Learning Neural PDE Solvers with Parameter-Guided Channel Attention

Makoto Takamoto [1] **Francesco Alesiani** [1] **Mathias Niepert** [1 2]

Abstract

Scientific Machine Learning (SciML) is concerned with the development of learned emulators of physical systems governed by partial differential equations (PDE). In application domains such as weather forecasting, molecular dynamics, and inverse design, ML-based surrogate models are increasingly used to augment or replace inefficient and often non-differentiable numerical simulation algorithms. While a number of ML-based methods for approximating the solutions of PDEs have been proposed in recent years, they typically do not adapt to the parameters of the PDEs, making it difficult to generalize to PDE parameters not seen during training. We propose a Channel Attention mechanism guided by PDE Parameter Embeddings (CAPE) component for neural surrogate models and a simple yet effective curriculum learning strategy. The CAPE module can be combined with neural PDE solvers allowing them to adapt to unseen PDE parameters. The curriculum learning strategy provides a seamless transition between teacher-forcing and fully auto-regressive training. We compare CAPE in conjunction with the curriculum learning strategy using a popular PDE benchmark and obtain consistent and significant improvements over the baseline models. The experiments also show several advantages of CAPE, such as its increased ability to generalize to unseen PDE parameters without large increases inference time and parameter count. An implementation of the method and experiments are available at https://github.com/nec-research/CAPE-ML4Sci.

[1]NEC Laboratories Europe, Heidelberg, Germany [2]University of Stuttgart, Stuttgart, Germany. Correspondence to: Makoto Takamoto <makoto.takamoto@neclab.eu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1. Introduction

Many real-world phenomena, ranging from weather forecasts to molecular dynamics and quantum systems, can be modeled with partial differential equations (PDEs). While for some problems the mathematical description of these equations is available, finding its solutions is complex and usually needs some numerical approximations. Numerical simulation methods have been developed for many years and have achieved a high level of accuracy in solving these equations. However, numerical methods are resource intensive and time-consuming even when run on larger supercomputers to obtain sufficiently accurate results. Especially high-resolution and high-dimensional hydrodynamic-type field equations are computationally demanding. Even more challenging are simulations with various PDE parameters since a numerical simulation is required for each of the initial conditions and for each of the PDE parameter's configurations.

Recently, there has been a rapidly growing interest in machine learning methods for the problem of solving PDEs due to their various applications in science and engineering (Guo et al., 2016; Lusch et al., 2018; Sirignano & Spiliopoulos, 2018; Raissi, 2018; Kim et al., 2019; Hsieh et al., 2019; Bar-Sinai et al., 2019; Bhatnagar et al., 2019; Pfaff et al., 2020; Wang et al., 2020; Khoo et al., 2021). For example, several prior studies reported that ML models can estimate solutions more efficiently than classical numerical simulators (Li et al., 2021a; Stachenfeld et al., 2021). Moreover, using neural networks as surrogate models allows us to compute derivatives with respect to the input variables. Differentiable surrogate models enable the use of automatic differentiation to solve inverse problems which have numerous real-world applications but are difficult to solve using traditional numerical methods (Coros et al., 2013; Allen et al., 2022). A considerable number of papers have shown the advantage of ML-based surrogate models (Li et al., 2020; 2021a; Stachenfeld et al., 2021; Lu et al., 2021). The majority of these methods, however, are purely data-driven, which does not allow us to change PDE parameters. The existing approaches taking PDE parameters into account, are tailored to specific neural network architectures. For instance, message-passing PDE solvers (Brandstetter et al., 2022) use PDE parameters as input but transform these parameters into node embedding and, therefore, cannot be

used with other methods, in particular, CNN based models. To overcome the shortcomings of existing data-driven SciML models, a straightforward approach would include the PDE parameters as additional input. However, this naive method requires modification of the BASE network which is potentially harmful to its accuracy. An alternative approach attaches an external parameter embedding module to the network. However, there are too many possible module structures and methods to provide the embedded parameter information to the base network, and it is in general non-trivial to select the best one. Still, to ensure a direct comparison, we implemented an extension of said node embedding method to convolution-based methods.

We propose a new and effective parameter embedding module by utilizing the channel-attention method inspired by classical numerical solvers using the implicit discretization method (see Sec. 2.3 and Sec. 2.4). The crucial idea is that a neural network generates intermediate (approximated) field data for future time steps which are then interpolated by a BASE model such as the FNO (Li et al., 2021a) to predict the field data for the next time step. CAPE can be combined with any existing autoregressive neural PDE solvers. Fig. 1 illustrates the proposed CAPE framework.

We make the following contributions. First, we propose a CAPE module which can be combined with any existing neural PDE solvers and can effectively transfer the PDE parameter information to the base network (BASE). Second, we propose a simple but effective curriculum learning strategy that seamlessly bridges the teacher-forcing and auto-regressive methods. Third, we perform extensive experiments using various PDEs with a large number of different parameters evaluating the effectiveness and efficiency of the proposed method in comparison with state-of-the-art methods.

2. CAPE: A Framework for Neural PDE Solvers

2.1. Background: Partial Differential Equations

Following the notation by (Brandstetter et al., 2022), we consider Partial Differential Equations (PDEs) over time dimension $t \in [0, T]$ and over spatial dimensions $\boldsymbol{x} = [x_1, \dots, x_D] \in \mathbb{X} \subseteq \mathbb{R}^D$ which can be written as

$$\partial_t \boldsymbol{u} = F(t, \boldsymbol{x}, \boldsymbol{u}, \partial_{\boldsymbol{x}} \boldsymbol{u}, \partial_{\boldsymbol{x},\boldsymbol{x}} \boldsymbol{u}, \dots), \quad (1)$$

$$\boldsymbol{u}(0, \boldsymbol{x}) = \boldsymbol{u}^0(\boldsymbol{x}),\ \boldsymbol{x} \in \mathbb{X}, \quad B[\boldsymbol{u}](t, \boldsymbol{x}) = 0, \quad (2)$$

where $(t, \boldsymbol{x}) \in [0, T] \times \partial\mathbb{X}$, and $\boldsymbol{u} : [0, T] \times \mathbb{X} \to \mathbb{R}^c$ is the solution of the PDE, where c is the field dimension, used to describe various field quantities such as velocity, pressure, and density, while $\boldsymbol{u}^0(x)$ is the initial condition at time $t = 0$, and $B[\boldsymbol{u}](t, \boldsymbol{x}) = 0$ are the boundary conditions at $\boldsymbol{x}$ in $\partial\mathbb{X}$, which is the boundary of the domain $\mathbb{X}$. Here, $\partial_{\boldsymbol{x}} \boldsymbol{u}, \partial_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{u}, \dots$ are the partial derivatives of the solution $\boldsymbol{u}$ with respect to the domain, while $\partial_t \boldsymbol{u}$ is the partial derivative with respect to time. The functional F describes the possibly non-linear interactions between the PDE's terms.

2.2. Problem Definition

We consider PDEs (Sec. 2.1) whose solution is described as a temporal sequence of field data $\{\boldsymbol{u}^k\}_{k=0,\dots,N} := \boldsymbol{u}^0, \boldsymbol{u}^1, \dots, \boldsymbol{u}^N$ where $\boldsymbol{u}^k$ is the field data at time step t_k, that is, the state of the physical system governed by the PDE under consideration at time t_k discretized using $\Delta t = T/N$. Each $\boldsymbol{u} \in \mathcal{X} \subseteq \mathbb{R}^{c \times x_1, \dots, x_D}$ represents the field tensor data with c, the number of physical variables such as density and velocity, and x_i the spatial dimensions of the i-th coordinate. For example, for a 1-d problem we have $\mathcal{X} \subseteq \mathbb{R}^{c \times x_1}$, for a 2-d problem $\mathcal{X} \subseteq \mathbb{R}^{c \times x_1 \times x_2}$, and for a 3-d problem $\mathcal{X} \subseteq \mathbb{R}^{c \times x_1 \times x_2 \times x_3}$. We will often refer to c as the channel dimension. We aim to emulate numerical simulators of PDEs which iteratively map $\mathcal{M} : \mathcal{X} \to \mathcal{X}$ from $\boldsymbol{u}^k$ to $\boldsymbol{u}^{k+1}$. The emulator (or surrogate model) is a learnable function modeled as a neural network NN with weights $\boldsymbol{\theta}$. We refer to the parameters of a neural network as weights to avoid a conflict in terminology with the parameters of PDEs. In the following, we denote the emulator's prediction at time index k as $\tilde{\boldsymbol{u}}^k$. Auto-regressive neural networks predict the next time step's field data based on a sequence of field data tensors of length ℓ

$$\tilde{\boldsymbol{u}}^{k+1} = \mathrm{NN}(\tilde{\boldsymbol{u}}^{k-\ell+1}, \dots, \tilde{\boldsymbol{u}}^k; \boldsymbol{\theta}).$$

Given the length of the input sequence $N \in \mathbb{N}$, and an initial input sequence $\left(\boldsymbol{u}^0, \dots, \boldsymbol{u}^{\ell-1}\right) = \left(\tilde{\boldsymbol{u}}^0, \dots, \tilde{\boldsymbol{u}}^{\ell-1}\right)$ of length $\ell < N$, the ML model auto-regressively generates the remaining sequence $\left(\tilde{\boldsymbol{u}}^\ell, \dots, \tilde{\boldsymbol{u}}^N\right)$. The training loss is typically the normalised root-mean-square error (RMSE) between the predicted and the true field data tensors

$$\mathbf{L}(\boldsymbol{\theta}) = \sum_{k=\ell}^{N} \mathrm{nRMSE}\left(\boldsymbol{u}^k,\ \tilde{\boldsymbol{u}}^k\right) \equiv \sum_{k=\ell}^{N} \frac{||\tilde{\boldsymbol{u}}^k - \boldsymbol{u}^k||_2}{||\boldsymbol{u}^k||_2}, \quad (3)$$

where $||u||_2$ is the L_2-norm of a (vector-valued) variable u. Since we are training an auto-regressive neural network, the gradients of the above loss can be backpropagated in time in various ways. We discuss this in the following sections. Figure 1(left) illustrates this auto-regressive approach to solving PDEs. In the vast majority of experimental setups, the assumption is made that $\ell > 1$, and, therefore, an initial input sequence of length ℓ is available to the model; in practice, this would require a numerical simulation to be run for $\ell - 1$-time steps from the initial condition and for each PDE parameter $\boldsymbol{\lambda}$. The main idea of CAPE is to learn to generate these sequences based on the current field data and parameter values $\boldsymbol{\lambda}$ and use those as input to an off-the-shelf neural surrogate model such as an FNO (Li et al., 2021a) or

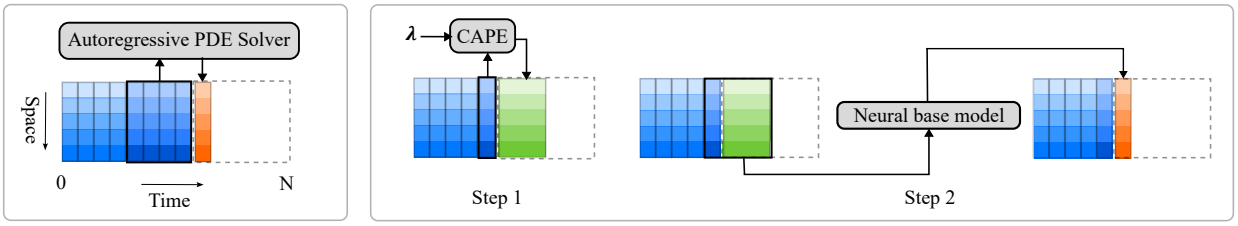

Figure 1. The standard autoregressive approach (left) and the proposed CAPE approach (right) which consists of two interdependent steps.

an U-Net (Ronneberger et al., 2015) to perform a complex interpolation.

2.3. Combining Neural PDE Solvers with the CAPE Module

The proposed approach is motivated by the need for neural PDE solvers to generalize to PDE parameters unseen during training. We propose CAPE, a novel neural network architecture that takes the prior state of the system $\tilde{\boldsymbol{u}}^k$ and PDE parameters $\boldsymbol{\lambda}$ as input and predicts the ℓ-*intermediate* future states $\{\hat{\boldsymbol{u}}_{\text{cape}}^{k\to k+i}\}_{i=1,\dots,\ell} = \text{CAPE}(\boldsymbol{u}^k, \boldsymbol{\lambda}; \boldsymbol{\theta}_{\text{CAPE}})$[1]. The output of CAPE is then used by the BASE network. The overall structure is provided in Fig. 1(right). The intuition behind this approach is that the intermediate future states capture information about the PDE parameters' impact by attending to the results of the convolutional operations. While we do not change the architecture of the base neural PDE solvers, we propose to use them to predict, given the past temporal states and the intermediate future states, the state for the next time step. This is contrary to the typical use of neural PDE solvers. The base network is trained jointly with the CAPE module. As shown in Sec. 3, this choice improves the prediction capability of the BASE network.

During training, the output of CAPE is augmented with an the additional loss term

$$\mathbf{L}_{\text{cape}}(\boldsymbol{\theta}_{\text{CAPE}}) = \sum_{k=\ell}^{N} \sum_{i=1}^{\min(\ell, N-k)} \text{nRMSE}\left(\hat{\boldsymbol{u}}_{\text{cape}}^{k\to k+i}, \boldsymbol{u}^{k+i}\right),$$

which forces the CAPE module to predict a temporal sequence of future field data $\{\boldsymbol{u}^{k+i}\}_{i=1,\dots,\ell}$.

Finally, the intermediate sequence $\{\hat{\boldsymbol{u}}_{\text{cape}}^{k\to k+i}\}_{i=1,\dots,\ell}$ is concatenated with $\boldsymbol{u}^k$, the field data at time t_k, and given to the base network to make the final prediction. In summary, the CAPE module transforms the input variables $\{\boldsymbol{u}^k, \boldsymbol{\lambda}\}$ into temporal-sequential intermediate field data $\{\boldsymbol{u}^k, \hat{\boldsymbol{u}}_{\text{cape}}^{k\to k+1}, \dots, \hat{\boldsymbol{u}}_{\text{cape}}^{k\to k+\ell}\}$ which is then interpolated by the base neural network. Before we introduce the inductive bias of the CAPE module, we motivate the general approach from a classical numerical simulation perspective.

[1]In principle it is possible for CAPE to also predict field data of past time steps: $\{\hat{\boldsymbol{u}}_{\text{cape}}^{k\to k+i}\}_{i=\pm1,\dots,\pm\ell}$.

CAPE as an implicit discretization method. Let us consider the following simple PDE:

$$\partial_t \boldsymbol{u} = F(\boldsymbol{u}; \boldsymbol{\lambda}). \tag{4}$$

The base neural network can be expressed with the equation

$$\boldsymbol{u}^{k+1} = f_{\text{base}}\left(\boldsymbol{u}^k, \{\hat{\boldsymbol{u}}_{\text{cape}}^{k\to k+i}\}_{i=1,..,\ell}; \boldsymbol{\theta}_{\text{BASE}}\right), \tag{5}$$

where f_{base} is the function expressed by the base neural surrogate model and

$$\{\hat{\boldsymbol{u}}_{\text{cape}}^{k\to k+i}\}_{i=1,\dots,\ell} = \text{CAPE}(\boldsymbol{u}^k, \boldsymbol{\lambda}; \boldsymbol{\theta}_{\text{CAPE}}) \tag{6}$$

are the approximated intermediate future states predicted by the CAPE module generated for time index k. The CAPE module acts as a pre-processor network providing sequential data $\boldsymbol{u}^k, \{\boldsymbol{u}^{k+i}\}_{i=1,\dots,\ell}$, while Eq. 5 can be interpreted as an interpolation network whose input is the current state of the physical system $\boldsymbol{u}^k$ and future extrapolated states of the system $\{\hat{\boldsymbol{u}}_{\text{cape}}^{k+i}\}_{i=1,\dots,\ell}$ (see the last figure of Fig. 5).

Now, these equations can be understood from a numerical method perspective, where using implicit discretization (Anderson et al., 2016), Eq. 4 reduces to:

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \Delta t F(\boldsymbol{u}^{k+1}; \boldsymbol{\lambda}) \equiv \tilde{F}(\boldsymbol{u}^k, \boldsymbol{u}^{k+1}; \boldsymbol{\lambda}). \tag{7}$$

Eq. 5 with $\ell = 1$ can be seen as a neural network approximation of the function $\tilde{F}$ used withing the implicit method of Eq. 7. When $\ell > 1$ CAPE can therefore be seen as a generalized ML-based variant of the implicit method for solving PDEs.

Generally, the implicit method is known to be independent of the Courant–Friedrichs–Lewy (CFL) condition (Lewy et al., 1928) which enables it to utilize a larger time step size, Δt, than the explicit method. However, the implicit method usually necessitates either an iterative method or a computationally expensive matrix inversion to obtain the value of u^{k+1} due to the existence of the unknown variable on the right-hand side of the equation. In contrast, our CAPE method allows us to evaluate u^{k+1} in Eq. 7 in a data-driven manner. We consider that our approach can

potentially result in more accurate and stable predictions of u^{k+1} even using a large time step size by reflecting the implicit method's behavior but with significantly reduced numerical costs.

2.4. PDE Parameter-Guided Channel Attention

CAPE computes 3 different d-dimensional channel attention masks $\boldsymbol{a}_\alpha \in \mathbb{R}^d, \alpha = 1, 2, 3$ from the parameters of the PDE $\boldsymbol{\lambda}$ using a 2-layer MLP

$$\boldsymbol{a}_\alpha = \boldsymbol{W}_{2,\alpha}\sigma(\boldsymbol{W}_{1,\alpha}\boldsymbol{\lambda}), \tag{8}$$

where $\boldsymbol{W}_{1/2,\alpha}$ is the weight matrics of the MLPs , d is the channel dimension in the feature space and σ is the GeLU activation function (Hendrycks & Gimpel, 2016). Here the bias terms are omitted for simplicity . $\boldsymbol{W}_\alpha = (\boldsymbol{W}_{1,\alpha}, \boldsymbol{W}_{2,\alpha})$ are the weights associated with three operators: a 1×1-convolution (g_1), a depth-wise convolution (g_2), and a spectral convolution (Li et al., 2021a) (g_3), that are used to compute the tensor representations $\boldsymbol{z}_\alpha^k \in \mathbb{R}^{d \times n_x \cdots}$ as $\boldsymbol{z}_\alpha^k = g_\alpha(\boldsymbol{u}^k, \boldsymbol{W}_\alpha)$. The tensors are then multiplied by the attention

$$\boldsymbol{v}_\alpha^k = \boldsymbol{a}_\alpha^k \odot_1 \boldsymbol{z}_\alpha^k \tag{9}$$

using the Hadamard operator ($\odot_1$) over the first dimension (the channel dimension) which is equivalent to the broadcast operation of ML programming languages, such as Numpy and PyTorch . The three convolutions can be interpreted as a finite difference method since convolution operations accumulate local information of a mesh, which, in principle, can simulate local interactions such as advection and diffusion. Intuitively, channel attention is equivalent to choosing an appropriate physical process for each PDE parameter. A similar mechanism has been proposed for visual tasks, called the squeeze-and-excitation networks (Hu et al., 2018) which enhances useful channels of the feature vector of convolutional networks through an attention mechanism. The feature $\boldsymbol{v}_\alpha^k \in \mathbb{R}^{d \times n_x \cdots}, \alpha = 1, , 2, 3$ are combined to form an intermediate feature $\boldsymbol{y}^k \in \mathbb{R}^{c \times \ell \times n_x \cdots}$ as

$$\boldsymbol{y}^k = h_{1\times 1, d \to c \times \ell}\left(\sigma\left(h_{1\times 1, c\to d}(\boldsymbol{u}^k) + \sum_\alpha \boldsymbol{v}_\alpha^k\right)\right) \tag{10}$$

where $h_{1\times 1,*}$ are 1×1 convolutions that adjust the number of dimensions, in particular $h_{1\times 1, c\to d} : c \times n_x \cdots \to d \times n_x \ldots$, while $h_{1\times 1, d\to c\times \ell} : d \times n_x \cdots \to c \times \ell \times n_x \ldots$. Finally, the sequence of predictions is computed

$$\left\{\boldsymbol{u}_{\text{cape}}^{k\to k+i}\right\}_{i=1,\ldots,\ell} = (\boldsymbol{u}^k + \text{LayerNorm}(\boldsymbol{y}_i^k))_{i=1,\ldots,\ell} \tag{11}$$

where $\boldsymbol{y}_i^k$ is the i-th element of the data tensor $\boldsymbol{y}^k$, selected from the second dimension [2]. For the sake of presentation, we omitted the batch dimension. Figure 2 illustrates the architecture of the CAPE module [3]. Visualization of the kernel after channel-attention and a short discussion of the role of the channel-attention in CAPE is provided in Appendix I.

2.5. Curriculum Learning

For each initial condition $\boldsymbol{u}^0$, the BASE and CAPE models (referred to as $\text{NN}(\boldsymbol{u}^k; \boldsymbol{\theta})$) jointly predict the full temporal sequence $\{\boldsymbol{u}^k\}_{k=1,..,N}$. We propose the following *curriculum learning* strategy, Fig. 3. For each training epoch, we split the temporal sequence into two parts. For the first part $(\boldsymbol{u}^0, ..., \boldsymbol{u}^{k_{\text{trans}}})$ we use *auto-regressive* training using $\tilde{\boldsymbol{u}}^k$, the prediction of the model at time index k, as input to predict the solution for time index $k+1$, that is, $\tilde{\boldsymbol{u}}^{k+1} = \text{NN}(\tilde{\boldsymbol{u}}^k; \boldsymbol{\theta})$. For the second part $(\boldsymbol{u}^{k_{\text{trans}}+1}, ..., \boldsymbol{u}^N)$, we train the model using *teacher-forcing*. The teacher-forcing strategy (Williams & Zipser, 1989; Bengio et al., 2015) computes the prediction for time index $k+1$ using a noisy version of the true value at time index k, that is, $\tilde{\boldsymbol{u}}^{k+1} = \text{NN}(\boldsymbol{u}^k + \epsilon; \boldsymbol{\theta})$ where ϵ is random noise increasing the stability at inference time (Sanchez-Gonzalez et al., 2018; 2020; Pfaff et al., 2020; Stachenfeld et al., 2021). The time index k_{trans} determines the time step where we switch from auto-regressive training to teacher-forcing and is computed using the following monotonically increasing function of the epoch number n

$$k_{\text{trans}} = \left\lfloor \frac{N}{2}\left(1 + \tanh\left[\frac{\frac{n}{N} - 0.5}{\Delta}\right]\right)\right\rfloor, \tag{12}$$

where N is the total epoch number, and Δ is a hyper-parameter controlling the steepness of the transition function. A plot of the function is provided in Fig. 7) and a detailed algorithm of the training strategy is provided in Appendix (Algorithm 1).

The strategy is based on the following two assumptions: (1) the prediction error decreases as the number of training epochs increases, (2) the accumulated error increases as the number of auto-regressive rollout steps increases. Teacher-forcing training is usually more stable since it avoids the accumulation of prediction errors and should be used exclusively in the first phase of training. The auto-regressive strategy simulates the behavior at test time and exposes the model to inputs that evolved further from the true data, making it more robust to error accumulation. For the same reasons, however, it tends to be less stable, especially in the early phase of training. The proposed curriculum-learning strategy is used to combine the advantages of both ap-

[2] We found that in the case of FNO, the LayerNormalization in Eq. 11 is harmful (see also Tab. 15). We also found that in the case of 2D NS we obtain improved results when modifying the right-hand side of Eq. 11 as: $\boldsymbol{u}^k(1 + \text{LN}(\boldsymbol{y}_i^k))$ and we applied it to obtain the 2D NS results in this paper.

[3] The results with other possible CAPE structures are provided in Appendix G which shows our proposed structure is the best choice.

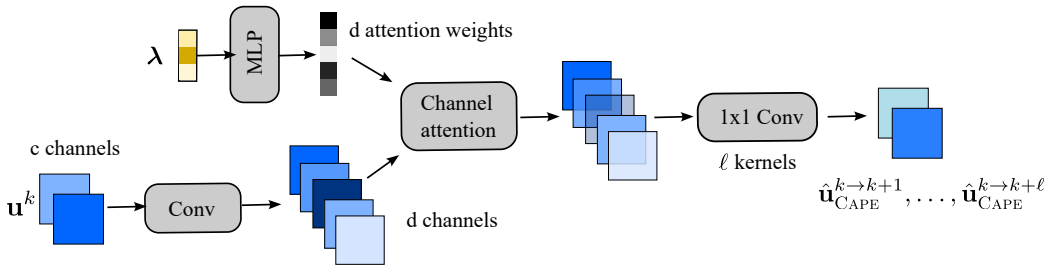

Figure 2. The CAPE module for one type of convolution (residual connections are omitted).

PDE	training parameters	testing (unseen) parameters in Fig. 6
1D Advection	$\beta = (0.2, 0.4, 0.7, 2.0, 4.0)$	$\beta = (0.1, 1.0, 7.0)$
1D Burgers	$\nu = (0.002, 0.007, 0.02, 0.04, 0.2, 0.4, 2.0)$	$\nu = (0.001, 0.01, 0.1, 1.0, 4.0)$
2D NS	$\eta = \zeta = (10^{-8}, 0.001, 0.004, 0.01, 0.04, 0.1)$	$\eta = \zeta = (0.007, 0.07)$

Table 1. PDE parameters used in the experiments.

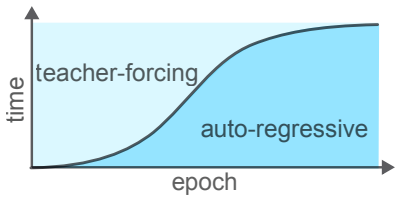

Figure 3. The proposed curriculum learning strategy leads to a smooth transition between one time-step learning (*teacher-forcing*) and fully autoregressive training (*auto-regressive*).

proaches.

3. Experiments

We used datasets provided by PDEBench (Pradita et al., 2022) a benchmark for SciML from which we selected the following PDEs [4]:

1D Advection Equation. This equation describes the pure-advection of waves

$$\partial_t u(t,x) + \beta \partial_x u(t,x) = 0, \tag{13}$$

where β is the PDE parameter describing advection velocity. The exact solution of this equation is: $u(t,x) = u_0(t, x - \beta t)$ where u_0 is the initial condition. Hence, this PDE can be used to check if the ML models understand the property of advection, updating the solution by just advecting the initial profile without changing it.

1D Burgers Equation. Burgers' equation is a mathematical model equation simulating the non-linearity and diffusivity in the hydrodynamic equation by a scalar variable

$$\partial_t u(t,x) + u(t,x)\partial_x u(t,x) = \nu/\pi \partial_{xx} u(t,x), \tag{14}$$

[4] An additional experiment results conducted on 2D Burgers equation is provided in Appendix C.

where ν is the diffusion coefficient and the parameter of this equation. This PDE can be used to check if ML models can understand the non-linear behavior from the second term in the left-hand side of Eq. 14 and the diffusion process whose strength is controlled by the parameter ν.

2D Compressible Navier-Stokes Equations (2D NS). The compressible Navier-Stokes equations (NS eqs., in the following) is one of the basic physics equations describing classical fluid dynamics

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{15}$$

$$\rho(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \eta \triangle \mathbf{v} + (\zeta + \eta/3)\nabla(\nabla \cdot \mathbf{v}), \tag{16}$$

$$\partial_t \left[\epsilon + \frac{\rho v^2}{2}\right] + \nabla \cdot \left[\left(\epsilon + p + \frac{\rho v^2}{2}\right)\mathbf{v} - \mathbf{v} \cdot \sigma'\right] = 0, \tag{17}$$

where ρ is the mass density, $\mathbf{v}$ is the velocity, p is the gas pressure, $\epsilon = p/(\Gamma - 1)$ is the internal energy, $\Gamma = 5/3$, σ' is the viscous stress tensor, and η, ζ are the shear and bulk viscosity, respectively.

For 1-dimensional PDEs, we used $N = 9000$ training instances and 1000 test instances for each PDE parameter with resolution 128. For 2-dimensional NS equations, we used $N = 900$ training instances and 100 test instances for each PDE parameter with spatial resolution 64×64.

Experiment Setup. We evaluated the neural models U-Net (Ronneberger et al., 2015) and FNO (Li et al., 2021a) with datasets provided by PDEBench (Takamoto et al., 2022) for various parameters for the 1D Advection equation, 1D Burgers equation, and 2D compressible Navier-Stokes equations. We also evaluated the message passing neural PDE Solvers (MPNN) (Brandstetter et al., 2022) as a baseline allowing conditional treatment of PDE parameters. As a baseline for multi-dimensional turbulent flow, we evaluated

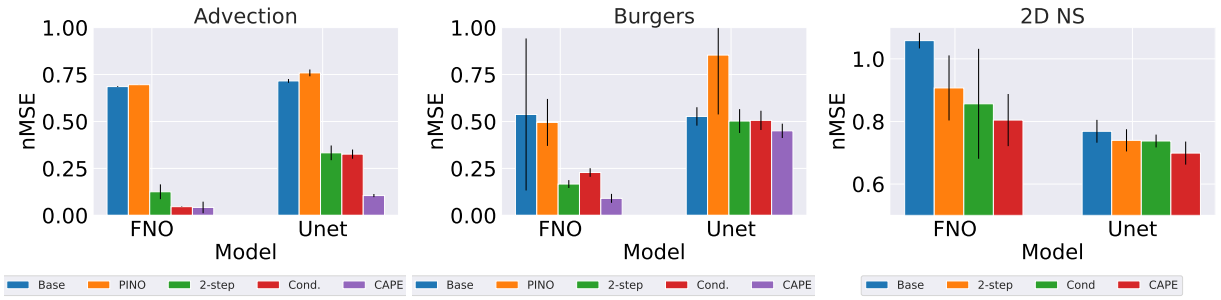

Figure 4. Plots of the normalized MSE (smaller is better) with an error bar for Advection eq. (Left), Burgers eq. (Middle), and 2D Compressible NS equations (Right).

PDE	model	BASE	BASE (PINO)	Conditional	prev. 2-steps	CAPE
1D Advection	FNO	$0.69^{\pm 2.2\times10^{-3}}$	$0.70^{\pm 1.6\times10^{-4}}$	$0.05^{\pm 1.2\times10^{-3}}$	$0.13^{\pm 3.9\times10^{-2}}$	$\mathbf{0.03}^{\pm 2.5\times10^{-3}}$
	Unet	$0.72^{\pm 1.0\times10^{-2}}$	$0.76^{\pm 1.8\times10^{-2}}$	$0.33^{\pm 2.0\times10^{-2}}$	$0.33^{\pm 3.9\times10^{-2}}$	$\mathbf{0.11}^{\pm 8.3\times10^{-3}}$
	MPNN	$0.32^{\pm 2.5\times10^{-2}}$	–	$0.07^{\pm 2.0\times10^{-3}}$	–	–
1D Burgers	FNO	$0.54^{\pm 0.40}$	$0.49^{\pm 1.3\times10^{-1}}$	$0.23^{\pm 2.3\times10^{-2}}$	$0.17^{\pm 2.1\times10^{-2}}$	$\mathbf{0.09}^{\pm 7.8\times10^{-3}}$
	Unet	$0.53^{\pm 4.9\times10^{-2}}$	$0.85^{\pm 3.2\times10^{-1}}$	$0.51^{\pm 5.1\times10^{-2}}$	$0.50^{\pm 6.3\times10^{-2}}$	$\mathbf{0.45}^{\pm 3.8\times10^{-2}}$
	MPNN	$0.27^{\pm 0.11}$	–	$0.13^{\pm 3.7\times10^{-3}}$	–	–
2D NS	FNO	$1.06^{\pm 2.5\times10^{-2}}$	–	$0.86^{\pm 0.18}$	$0.91^{\pm 0.10}$	$\mathbf{0.80}^{\pm 8.3\times10^{-2}}$
	Unet	$0.77^{\pm 3.6\times10^{-2}}$	–	$0.74^{\pm 0.02}$	$0.74^{\pm 3.5\times10^{-2}}$	$\mathbf{0.70}^{\pm 3.7\times10^{-2}}$
	MPNN	N/A	–	N/A	–	–
	TF-Net	N/A	N/A	N/A	$1.28^{\pm 0.19}$	N/A

Table 2. List of the normalized RMSE (the smaller, the better) for Advection eq., Burgers eq., and 2D Compressible NS equations.

the performance of the TF-Net (Wang et al., 2020) on 2D compressible Navier-Stokes equations. We trained each of the neural models (1) **Base**: without any changes (vanilla model), (2) **PINO**: with a PINO loss (Li et al., 2021b), (3) **Conditional**: the parameters are added to the input data as new channel-dimensions, (4) **2-step**: with the field data for the current and previous time-steps as input $(\boldsymbol{u}^k, \boldsymbol{u}^{k-1})$, and (5) **CAPE**: with the CAPE module. Other than case (4), we only provided field data for one time step to the models and, therefore, the models cannot obtain PDE parameters' information from the given data. The PINO loss function regularizes the ML models to satisfy the residuals of the PDEs and might lead to an improved generalization behavior for unseen PDE parameters. The CAPE module predicts intermediate field data for one future time step and this is used as the input to the BASE model together with the field data of the current time step.[5]. The amount of field data provided to the BASE network in cases (4) and (5) is the same: in case (4) the model always obtains field data for time steps k and $k-1$ as input to predict the field data for time step $k+1$ while in case (5) the BASE model obtains field data for time step k and intermediate field data for time step $k+1$ generated by the CAPE model to predict the field data for time step $k+1$. Hence, the model in case (4) obtains two true field data for 2 initial steps and, therefore, has the opportunity to adapt to different PDE parameter values. Hence, while case (4) is more expensive, we consider it a strong baseline for the problem.

Since the solutions of each PDE are not normalized and based on prior results on evaluating PDE solvers (Pradita et al., 2022), we measure the normalized RMSE (nRMSE). We used the normalized RMSE loss function L_{nRMSE} with the auxiliary loss function of the CAPE module $\mathbf{L}_{\text{cape}} := \mathbf{L}_{\text{nMSE}} + \alpha \mathbf{L}_{\text{cape}}$ where α is the weight coefficient. The optimization was performed with Adam (Kingma & Ba) for 100 epochs. The learning rate was divided by 2.0 every 20 epochs. For a fair comparison, we made the model size of the different methods as similar as possible. A table with model parameter sizes is provided in Tab. 7 in the appendix. A more detailed description of the hyper-parameters is provided in Appendix B [6].

Varying PDE parameters. Fig. 4 shows bar plots comparing the BASE models with and without CAPE module, the models with PINO loss, and the models with the 2-steps as

[5]First introduced in (Li et al., 2021a) with twenty steps as input.

[6]CAPE's hyper-parameters was determined using a smaller train-validation split, details of which are provided in Sec. B.2.

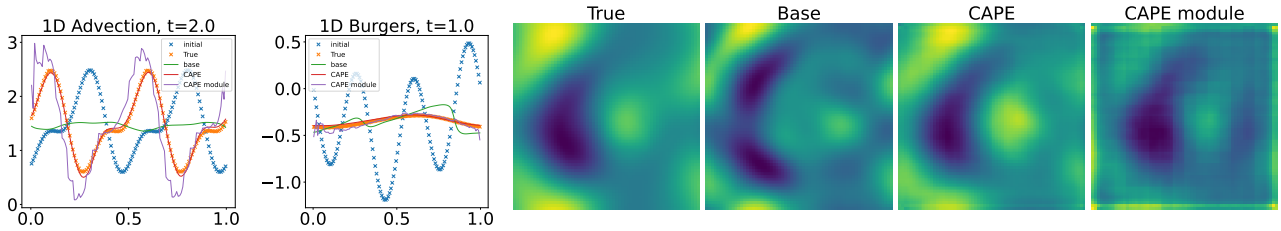

Figure 5. Visualization of the results: Advection eq. at the final time-step ($t = 2.0$) (Left), Burgers eq. at $t_k = 20(t = 1.0)$ (2nd-left) at the final time-step, and V_x of 2D NS equations at $t_k = 5(t = 0.25)$ (Right). Here "Base" is the vanilla FNO, "CAPE " is the FNO with CAPE, "CAPE module" is the direct output from CAPE module only; the CAPE module provides a higher frequency proposal to the BASE model which then more accurately predicts the field data.

input (see also Tab. 2)[7] . The CAPE module results in the lowest error in all cases. In particular, the CAPE module leads to an impressible error reduction ranging from 20 % (2D NS equation) to 95 % (1D Advection). We partly attribute this to the BASE network's ability to capture physical dynamics from the PDE parameter-dependent data provided by the CAPE module. The vanilla FNO is a state-of-the-art model and is superior to the U-net as a BASE model [8]. Interestingly, the PINO loss provides almost no benefit in our setting. We hypothesize that the PINO loss is heavily affected by and dependent on the time-step size. A more detailed explanation of this observation is given in Appendix D. Interestingly, the CAPE module provides either comparable or a little better results than the case with 2-step information. This indicates that the CAPE module succeeded in providing equivalent and even more useful information to the BASE network.

Generalization Ability. Fig. 6 plots the normalized MSE for each parameter value of the PDEs using FNO as the BASE network. The parameter of the 1D Advection PDE controls the advection velocity and the parameters of the remaining equations control the strength of the diffusion process. First, the 1D-Advection result shows that the CAPE module overfits with the trained parameter ($\beta = 0.2, 0.4, 0.7, 2.0, 4.0$), though it showed a very nice generalization performance to the trained PDE parameters. This could also be indicated by the fact that only the CAPE result for the 1D Advection equation showed a much lower error than the approach receiving the 2-step field data which in theory provides a similar amount of information as the CAPE module. Hence, to evaluate generalization to the unseen PDE parameters, we expect the difference of these errors to be correlated with the CAPE module overfitting to the trained parameters.

On the other hand, the other cases (the parameter describes the diffusion process) showed a good generalization to unseen PDE parameters. Note that the plots also indicate that vanilla models show a preference for the parameter regime; in all the cases, the vanilla models exhibit better results on smaller diffusion coefficients but lose accuracy as the diffusion coefficients increase.

Model	Ablation	nMSE	
FNO	curriculum strategy	$\mathbf{8.0 \times 10^{-1}}$	
	fully autoregressive	$1.3 \times 10^{+0}$	(+0.5)
	only teacher-forcing	$3.2 \times 10^{+0}$	(**+2.4**)
Unet	curriculum strategy	$\mathbf{7.0 \times 10^{-1}}$	
	fully autoregressive	$1.0 \times 10^{+0}$	(**+0.3**)
	only teacher-forcing	$1.0 \times 10^{+0}$	(**+0.3**)

Table 3. Ablation study for the 2D CFD equations with FNO and Unet as BASE model.

Ablation experiments. In this section, we performed an ablation study to separate the impact of the curriculum learning strategy. The ablation study on CAPE structure is provided in Tab. 15 in Appendix. Tab. 3 lists the result for the 2D NS equations using FNO and Unet as the BASE network. The proposed curriculum learning strategy drastically impacts the accuracy of the model in all cases, indicating the effectiveness of seamlessly bridging teacher-forcing and auto-regressive training. The full ablation study results are provided in Tab. 14 in Appendix.

Qualitative analysis of the CAPE module. In Fig. 5 we plot some representative outputs of the vanilla FNO, the CAPE module, and the overall CAPE model, and compare them with the true solutions. Interestingly, we can see that the BASE network often interpolates a higher noise approximation of the CAPE module into the typical shape (style) of

[7]The N/A for the results of 2D MPNN are because only 1D examples are provided in the official repository. Also TF-Net assumes to include more than one temporal steps, so the results other than "prev. 2-steps" becomes N/A.

[8]For the 2D NS PDE, the U-net achieves a smaller error than the FNO. We hypothesize that this is partly due to the difference in model size (the number of weights of the U-net is nearly 10 times larger) and partly because the U-net typically excels at image-to-image mapping problems.

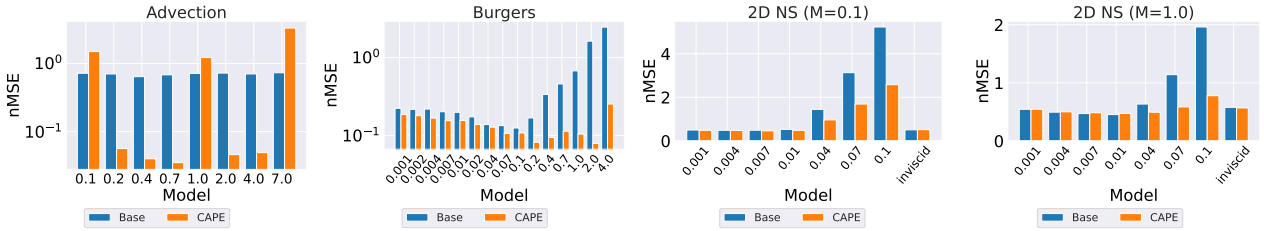

Figure 6. Plots of the normalized MSE (smaller is better) in terms of each PDE parameter for Advection eq. (Left), Burgers eq. (Middle-Left), 2D NS with $M = 0.1$ (Middle-right), and 2D NS with $M = 1$ (Right). The results were obtained using the FNO.

the final solution.

Inference time. We provide the comparison of the inference time between the hybrid approach (including 2 initial steps as input as done in the FNO paper) and CAPE (only using the initial step as the input). Here we only consider a scenario where the initial time steps are obtained from a numerical simulation. Tab. 4 lists the inference time for solving the 2D NS equation. The inference time of the FNO with the CAPE module is much shorter than the hybrid method where the inference time is dominated by the simulation time [9].

PDE	Resolution	Total Inference time [sec]	simulation time [sec]
Simulation + FNO	512^2	582.8	582.6
CAPE + FNO	512^2	1.3	–

Table 4. Inference time comparison of simulation (initial steps=10) + FNO and CAPE + FNO in the case of 2D CFD ($\eta = \zeta = 0.1$). The time-step size is $\Delta t = 0.05$ and the computations were performed until $t = 1.0$ as in this paper's other experiments.

4. Related Work

Scientific Machine Learning Models Scientific Machine Learning aims at data-driven modeling of physical systems. A notable example is Physics Informed Neural Networks (Raissi et al., 2019; Cai et al., 2022) (PINNs) that, having access to the PDE of a system, learns a neural network over the domain $\mathcal{X} \to \mathbb{R}^d$ by enforcing small residual, i.e. the error when the solution is evaluated by the PDE or the boundary conditions, over a set of sampled points. While PINNs have the capacity to model various physical systems, they need to be trained for each new condition or parameter. After several pioneering work (Long et al., 2018; 2019; Wang et al., 2020; Belbute-Peres et al., 2020), Neural Operators (Li et al., 2021b), as FNO (Li et al., 2021a) or Graph NO (Li et al., 2020), was proposed to model the continuous operators over an infinite space and have shown the ability to generalization at multiple scales. Also, more traditional image-to-image neural networks such as the U-Net (Ronneberger et al., 2015) can be adopted to model NOs. Physics-Informed Neural Operators (PINO) (Li et al., 2021b) improve the representational power of PINNs by pre-training a NO but having similar limitations to NOs. Message passing neural PDE solver (Brandstetter et al., 2022) extends the message passing principle to solve PDEs. In signal processing, for the artificial bandwidth extension (ABE) task, the Time-Frequency Network(Dong et al., 2020) (TFNet) has been proposed, which shares a similar concept of channel attention.

Training Autoregressive Models As was discussed in Sec. 2.5, there are two representative training strategies for SciML, that is, teacher-forcing and auto-regressive training. The teacher-forcing strategy was originally developed in natural language processing (Williams & Zipser, 1989; Bengio et al., 2015) which predicts n+1-th step data (word) using the true n-th step information (word). This method is known to prevent from the error-accumulation in the predicted sequential data during the model training. In the case of Scientific ML, it was found that it is profitable to add a random noise for improving the robustness against the accumulated error at the inference time (Sanchez-Gonzalez et al., 2018; 2020; Pfaff et al., 2020; Stachenfeld et al., 2021). On the other hand, the auto-regressive strategy uses the previous prediction of the model as n-th timestep information. Because of the error accumulation problem, not so many works were adopted, e.g. (Li et al., 2021a). Recently, (Brandstetter et al., 2022) proposed a reconciling method for this problem which is the so-called "pushforward trick". To increase stability, this method uses an adversarial-style loss which predicts the next timestep data using the previous prediction which is calculated using true data. Note that our method adopted a curriculum training strategy to prevent error accumulation instead of using an additional loss function.

5. Conclusion and Limitations

The CAPE module allows any data-driven SciML models to incorporate PDE parameters. We propose a simple but ef-

[9] The experiments were run using an Nvidia GeForce RTX 3090 with CUDA-11.6. The ML models are implemented using PyTorch 1.12.1 and the numerical simulations with JAX-0.3.17.

fective curriculum training strategy that allows us to bridge teacher-forcing and auto-regressive learning. We performed an extensive set of experiments and showed the effectiveness and efficiency of our method from various aspects: generalization of seen/unseen PDE parameters during training, parameter efficiency, and inference time.

One of our key findings is the behavior of ML models without parameter embeddings which either (1) exhibit poor performance uniformly for all the PDE parameters (1D Advection eq.), or (2) overfit to a specific parameter regime (1D Burgers eq. and 2D NS eqs.). Hence, the ML model in this case does not generalization to PDE parameters unseen during training. On the other hand, the models with CAPE module generalized well for 1D Burgers' eq. and 2D NS eqs. whose parameters govern the physical systems' diffusion behavior. CAPE cannot be generalized for 1D Advection eq., and we consider it necessary to formulate CAPE to be more physics-informed which we consider for future work.

Limitations CAPE's scope is restricted to classical field equation problems such as solving hydrodynamic equations. It cannot be applied to particle simulations such as molecular dynamics simulations. Since CAPE is using convolution attention, it is limited to regular grids.

ACKNOWLEDGMENTS

We thank Marimuthu Kalimuthu for many fruitful comments on our manuscript. We acknowledge support by the Stuttgart Center for Simulation Science (SimTech).

References

Allen, K. R., Lopez-Guevara, T., Stachenfeld, K., Sanchez-Gonzalez, A., Battaglia, P., Hamrick, J., and Pfaff, T. Physical Design using Differentiable Learned Simulators. *arXiv e-prints*, art. arXiv:2202.00728, February 2022.

Anderson, D., Tannehill, J. C., and Pletcher, R. H. *Computational fluid mechanics and heat transfer*. Taylor & Francis, 2016.

Bar-Sinai, Y., Hoyer, S., Hickey, J., and Brenner, M. P. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

Belbute-Peres, F. D. A., Economon, T., and Kolter, Z. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *international conference on machine learning*, pp. 2402–2411. PMLR, 2020.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/e995f98d56967d946471af29d7bf99f1-Paper.pdf.

Bhatnagar, S., Afshar, Y., Pan, S., Duraisamy, K., and Kaushik, S. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

Brandstetter, J., Worrall, D., and Welling, M. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

Cai, S., Mao, Z., Wang, Z., Yin, M., and Karniadakis, G. E. Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica Sinica*, pp. 1–12, 2022.

Coros, S., Thomaszewski, B., Noris, G., Sueda, S., Forberg, M., Sumner, R. W., Matusik, W., and Bickel, B. Computational design of mechanical characters. *ACM Transactions on Graphics (TOG)*, 32(4):1–12, 2013.

Dong, Y., Li, Y., Li, X., Xu, S., Wang, D., Zhang, Z., and Xiong, S. A time-frequency network with channel attention and non-local modules for artificial bandwidth extension. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6954–6958. IEEE, 2020.

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

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hsieh, J.-T., Zhao, S., Eismann, S., Mirabella, L., and Ermon, S. Learning neural pde solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A. M., Hoffman, M. D., Dinculescu, M., and Eck, D. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

Khoo, Y., Lu, J., and Ying, L. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., and Solenthaler, B. Deep fluids: A generative network for parameterized fluid simulations. In *Computer graphics forum*, volume 38, pp. 59–70. Wiley Online Library, 2019.

Kingma, D. P. and Ba, J. In *ICLR (Poster)*.

Lewy, H., Friedrichs, K., and Courant, R. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische annalen*, 100:32–74, 1928.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *International Conference on Learning Representations (ICLR)*, 2021a.

Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021b.

Long, Z., Lu, Y., Ma, X., and Dong, B. Pde-net: Learning pdes from data. In *International conference on machine learning*, pp. 3208–3216. PMLR, 2018.

Long, Z., Lu, Y., and Dong, B. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

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

Mirza, M. and Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

Pradita, T., Leiteritz, R., Takamoto, M., MacKinlay, D., Alesiani, F., Pflüger, D., and Niepert, M. PDEBench: A diverse and comprehensive benchmark for scientific machine learning, 2022. URL https://darus.uni-stuttgart.de/privateurl.xhtml?token=1be27526-348a-40ed-9fd0-c62f588efc01.

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

Raissi, M., Perdikaris, P., and Karniadakis, G. E. 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, February 2019. doi: 10.1016/j.jcp.2018.10.045.

Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pp. 4470–4479. PMLR, 2018.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020.

Shaw, P., Uszkoreit, J., and Vaswani, A. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

Sirignano, J. and Spiliopoulos, K. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

Stachenfeld, K., Fielding, D. B., Kochkov, D., Cranmer, M., Pfaff, T., Godwin, J., Cui, C., Ho, S., Battaglia, P., and Sanchez-Gonzalez, A. Learned coarse models for efficient turbulence simulation. *arXiv preprint arXiv:2112.15275*, 2021.

Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., and Niepert, M. PDEBench: An Extensive Benchmark for Scientific Machine Learning. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks*, 2022. URL https://arxiv.org/abs/2210.07182.

Tu, Z., Talebi, H., Zhang, H., Yang, F., Milanfar, P., Bovik, A., and Li, Y. Maxvit: Multi-axis vision transformer. *ECCV*, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

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

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Learning Neural PDE Solvers with Parameter-Guided Channel Attention

A. Additional Related Work

Parameter Embedding There has been an interest to put additional information to DNN. For example, Transformer-type models take into account the information of the position of words in the sentence using positional encoding (Vaswani et al., 2017; Shaw et al., 2018; Huang et al., 2018). In the case of data generation, cGAN (Mirza & Osindero, 2014) accepts a conditional parameter to the generator network. In the case of SciML, PINN (Cai et al., 2022) and PINO (Li et al., 2021b) can explicitly take into account PDE parameters during training but cannot change them during the test time. Recently Message-passing PDE solver (Brandstetter et al., 2022) was proposed in which PDE parameters and boundary conditions can be freely embedded into the network. However, this is specialized only for these models, and cannot apply the other models as our proposed method.

B. Detailed Training Setup

B.1. General Setup

As is explained in Sec. 3, we used datasets provided by PDEBench (Pradita et al., 2022) a benchmark for SciML from which we downloaded datasets of the following PDEs: 1D Advection equation, 1D Burgers equation, and 2D compressible NS equations. For 1-dimensional PDEs, we used $N = 9000$ training instances and 1000 test instances for each PDE parameter with spatial resolution: 128 ($\Delta x = 1/128$) and temporal step-size: $\Delta t = 0.05$. For 2-dimensional NS equations, we used $N = 900$ training instances and 100 test instances for each PDE parameter with spatial resolution: 64×64 ($\Delta x = \Delta y = 1/128$) and temporal step-size: $\Delta t = 0.05$. Smaller temporal step-size results are also provided in Appendix H.

Concerning the training, the optimization was performed with Adam (Kingma & Ba) for 100 epochs. The learning rate was set as 3×10^{-3} which is divided by 2.0 every 20 epochs. The mini-batch size we used was 50 for all the cases. To stabilize the CAPE module's training in the initial phase, we empirically found it is a little better if we have a warm-up phase during which only CAPE module is updated. We performed warm-up for the first 3 epochs, which slightly reduce the final performance fluctuations resulting from the randomness of the initial weights of the network. In the CAPE module, the kernel size of the depth-wise convolution was set as: 5. The training was performed on GeForce RTX 2080 GPU for 1D PDEs and GeForce GTX 3090 for 2D NS equations. For PINO loss, we set the coefficient 1 following the original implementation.

B.2. Hyper-parameter Selection

The hyper-parameters and the BASE network parameters are listed in Tab. 5. To clarify that our result with CAPE was not overfitted to the test dataset, we also performed a hyper-parameter search of the coefficient α of the loss term for CAPE L_{cape} which is the unique hyper-parameter we can tune in the experiments. We created new data for 1D Advection equation with advection velocity $\beta = 0.3, 0.5, 1.2$ which are not included in our main experiments provided in Tab. 1. We split the data into train/validation/test with ratio: $(0.9, 0.05, 0.05)$, and saved the best model in terms of the validation loss value. The results are summarized in Tab. 6 which indicates that the best parameter exists around $\alpha = 10^{-4}$ independent of the test set, and our choice in Tab. 5 is validated.

B.3. Networks' sizes comparison

The networks' structures of the BASE models are presented in Tab. 5, while the resulting network size is listed in Tab. 7.

B.4. Conditional Modelling

Here we provide a more detailed explanation for the conditional models in Sec. 3. In this paper, the conditional models have the same model structures as the vanilla ones, but we only change the input data as:

$$u^k \in \mathbb{R}^{C \times N_1 \times \cdots} \to \text{concatenate}(u^k, \lambda) \in \mathbb{R}^{(C+1) \times N_1 \times \cdots}, \tag{18}$$

where the PDE parameter are taken as a part of the input by concatenating it to the field data's new channel dimension. Although it is possible to consider a more elaborate method, such as performing an MLP on the PDE parameters, we avoid those cases for simplicity.

Dimension	Model	width	mode	d	mode (CAPE)	α
1D	FNO	36	12	–	–	–
	FNO w.t. CAPE	20	12	64	12	5.7×10^{-5}
2D	FNO	28	12	–	–	
	FNO w.t. CAPE	20	12	64	9	8.3×10^{-5}
Dimension	**Model**	**init features**		**d**	**mode (CAPE)**	α
1D	Unet	32		–	–	–
	Unet w.t. CAPE	32		64	12	5.7×10^{-5}
2D	Unet	32		–	–	
	Unet w.t. CAPE	30		64	9	8.3×10^{-5}

Table 5. Network Parameters. Where d is the channel number of CNNs in CAPE.

PDE	α	nRMSE
1D Advection	10^{-6}	0.069
	10^{-5}	0.073
	5×10^{-5}	0.046
	10^{-4}	0.045
	10^{-3}	0.068

Table 6. Results on Train/Validation/Test data set with 1D Advection equation. The model is FNO with CAPE.

B.5. Modification on the Message-Passing PDE Solvers

In Sec. 3, we consider the Message-Passing PDE Solvers (Brandstetter et al., 2022) as a baseline model that accepts PDE parameters. For a fair comparison, we are forced to modify the model as (1) accepting only 1-time step data, (2) adding a case of "time-window" parameter with 10 for the decoder. Concerning the first case, the original model assumes to accept sequential data whose time-step size must be equal to the size of the "time-window" parameter. For the second modification, we added a new 1D-convolution layer accepting the "time-window" parameter equal to 10. The detailed structure of the new decoder is provided in Tab. 8.

B.6. Modification on the TF-Net

In Sec. 3, we employed the TF-Net (Wang et al., 2020) as a baseline model for 2D compressible Navier-Stokes equations. However, the original model does not allow us to flexibly change the temporal filter size (the number of temporal time-step). Consequently, we made modifications to the model to suit our experimental setup, where only previous 2 time-step data was considered. In our experiment, we replaced the original temporal filter into a simple 1×1-Convolution layer whose kernel size is unity and the channel number is two for both input and output channels. We consider that the somewhat unsatisfactory results obtained with our TF-Net can be partly attributed to this modification.

C. An additional experiments on 2D Burgers equation

In this section, we provide an additional experiment results conducted on 2-dimensional Burgers equation whose expression is given as:

$$\partial_t u(t,x,y) + u(t,x,y)(\partial_x u(t,x,y) + \partial_y u(t,x,y)) = \nu/\pi(\partial_{xx} u(t,x,y) + \partial_{yy} u(t,x,y)), \tag{19}$$

where ν is the diffusion coefficient and the parameter of this equation. Note that we keep the variable as a scalar function and also the diffusion coefficient as a constant for both of the spatial directions, for simplicity. The considered diffusion coefficient are listed in Tab. 10. The results are provided in Tab. 9 which shows our CAPE module provides better results than the conditional model[10].

[10]The experimental setup adheres to the configuration employed in the 2D CFD scenario.

Dimension	Model	# Parameters
1D	FNO	73K
	FNO w.t. CAPE	68K
	Unet	2.71M
	Unet w.t. CAPE	2.75M
	MPNN	614k
2D	FNO	0.91M
	FNO w.t. CAPE	0.82M
	Unet	7.8M
	Unet w.t. CAPE	7.2M
	TF-net	7.4M

Table 7. Model Size

Module	in-channel	out-channel	kernel size	stride
1D Conv-1	1	8	18	5
1D Conv-2	8	1	14	1

Table 8. Decoder CNN structure for Message-Passing PDE Solvers

D. Discussion of Results for the PINO Loss

Reason why PINO loss does not work The PINO loss function is an emulation of the PINO loss function using ML's output. For example, the PINO loss function of the 1D Advection equation case is:

$$L_{\text{PINO}} = \frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} - \beta \mathcal{F}^{-1}(ik\mathcal{F}(u)). \tag{20}$$

On the other hand, the usual spectral method solves the equation as:

$$\tilde{u}_j^{n+1} = u_j^{n-1} + 2\Delta t \beta \mathcal{F}^{-1}(ik\mathcal{F}(u)). \tag{21}$$

By substituting Eq. 21, Eq. 20 reduces to:

$$L_{\text{PINO}} = \frac{u_j^{n+1} - \tilde{u}_j^{n+1}}{2\Delta t}. \tag{22}$$

This shows that the PINO loss function penalizes the machine learning model prediction to be close to the spectral method prediction. However, in general, the classical direct simulation methods have to use the time-step size Δt restricted by the theoretical stability condition, such as the CFL condition. And the prediction $\tilde{u}_j^{n+1}$ becomes completely wrong if the used Δt does not satisfy the stability condition, resulting in the PINO loss function leading to a completely harmful effect for the ML models. In our experiments, $\Delta t = 0.05$ is larger than the time step demanded by the stability condition, e.g., $\Delta t < \Delta x/\beta \sim 0.0025$ if we set $\Delta x = 1/128, \beta = 1$. So, we consider that our experiment result showing worse error from PINO loss function is a natural result from this consideration.

E. Detailed Results

E.1. PDE Parameter Dependence Study

In Tab. 11, Tab. 12, and Tab. 13, we provide the nRMSE values plotted in Fig. 6.

E.2. Curriculum Strategy Study

In Tab. 14 we provide a full ablation study result of the curriculum strategy.

PDE	model	BASE	Conditional	prev. 2-steps	CAPE
2D Burgers	FNO	$0.0357^{\pm 0.8\times 10^{-3}}$	$0.0274^{\pm 0.4\times 10^{-3}}$	$\mathbf{0.0249}^{\pm 1.5\times 10^{-3}}$	$0.0267^{\pm 2.6\times 10^{-3}}$
	Unet	$0.31^{\pm 0.04}$	$0.32^{\pm 0.10}$	$\mathbf{0.21}^{\pm 0.07}$	$0.28^{\pm 0.06}$
	TFNet	N/A	N/A	$0.0427^{\pm 0.023}$	N/A

Table 9. List of the normalized RMSE (the smaller, the better) for 2D Burgers eq.

PDE	training & test parameters
2D Burgers	$\nu = (0.001, 0.004, 0.01, 0.04, 0.1, 0.4, 1.0)$

Table 10. PDE parameters used in the experiments on 2D Burgers equation.

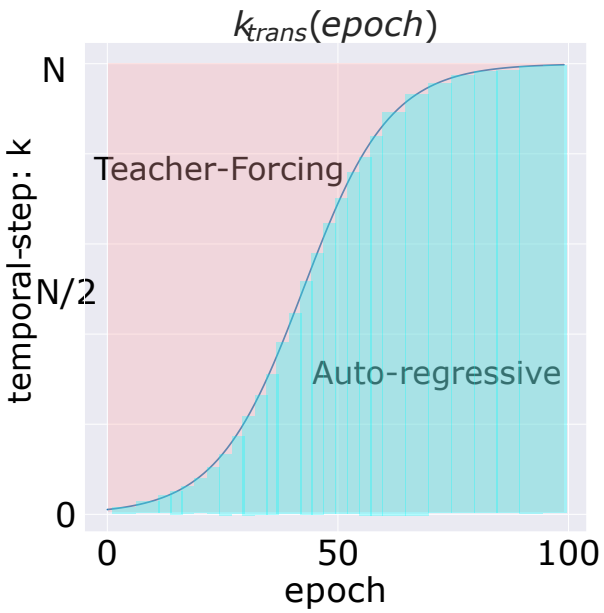

Figure 7. A plot of an instance of the function defined in Eq. 12 where we set: $\Delta = 0.25$.

F. Detailed Description of the Curriculum Strategy

Fig. 7 plots the profile of Eq. 12 in terms of the epoch number where the maximum epoch number is assumed 100. We also provided the detailed algorithm of our curriculum strategy in terms of epochs and temporal steps in Algorithm 1. In our all the calculation with curriculum strategy, we set: $\Delta = 0.2$.

Concerning the form of k_{trans} in Eq. 12, it has many variations, such as a simple linear growth. We empirically decided the expression of it as given in Eq. 12 which allows to rather gradual start and end of the transition process thanks to the form of hyperbolic tangent function.

G. Study for CAPE module structure

G.1. Ablation Study

In this section, we provided results of ablation study for our CAPE module's internal structure to provide an insight of the inductive bias of CAPE. In this study, we performed training without (1) spectral-convolution, (2) 1×1-convolution, and (3) depthwise-convolution. The results were provided in Tab. 15 that indicates that all the 3-convolution layers and LayerNormalization play important roles on the error, but the spectral-convolution has the strongest impact. However, it also shows that the important factor depends on PDEs because of the difference of PDE natures (e.g., advection, diffusion, or

PDE	β type	0.1	0.2	0.4	0.7	1.0	2.0	4.0	7.0
Advection	BASE	0.716	0.700	0.638	0.680	0.714	0.721	0.700	0.729
	CAPE	0.846	0.056	0.040	0.035	1.218	0.046	0.049	3.300

Table 11. The nRMSEs FNO with CAPE in terms of each advection velocity of 1D Advection equation. Visualization is given in Fig. 6

PDE	ν type	0.001	0.002	0.004	0.007	0.01	0.02	0.04	0.07	0.1
Burgers	BASE	0.223	0.216	0.218	0.201	0.198	0.173	0.138	0.134	0.124
	CAPE	0.185	0.179	0.167	0.155	0.155	0.138	0.127	0.106	0.107

Table 12. The nRMSEs FNO with CAPE in terms of each diffusion coefficient of 1D Burgers equation. Visualization is given in Fig. 6

non-linear system equations, and so on). It also indicates that our selection always shows a better result, though not always the best.

G.2. Study on Other Possibility of CAPE Structure

Inspired by recent work (Tu et al., 2022), we also tried a sequential manner of the convolution layers in the CAPE module. as described in Eq. 10.

$$\boldsymbol{y}_\alpha^k(x) = h_{1\times1,d\to c\times\ell}\left(\sigma\left(h_{1\times1,c\to d}(\boldsymbol{u}^k) + \boldsymbol{v}_\alpha^k\right)\right)(x), \tag{23}$$

$$\boldsymbol{y}^k(x) = \boldsymbol{y}_{\alpha_1}^k(\boldsymbol{y}_{\alpha_2}^k(\boldsymbol{y}_{\alpha_3}^k(x))), \tag{24}$$

where x is the output of the previous layer or input, and α_k is either spectral convolution, 1×1 convolution, or depthwise convolution. The result on 2D NS equations is provided in Tab. 16. It shows that in this case the order (depthwise conv., 1×1 conv., spectral conv.) is the best choice. However, it also shows that the best error is still larger than the vanilla CAPE module if the model weight parameter number is the same.

H. Larger Time-step Experiments

In our experiments, the time-step size of the data might be somewhat small (1D: 40, 2D: 20). This small number of time step, however, might be attributed to the effectiveness of CAPE and the curriculum strategy because the small number of time steps may reduce the accumulation error in time due to the autoregressive nature of the model's prediction, which can result in training instability and a lack of generalizability. To clarify the robustness of CAPE and the curriculum strategy in the light of the time-step number, we performed additional experiments with 100-time steps ($\Delta t = 0.02$). The results are provided in Tab. 17 which evinced that both FNO and FNO w.t. CAPE were trained effectively, without any instability or severe error accumulation, though the errors themselves are moderately increased relative to the case with 40 steps.

I. Visualization of Attention Weights

Fig. 8 and Fig. 9 are the plots of the Convolution kernel after being multiplied by the channel attention. For simplicity, we only consider the one-dimensional cases and the depthwise convolution kernel that is more interpretable than the other convolution kernels. Fig. 8 is the case of 1D Advection equation. It shows that the kernel is very sparse and the channel attention chooses different kernel as the advection velocity increases, resulting in adjusting appropriate information transfer corresponding with the input advection velocity. Fig. 9 is the case of 1D Burgers equation. It shows that the kernel becomes nearly zero by channel attention when the diffusion coefficient is small, indicating a small diffusion. On the other hand, the number of activated kernels increases with the diffusion coefficient, indicating that the operation becomes closer to the averaging, corresponding with the function of the diffusion.

PDE	ν type	0.2	0.4	0.7	1.0	2.0	4.0
Burgers	BASE	0.168	0.335	0.458	0.674	1.626	2.460
	CAPE	0.081	0.094	0.113	0.104	0.079	0.253

PDE	$\eta = \zeta$ type	10^{-8}	0.001	0.004	0.007	0.01	0.04	0.07	0.1
2D NS ($M = 0.1$)	BASE	0.508	0.500	0.488	0.491	0.529	1.447	3.132	5.228
	CAPE	0.516	0.487	0.482	0.462	0.486	0.965	1.692	2.582
2D NS ($M = 1.0$)	BASE	0.579	0.545	0.495	0.471	0.453	0.635	1.141	1.962
	CAPE	0.569	0.544	0.501	0.485	0.474	0.494	0.585	0.779

Table 13. The nRMSEs FNO with CAPE in terms of each viscosity of 2D NS equations. Visualization is given in Fig. 6

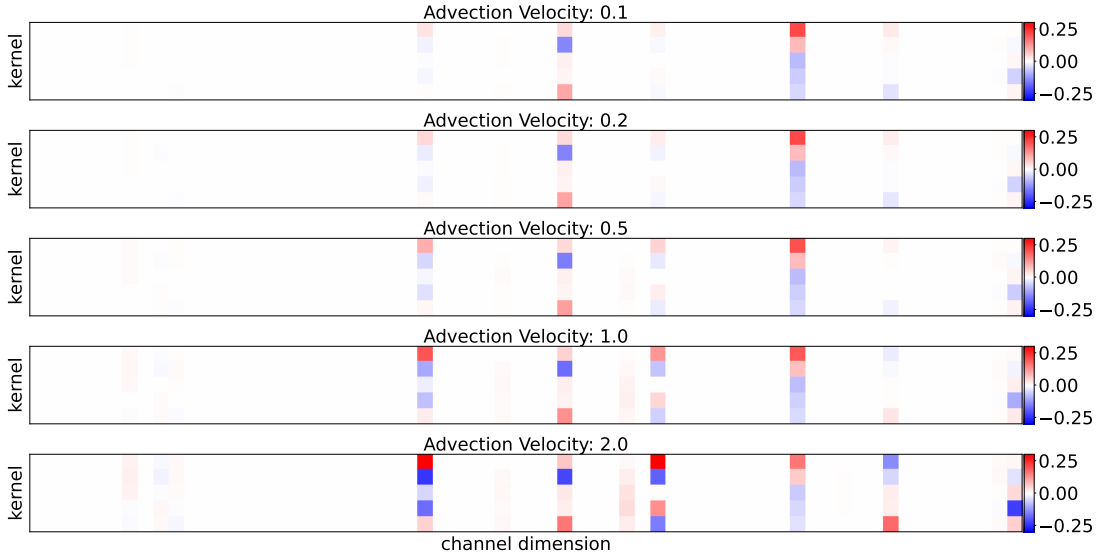

Figure 8. Plot of attention weights in the case of 1D Advection equation.

PDE	Model	Ablation	nMSE	
Advection	FNO	curriculum strategy	$\mathbf{0.04}$	
		pure Autoregressive	0.11	$(+\mathbf{0.07})$
		pure Teacher-Forcing	$\mathbf{0.04}$	(± 0.00)
Advection	Unet	curriculum strategy	$\mathbf{0.11}$	
		pure Autoregressive	0.17	$(+\mathbf{0.06})$
		pure Teacher-Forcing	0.12	$(+0.01)$
Burgers	FNO	curriculum strategy	$\mathbf{0.13}$	
		pure Autoregressive	0.16	$(+\mathbf{0.03})$
		pure Teacher-Forcing	$\mathbf{0.13}$	$(+0.00)$
Burgers	Unet	curriculum strategy	$\mathbf{0.45}$	
		pure Autoregressive	0.94	$(+\mathbf{0.49})$
		pure Teacher-Forcing	0.84	$(+0.39)$
2D NS	FNO	curriculum strategy	$\mathbf{8.0 \times 10^{-1}}$	
		pure Autoregressive	$1.3 \times 10^{+0}$	$(+0.5)$
		pure Teacher-Forcing	$3.2 \times 10^{+0}$	$(+\mathbf{2.4})$
2D NS	Unet	curriculum strategy	$\mathbf{7.0 \times 10^{-1}}$	
		pure Autoregressive	$1.0 \times 10^{+0}$	$(+\mathbf{0.3})$
		pure Teacher-Forcing	$1.0 \times 10^{+0}$	$(+\mathbf{0.3})$

Table 14. Ablation study for the Advection, Burgers and 2D CFD equations with FNO as BASE model.

Algorithm 1 Algorithm of the curriculum training strategy

Input model parameters $\boldsymbol{\theta}$, training epoch number n, total training epoch number M, Training samples: $\{\boldsymbol{u}^i\}_{i=0,\cdots,N}$, temporal index k, final time step of the training sample N, ϵ is the random noise.

for n $= 0$ **to** M **do**
  Calculate k_{trans} following Eq. 12,
  for k $= 0$ **to** $N-1$ **do**
    if $k \leq k_{\text{trans}}$ **then**
      $\tilde{\boldsymbol{u}}^{k+1} = \text{NN}(\tilde{\boldsymbol{u}}^k + \epsilon; \boldsymbol{\theta})$
    else
      $\tilde{\boldsymbol{u}}^{k+1} = \text{NN}(\boldsymbol{u}^k + \epsilon; \boldsymbol{\theta})$
    end if
    $\mathbf{L}^k \leftarrow \text{MSE}(\tilde{\boldsymbol{u}}^{k+1}, \boldsymbol{u}^{k+1})$
  end for
  $\boldsymbol{\theta} \leftarrow$ Optimizer $(\sum_{k=1}^{N} \mathbf{L}^k)$
end for

PDE	model	Ablation	nRMSE
1D Advection	FNO	BASE	0.04
		w/t Spectral Convolution	0.06 **(+0.02)**
		w/t 1×1 Convolution	0.05 (+0.01)
		w/t Depthwise Convolution	0.05 (+0.01)
		w/t LN	0.03 (-0.01)
1D Burgers	FNO	BASE	0.13
		w/t Spectral Convolution	0.13 ($\pm$0.00)
		w/t 1×1 Convolution	0.12 (-0.01)
		w/t Depthwise Convolution	0.13 ($\pm$0.00)
		w/t LN	0.09 (-0.04)
2D NS	FNO	BASE	0.80
		w/t Spectral Convolution	0.91 (+0.11)
		w/t 1×1 Convolution	0.80 ($\pm$0.00)
		w/t Depthwise Convolution	0.76 (- 0.04)
		w/t LN	1.24 **(+0.45)**

Table 15. Ablation study of CAPE internal Structure.

PDE	model	# model parameters	module order	nRMSE
2D NS	FNO	1.16M	(1×1,D,S)	0.78
			(1×1,S,D)	0.77
			(D,1×1,S)	**0.64**
			(D,S,1×1)	0.71
			(S,1×1,D)	0.73
			(S,D,1×1)	0.82
2D NS	FNO	0.83M	(D,1×1,s)	0.79

Table 16. Sequential type CAPE internal Structure. (S, 1×1, D) mean (spectral conv., 1×1 conv., depthwise conv.).

PDE	model	nRMSE
1D Advection	FNO	0.90
	FNO with CAPE	0.18

Table 17. nRMSE of the case with larger time-steps on 1D Advection equation using FNO. In the experiments, we trained the FNO/FNO with CAPE with a larger time step number: 100 steps.

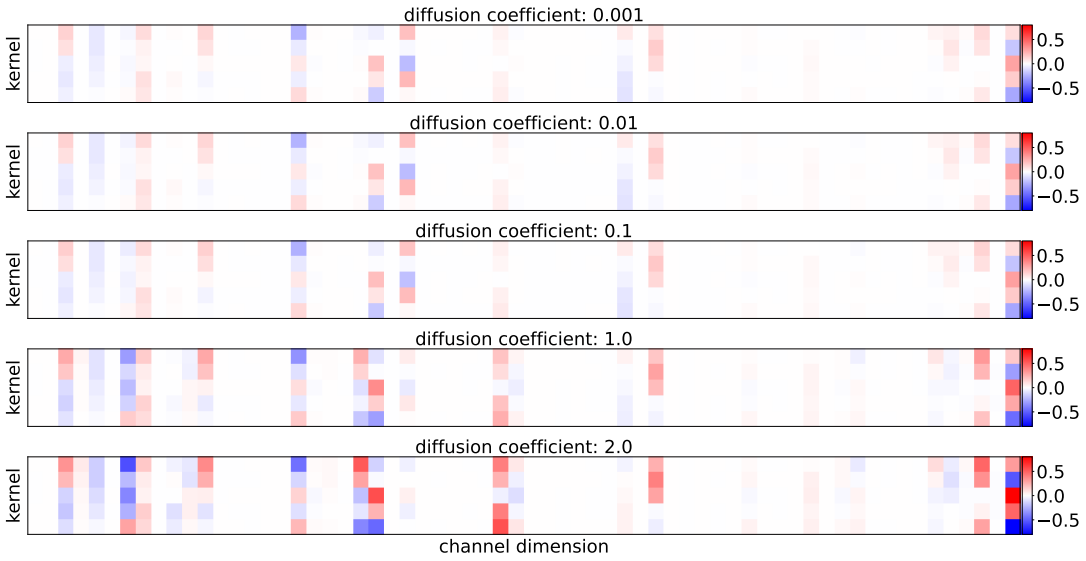

Figure 9. Plot of attention weights in the case of 1D Burgers equation.