frac{(\hat{u}_t^{j+1} - \hat{u}_t^j) - (\hat{u}_t^j - \hat{u}_t^{j-1})}{\delta x^2}, x_j \neq \{x_1, x_d\} \tag{17}$$
$$\hat{u}_{t+1}^j = f_{t+1}(x_j), x_j = \{x_1, x_d\} \tag{18}$$

Given the input $\hat{u}_t \in \mathbb{R}^d$ and output $\hat{u}_{t+\Delta t} \in \mathbb{R}^d$, the output $\hat{u}_{t+\Delta t}$ is parameterized by linear model as $\hat{u}_{t+\Delta t} = \hat{u}_t W$ where $W \in \mathbb{R}^{d\times d}$ denotes the learned parameters. The physics constrained loss aims to learn the parameters W^* of the linear model that satisfies:

$$W^* = \arg\min_W \frac{1}{N} \sum_{i=1}^{N} \|\hat{u}_t^i W - y^i\|^2, \tag{19}$$

where i denotes the index of training samples and $y^j = f_{t+1}(x_j), x_j = \{x_1, x_d\}$; $y^j = \hat{u}_t^j - \frac{\delta t}{\delta x^2}\left((\hat{u}_t^{j+1} - \hat{u}_t^j) - (\hat{u}_t^j - \hat{u}_t^{j-1})\right), x_j \neq \{x_1, x_d\}$.

By applying spatial decomposition, the input and output are equally partitioned into K blocks $\{\hat{u}_t^1, \cdots, \hat{u}_t^K\}$ and $\{\hat{u}_{t+\Delta t}^1, \cdots, \hat{u}_{t+\Delta t}^K\}$. Each block contains d/K coordinates Then according to the MSR loss, the optimization goal becomes:

$$W_1^*, \cdots, W_K^* = \arg\min_{W_1,\cdots,W_K} \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \|(\hat{u}_t^{i,k} W_k - y^{i,k})\|^2, \tag{20}$$

where $W_k \in \mathbb{R}^{m\times m}, m = d/K$ for $k = 1, \cdots, K$.

Proof: We first consider the case that $\sum_{i=1}^N (\hat{u}_t^{i,k})^\tau \hat{u}_t^{i,k}$ is full rank. The minimizer of Eq.(20) is $W_k^* = (\sum_{i=1}^N (\hat{u}_t^{i,k})^\tau \hat{u}_t^{i,k})^{-1} (\sum_{i=1}^N (\hat{u}_t^{i,k})^\tau y_{i,k})$. We denote the matrix $A = (\sum_{i=1}^N (\hat{u}_t^{i,k})^\tau \hat{u}_t^{i,k})^{-1}$, We construct a $d\times d$ matrix B by letting $B(k+id/K, k+jd/K) = A(i,j)$, for $i = 0, \cdots, d/K; j = 0, \cdots, d/K$; otherwise, $B(i,j) = 0$. Then it is easy to check that the matrix B is the pseudo-inverse of $\sum_{i=1}^N (\hat{u}_t^i)^\tau \hat{u}_t^i$. The minimizer of Eq.(19) is (Bartlett et al., 2020) $B(\sum_{i=1}^N (\hat{u}_i^t)^\tau y_i)$. As the matrix B only has non-zero values on the coordinates that correspond to the k-th block, we have the k-the block of W^* equals W_k^* and other blocks equal zero matrices. Denoting the matrix composed of all the samples with the bold symbol without the superscript i such as $\hat{\boldsymbol{u}}_t$ for $\{\hat{u}_t^i\}$ and $\hat{\boldsymbol{u}}_t^k$ for $\left\{\hat{u}_t^{i,k}\right\}$, we have $\sum_{i=1}^N (\hat{u}_t^{i,k})^\tau \hat{u}_t^{i,k} = (\hat{\boldsymbol{u}}_t^k)^\tau \hat{\boldsymbol{u}}_t^k$ and $\sum_{i=1}^N (\hat{u}_t^i)^\tau \hat{u}_t^i = (\hat{\boldsymbol{u}}_t)^\tau \hat{\boldsymbol{u}}_t$. By Rank–nullity theorem, it is easy to see that $rank((\hat{\boldsymbol{u}}_t)^\tau \hat{\boldsymbol{u}}_t) = rank(\hat{\boldsymbol{u}}_t)$ and $rank((\hat{\boldsymbol{u}}_t^k)^\tau \hat{\boldsymbol{u}}_t^k) = rank(\hat{\boldsymbol{u}}_t^k)$. Then we get the results in the proposition.

For the case that $\sum_{i=1}^N (\hat{u}_i^{t,k})^\tau \hat{u}_i^{t,k} \leq d/K$, we can select its maximal linearly independent group to obtain its pseudo-inverse and apply similar analyses to get the results. In the case of the implicit method, the term $\hat{u}_t^i W$ in the physics constrained loss becomes $\hat{u}_t^i WV$ where V is an invertible matrix. This also does not change the conclusion.

6.2 IMPLEMENTATION DETAILS

We implemented FNO with the original 2-dimensional version in the official repository, where we set the truncation mode to 12 and the width to 64. For the LordNet, we only stack 2 Lord modules and fix the channel count to 64 in all layers. In the position-wise embedding of the 2 Lord modules, we stack two 1×1 Convolutional layers, where the hidden embedding contains 256 and 128 channels separately, and GELU activation is used between the Convolutional layers. The implementation of Unet is based on the U-Net architecture (Ronneberger et al., 2015) with 20 hidden channels, which is consistent with that in (Wandel et al., 2020) The learning rates and training samples are described as follows. To keep out the potential influence of computational resources like cores and memory, we test the speed of NeuralStagger under the setting that each coarse-resolution solvers have sufficient resources to use. Therefore, we run each solver on Nvidia A100 GPUs with the batch size equals to 1. The time per step shown in Table 1 is calculated by dividing the inference time of the coarse-resolution solver by the temporal factor s_T. The time of decomposition and reconstruction is ignored because the operation supported by 'pixel shuffle' is super efficient. We also calculated GMACs (multiply-accumulate Operations) per card, which is the average computational load of simulation for 16 timesteps. Note that for the GMACs of FNO, we do not include the operation of Fourier transform.

Periodic Boundary Condition We generate the data with random fields to generate a periodic function on a 64×64 grid with a time-step of 1e-2 where we record the solution every time step, where the external force is fixed $f(x) = 0.1sin(2\pi(x+y)) + cos(2\pi(x+y))$. For the perioidc boundary and lid-driven boundary conditions, we use the vorticity-stream function form of Eq. 13 as the physics-constrained loss. With the Helmholtz decomposition to Eq. 13, we rewrite the Navier-Stokes equation:

$$\frac{\partial \omega}{\partial t} = \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} + \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} + \frac{1}{\text{Re}}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right) \tag{21}$$

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega, \tag{22}$$

where ω is the vorticity function, ψ is the stream function, and Re is the Reynolds number. The initial condition ω_0 is generated by random field satisfying the distribution $\mathcal{N}\left(0, 8^3(-\Delta + 64I)^{-4.0}\right)$. We use 6000 states for training. In this case, we use FNO to test NeuralStagger and decay the initial learning rate 3e-3 with a factor of 0.9 every 5000 iterations.

Lid-driven Cavity boundary condition We generate data on a 64×64 but we train the neural network to predict the values of ψ inside the boundary, which is a 2-dimensional matrix of the shape $(H-2)\times(W-2)$. The random initial conditions are generated in the same way as the periodic boundary conditions. To make the initial state consistent with the boundary condition, we solve with the numerical solver for the first $T_0 = 1.98$ and use ω_{T_0} as the initial state. We use 8000 states for training with FNO, and decay the initial learning rate 3e-3 with a factor of 0.9 every 10000 iterations.

Flow around Obstacles The data generation is the same as the setting used in (Wandel et al., 2020), where the resolution of the domain is 100×300, $\Delta t = 4, \rho = 4, \mu = 0.1$. In training, different types of environments are used including magnus, box, pipe, and wing. The locations and the velocity are variable during the training, e.g., the velocity is ranged from 0.0 to 1 m/s, the diameter of the obstacle is ranged from 10 to 40, and the coordinate x of the location is randomly from to 65 to 75 and the coordinate y of that is from 40 to 60. And then for the test, we randomly select the location and flow velocity to test and in our experiment, the Reynolds number of tests is 517. In this case, we train the model from scratch without any data for $s_T = 1$. For $s_T > 1$, we use the benchmark to pre-generate the initial sequence $\hat{u}_{0,s_T}$ for training. The learning rate is 1e-3 for Lordnet and 3e-3 for FNO, both with a factor of 0.9 every 5000 iterations. The quantitative comparison in this paper is conducted on a 100×300 grid. For the optimal control of vortex shedding experiment, the domain size is 100x300, and used the trained neural PDE solver based on the above training settings. The Reynolds number here is 880. The optimizer for both Unet and LordNet is Adam optimizer with a learning rate of 1e-3.

6.3 THE RESULTS OF THREE CASES WITH DIFFERENT SPATIAL-TEMPORAL FACTORS

Table 2: Tests on Navier-Stokes equation with periodic boundary condition.

Temporal Skipping	$L_T = 2$, (1,1) Error-1	$L_T = 2$, (1,1) Error-200	$L_T = 2$, (2,2) Error-1	$L_T = 2$, (2,2) Error-200.
1	0.0000058	0.0011939	0.0000074	0.0016352
5	0.0000297	0.0012140	0.0000308	0.0016848
10	0.0000626	0.0013126	0.0000654	0.0016719
20	0.0001074	0.0011042	0.0001321	0.0017580
30	0.0001826	0.0014243	0.0001975	0.0017017
40	0.0002501	0.0012091	0.0002545	0.0012931

Table 3: Tests on Navier-Stokes equation with Lid-driven cavity boundary condition.

Temporal Skipping	$L_T = 27$, (1,1) Error-1	$L_T = 27$, (1,1) Error-2700	$L_T = 27$, (2,2) Error-1	$L_T = 27$, (2,2) Error-2700.
1	1.82e-5	0.00228	1.78 e-5	0.00283
27	3.76e-4	0.00255	5.38 e-4	0.00390
54	8.78e-4	0.00404	1.18 e-3	0.00420
108	1.86e-3	0.00461	2.30 e-3	0.00478

Table 4: Performance of NeuralStagger with different decomposition factors and neural networks in the flow-around-obstacles setting.

Config	Temporal factor	Spatial factors	Folded pipe	Cylinder	GMACs per card
U-net (Wandel et al., 2020)	-	-	1.67 e-4	1.24 e-4	29.60
	1	(1, 3)	1.93 e-4	2.51e-4	9.76
	1	(2, 6)	4.31 e-4	7.93e-4	2.40
	2	(1, 3)	6.76 e-4	9.12e-4	4.88
	8	(1, 3)	6.43 e-4	2.02e-3	1.22
	16	(1, 3)	1.11 e-3	3.70e-3	0.61
FNO (Li et al., 2020a)	-	-	9.12e-4	1.00e-3	13.52
	1	(1, 3)	1.11 e-3	1.05e-3	4.72
	1	(2, 6)	1.84 e-3	3.70e-3	1.37
	2	(1, 3)	1.08 e-3	6.24e-4	2.36
	8	(1, 3)	1.18e-3	4.09e-3	0.59
	16	(1, 3)	1.41e-3	8.20-e3	0.30
LordNet (Shi et al., 2022)	-	-	8.97e-6	1.07e-5	71.04
	1	(1, 3)	1.30e-5	5.01e-5	19.84
	1	(2, 6)	6.74e-4	8.21e-3	4.46
	2	(1, 3)	5.51e-5	1.19e-4	9.92
	8	(1, 3)	4.67e-5	4.41e-4	2.48
	16	(1, 3)	4.48e-5	1.51e-4	1.24