Vectorized Conditional Neural Fields: A Framework for Solving Time-dependent Parametric Partial Differential Equations

Jan Hagnberger [1] **Marimuthu Kalimuthu** [1 2 3] **Daniel Musekamp** [1 3] **Mathias Niepert** [1 2 3]

Abstract

Transformer models are increasingly used for solving Partial Differential Equations (PDEs). Several adaptations have been proposed, all of which suffer from the typical problems of Transformers, such as quadratic memory and time complexity. Furthermore, all prevalent architectures for PDE solving lack at least one of several desirable properties of an ideal surrogate model, such as (i) generalization to PDE parameters not seen during training, (ii) spatial and temporal zero-shot super-resolution, (iii) continuous temporal extrapolation, (iv) support for 1D, 2D, and 3D PDEs, and (v) efficient inference for longer temporal rollouts. To address these limitations, we propose *Vectorized Conditional Neural Fields* (VCNeFs), which represent the solution of time-dependent PDEs as neural fields. Contrary to prior methods, however, VCNeFs compute, for a set of multiple spatio-temporal query points, their solutions in parallel and model their dependencies through attention mechanisms. Moreover, VCNeF can condition the neural field on both the initial conditions and the parameters of the PDEs. An extensive set of experiments demonstrates that VCNeFs are competitive with and often outperform existing ML-based surrogate models.

1. Introduction

The simulation of physical systems often involves solving Partial Differential Equations (PDEs), and machine learning-based surrogate models are increasingly used to address this challenging task (Lu et al., 2019; Li et al., 2020b; Cao, 2021). Utilizing ML for solving PDEs has several advantages, such as faster simulation time than classical numerical PDE solvers, differentiability of the surrogate models (Takamoto et al., 2022), and their ability to be used even when the underlying PDEs are not known exactly (Li et al., 2020a). However, if knowledge about the PDEs is available, it can be added to the model as in Physics-Informed Neural Networks (PINNs; Raissi et al. (2017)).

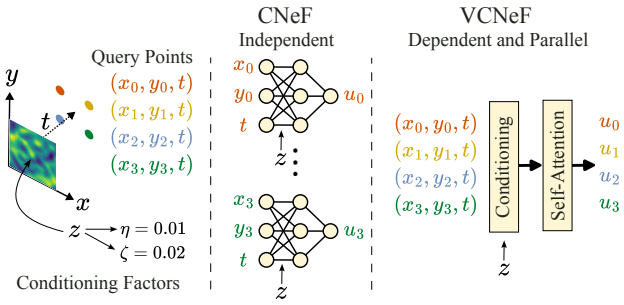


Figure 1: Conditional Neural Field (CNeF) vs proposed Vectorized CNeF (VCNeF) for solving parameterized PDEs.

Transformers (Vaswani et al., 2017) and its numerous variants are successfully used in natural language processing (Devlin et al., 2018), speech processing (Gulati et al., 2020), and computer vision (Dosovitskiy et al., 2020). Due to their remarkable ability to effectively model long-range dependencies in sequential data and to have favorable scaling behavior, Transformers are used in an increasing number of additional applications. Transformer models have been gaining traction in Scientific Machine Learning (SciML) to model physical systems (Geneva & Zabaras, 2020), solve PDEs (Cao, 2021; Li et al., 2023a;b; Hao et al., 2023), and pretrain multiphysics SciML foundation models (McCabe et al., 2023). Meanwhile, recent advances in neural networks for computer graphics tasks have introduced Neural Fields (Xie et al., 2021), which have proven to be an efficient method to solve PDEs (Sitzmann et al., 2020; Chen et al., 2023b;a; Yin et al., 2023; Serrano et al., 2023).

Despite these recent advances in neural architectures for

[1]Machine Learning and Simulation Lab, Institute for Artificial Intelligence, University of Stuttgart, Stuttgart, Germany [2]Stuttgart Center for Simulation Science (SimTech) [3]International Max Planck Research School for Intelligent Systems (IMPRS-IS). Correspondence to: Jan Hagnberger <j.hagnberger@gmail.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

The source code of VCNeF is available at https://github.com/jhagnberger/vcnef/

PDE solving, current methods lack several of the characteristics of an ideal PDE solver: (i) generalization to different Initial Conditions (ICs), (ii) PDE parameters, (iii) support for 1D, 2D, and 3D PDEs, (iv) stability over long rollouts, (v) temporal extrapolation, (vi) spatial and temporal super-resolution capabilities, all with affordable cost, high speed, and accuracy.

Towards developing a model that encompasses these ideal characteristics, we propose *Vectorized Conditional Neural Field* (VCNeF), a linear transformer-based conditional neural field that solves PDEs continuously in time, endowing the model with temporal as well as spatial Zero-Shot Super-Resolution (ZSSR) capabilities. The model introduces a new mechanism to condition a neural field on Initial Conditions (ICs) and PDE parameters to achieve generalization to both ICs and PDE parameter values not seen during training. While modeling the solution using neural fields such as PINNs naturally provide temporal and spatial ZSSR, these methods are inefficient since we need to query them separately for every temporal and spatial location in the domain. We achieve faster training and inference by vectorizing these computations on GPUs. Moreover, the proposed method explicitly models dependencies between multiple simultaneous spatio-temporal queries to the model.

Concretely, we focus on training and evaluating VCNeF on 1D, 2D, and 3D Initial Value Problems (IVPs) where an IC is given and one predicts multiple future timesteps, as this setting is best suited for real-world applications. The IC could be the data from measurements, and longer rollouts are required to simulate the system under consideration. Additionally, we train our model on multiple PDE parameter values to evaluate its capability to generalize to unseen PDE parameter values.

In summary, we make the following contributions:

- A time-continuous transformer-based architecture that represents the solutions to PDEs at any point in time as neural fields, even those not encountered during training, which is accomplished by explicitly incorporating the query time.
- We empirically verify that VCNeFs generalize robustly to PDE parameter values not seen during training through effective parameter conditioning while also possessing intrinsic capabilities for spatial and temporal zero-shot superresolution.
- A model that naturally provides an implicit vectorization of the spatial coordinates that allows for faster training and inference. It also allows computing the solution of multiple spatial points in one forward pass and exploits spatial dependencies instead of processing them independently.

2. Problem Definition

In this section, we formally introduce the problem of solving parametric PDEs using neural surrogate models.

Partial Differential Equations. Following Brandstetter et al. (2022), PDEs over the time dimension, denoted as $t \in [0, T]$, and over multiple spatial dimensions, indicated by $\boldsymbol{x} = (x_x, x_y, x_z, \ldots)^\top \in \mathbb{X} \subseteq \mathbb{R}^D$ with D dimensions of a PDE, can be expressed as

$$\begin{gathered} \partial_t u = F(t, \boldsymbol{x}, u, \partial_{\boldsymbol{x}} u, \partial_{\boldsymbol{xx}} u, \ldots) \text{ with } (t, \boldsymbol{x}) \in [0, T] \times \mathbb{X} \\ u(0, \boldsymbol{x}) = u(0, \cdot) = u^0(\boldsymbol{x}) = u^0 \text{ with } \boldsymbol{x} \in \mathbb{X} \\ B[u](t, \boldsymbol{x}) = 0 \text{ with } (t, \boldsymbol{x}) \in [0, T] \times \partial\mathbb{X} \end{gathered} \tag{1}$$

where $u : [0, T] \times \mathbb{X} \to \mathbb{R}^c$ represents the solution function of the PDE that satisfies IC $u(0, \boldsymbol{x})$ for time $t = 0$ and the Boundary Conditions (BCs) $B[u](t, \boldsymbol{x})$ if $\boldsymbol{x}$ is on the boundary $\partial\mathbb{X}$ of the domain $\mathbb{X}$. c denotes the number of output channels or field variables of the PDE. Solving a PDE means determining (an approximation of) the function u that satisfies Equation (1). PDEs often contain a parameter, such as the diffusion or viscosity coefficient, which influences their dynamics. We denote the vector of PDE parameter(s) as $\boldsymbol{p}$[1]. The notation $\partial_{\boldsymbol{x}} u, \partial_{\boldsymbol{xx}} u, \ldots$ represents the i^{th} order (where $i \in [1, 2, .., n]$) partial derivative $\frac{\partial u}{\partial \boldsymbol{x}}, \frac{\partial^2 u}{\partial \boldsymbol{x}^2}, \ldots, \frac{\partial^n u}{\partial \boldsymbol{x}^n}$.

Train and Test Data. One has to use discretized data generated by a numerical solver to train surrogate models. The temporal domain $[0, T]$ is discretized into N_t timesteps yielding a sequence $(u(t_0, \cdot), u(t_1, \cdot), \ldots, u(t_{N_t - 1}, \cdot))$ which describes the evolution of a PDE. $\Delta t = t_{i+1} - t_i$ denotes the temporal step size or resolution. We denote the number of timesteps used for the IC as N_i. Since we focus on initial value problems with one timestep as the IC, it holds $N_i = 1$. The spatial domain $\mathbb{X}$ is also transformed into a grid $\boldsymbol{X}$ by discretizing each spatial dimension. Each grid element localizes a point in the spatial domain of the PDE. For 1D PDEs, the grid $\boldsymbol{X} = (\left(\boldsymbol{x_i} = (x_{x_i})\right)_{i=1}^{s_x})^\top \in \mathbb{R}^{s_x}$ and s_x denotes the spatial resolution (i.e., number of spatial points) of the x-axis. Similarly, for 2D PDEs the grid $\boldsymbol{X} = (\left(\boldsymbol{x_i} = (x_{x_i}, x_{y_i})\right)_{i=1}^{s_x \cdot s_y})^\top \in \mathbb{R}^{(s_x \cdot s_y) \times 2}$ and s_x, s_y denote the spatial resolutions of the x and y axis, respectively. $u(t_i, \boldsymbol{X}) = (u(t_i, \boldsymbol{x_1}), u(t_i, \boldsymbol{x_2}), \ldots, u(t_i, \boldsymbol{x_s}))^\top \in \mathbb{R}^{s \times c}$ with $s = s_x \cdot s_y \cdot s_z \cdot \ldots$ contains the solutions at different spatial locations on the grid $\boldsymbol{X}$. The PDE parameters are stacked as a vector $\boldsymbol{p} = (p_1, \ldots, p_j)^\top \in \mathbb{R}^j$ where p_i represents the value of a PDE parameter.

A dataset $\mathcal{D} = \{(\boldsymbol{I_1}, \boldsymbol{Y_1}), \ldots, (\boldsymbol{I_N}, \boldsymbol{Y_N})\}$ for each PDE consists of N samples. $\boldsymbol{I_j} = (u(t_0, \boldsymbol{X}), \ldots, u(t_{N_i}, \boldsymbol{X}))$

[1] Scalars are represented with a small letter (e.g., a), vectors with small boldfaced letter (e.g., $\boldsymbol{a}$), and matrices and N-way tensors (s.t. N $\geq$ 3) with a capital boldfaced letter (e.g., $\boldsymbol{A}$).

denotes the solutions given as IC and $\boldsymbol{Y_j} = ((u(t_0, \boldsymbol{X}), \ldots, u(t_{N_t}, \boldsymbol{X}))$ denotes the target sequence of timesteps which represents the trajectory of the PDE.

Training Objective. The training objective aims to optimize the parameters θ (i.e., weights and biases) of the model f_θ that best approximate the true function u by minimizing the empirical risk over the dataset $\mathcal{D}$

$$\underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{j=1}^{N_t} \mathcal{L}\left(f_\theta\left(t_j, \boldsymbol{X} \mid \boldsymbol{I_i}\right), \boldsymbol{Y}_{\boldsymbol{i},j}\right), \tag{2}$$

where $\mathcal{L}$ denotes a suitable loss function such as the Mean Squared Error (MSE). $f_\theta(t_j, \boldsymbol{X}|\boldsymbol{I_i})$ represents the prediction of the neural network for timestep t_j and a grid $\boldsymbol{X}$ given the initial condition $\boldsymbol{I_i}$.

3. Background and Preliminaries

We briefly recall (conditional) neural fields and relate them to solving parametric PDEs.

Neural Fields. In physics, a *field* is a quantity that is defined for all spatial and temporal coordinates. Neural Fields (NeFs; Xie et al. (2021)) learn a function f which maps the spatial and temporal coordinates (i.e., $\boldsymbol{x} \in \mathbb{R}^D, t \in \mathbb{R}_+$ respectively) to a quantity $\boldsymbol{q} \in \mathbb{R}^c$. Mathematically, a neural field can be expressed as a function

$$f_\theta : (\mathbb{R}_+ \times \mathbb{R}^D) \to \mathbb{R}^c \text{ with } (t, \boldsymbol{x}) \mapsto \boldsymbol{q} \tag{3}$$

that is parametrized by a neural network with parameters θ. For solving PDEs, the function f_θ models the solution function u, and the quantity $\boldsymbol{q}$ represents the solution's value for the different channels, each representing a physical quantity (e.g., density, velocity, etc.). This architectural design takes inspiration from the Eulerian specification of the flow field from classical field theory, where a field of interest is prescribed both by spatial and temporal coordinates. PINNs (Raissi et al., 2017) are a special case of neural fields with a physics-aware loss function, modeling the solution u as

$$f_\theta : (\mathbb{R}_+ \times \mathbb{R}^D) \to \mathbb{R}^c \text{ with } (t, \boldsymbol{x}) \mapsto u(t, \boldsymbol{x}) \tag{4}$$

where f_θ denotes the neural field that maps the input spatial and temporal locations to the solution of the PDE.

Conditional Neural Fields. Conditional Neural Fields (CNeFs; Xie et al. (2021)) extend NeFs with a conditioning factor $\boldsymbol{z}$ to influence the output of the neural field. The conditioning factor was originally introduced for computer vision to control the colors or shapes of objects that are being modeled. In contrast, we condition the neural field, which models the solution of the PDE, on the initial value or IC and the PDE parameters (cf. Figure 1). Thus, the conditioning factor influences the entire field.

4. Method

In this section, we propose *Vectorized Conditional Neural Fields* by explaining the transition from (conditional) neural fields to vectorized (conditional) neural fields. We also introduce our transformer-based architecture.

4.1. *Vectorized* Conditional Neural Fields

Typically, a (conditional) neural field generates the output quantities for all input spatial and temporal coordinates in multiple and independent forward passes. The training and inference times can be improved by processing multiple inputs in parallel on the GPU, which is possible since all forward passes are independent. However, there are spatial dependencies between different input spatial coordinates, particularly for solving PDEs, that will not be exploited with CNeFs or by processing multiple inputs of CNeFs in parallel. Consequently, we propose extending CNeFs to

- take a vector with *arbitrary* spatial coordinates of *variable size* (a set of query points) as input,
- exploit the dependencies of the input coordinates when generating the outputs,
- generate all outputs for the inputs in one forward pass.

Hence, we name our proposed model *Vectorized Conditional Neural Field* since it implicitly generates a vectorization of the input spatial coordinates for a given time t. The VCNeF model represents a function

$$\begin{aligned} f_\theta &: (\mathbb{R}_+ \times \mathbb{R}^{s \times D}) \to \mathbb{R}^{s \times c} \\ \text{with } (t, \boldsymbol{X}) &\mapsto u(t, \boldsymbol{X}) = \begin{pmatrix} u(t, \boldsymbol{x_1}) \\ \vdots \\ u(t, \boldsymbol{x_s}) \end{pmatrix} \end{aligned} \tag{5}$$

where $u(t, \boldsymbol{x_i})$ denotes the PDE solution for the spatial coordinates $\boldsymbol{x_i}$. Note that we do not impose a structure on the spatial coordinates $\boldsymbol{x_i}$ and that the number of spatial points (i.e., s) can be arbitrary. The model can process multiple timesteps t in parallel on the GPU to further improve the training and inference time since VCNeF does not exploit dependencies between the temporal coordinates.

4.2. VCNeF for Solving PDEs

VCNeFs directly learn the solution function u of a PDE by mapping a timestep t_n to a subsequent timestep t_{n+1}. Hence, VCNeFs are not autoregressive by design. The model is conditioned on the IC to allow for generalization to different ICs and on the PDE parameters $\boldsymbol{p}$ to generalize to PDE parameter values not seen during training. The VCNeF

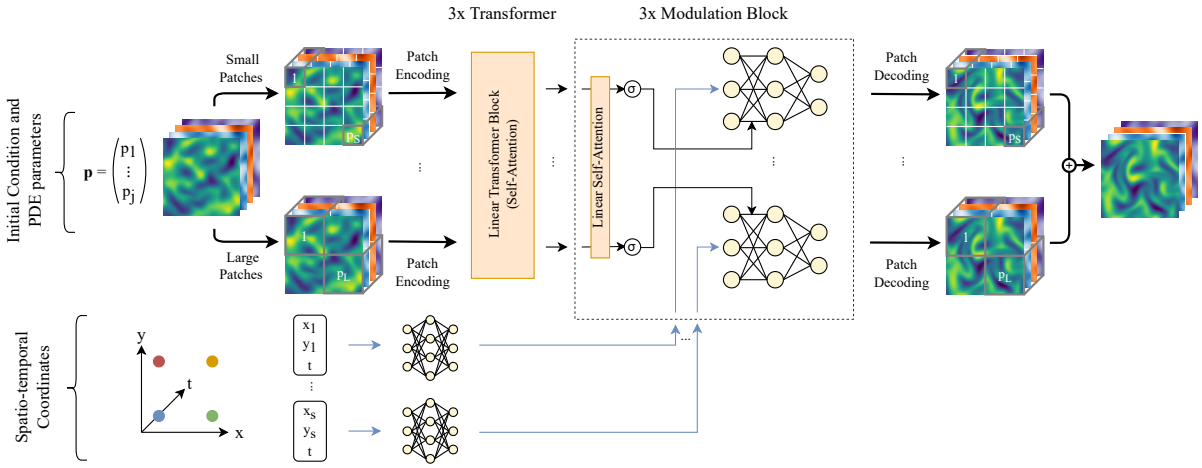


Figure 2: An illustration of the VCNeF architecture for solving parametric time-dependent 2D PDEs. Latent representations of ICs are generated with a multi-scale patching mechanism (Chen et al., 2021). A modulation block consists of self-attention, activation function σ, and a modulated neural field that uses the scaling of FiLM (Perez et al., 2018) to condition the spatio-temporal coordinates on ICs.

model can be expressed as a function

$$f_\theta : (\mathbb{R}_+ \times \mathbb{R}^{s\times D} \times \mathbb{R}^{s\times c} \times \mathbb{R}^j) \to \mathbb{R}^{s\times c}$$
$$\text{with } (t, \boldsymbol{X}, u(0,\boldsymbol{X}), \boldsymbol{p}) \mapsto u(t, \boldsymbol{X}|u(0,\boldsymbol{X}), \boldsymbol{p}), \tag{6}$$

where θ represents the parameters of the neural network, $\boldsymbol{X} \in \mathbb{R}^s$ the grid with query spatial coordinates, t the query time, $u(0, \boldsymbol{X})$ the IC, and $\boldsymbol{p}$ the vector of PDE parameters. $u(t, \boldsymbol{X}|u(0, \boldsymbol{X}), \boldsymbol{p})$ denotes the solution function, which depends on the given IC and PDE parameters, that is directly regressed by VCNeF. The shape of the grid $\boldsymbol{X}$ depends on the dimensionality of the PDE.

As a consequence, VCNeFs do not have to generate the PDE trajectory autoregressively. If the complete trajectory is needed, VCNeF can be queried with the desired times t along the trajectory. Furthermore, the model is continuous in time and can do temporal ZSSR (i.e., changing the temporal discretization Δt after training) as well as spatial ZSSR (i.e., changing spatial resolution or grid at inference time). Thus, the model can be queried with arbitrary $t \in (0, T]$ and a finer grid $\boldsymbol{\mathcal{X}}$ which is different from the grid $\boldsymbol{X}$ seen during training.

$$f_\theta(t, \boldsymbol{\mathcal{X}}, u(0,\boldsymbol{\mathcal{X}}), \boldsymbol{p}) \approx u(t, \boldsymbol{\mathcal{X}}|u(0,\boldsymbol{\mathcal{X}}), \boldsymbol{p}) \quad \forall t \in (0, T] \tag{7}$$

Alternatively, VCNeFs can be seen as an implementation of a neural operator (Kovachki et al., 2021) that is time-continuous and maps the input function (i.e. IC) to an output function that depends on both the IC and time t. However, prior work about neural operators (Li et al., 2020b;a; Cao, 2021; Li et al., 2023a) usually does not focus on time continuity. See Appendix B.1 for more details.

4.3. Neural Architecture

We propose a transformer-based VCNeF that applies self-attention to the spatial domain to capture dependencies between the spatial coordinates. The input spatio-temporal coordinates and the physical representation of ICs are represented in a latent space. Both latent representations are fed into modulation blocks that capture spatial dependencies and condition the coordinates on the IC. The output of the modulation blocks, which represent the solution, is then decoded to obtain the representation in the physical space.

Latent Representation of Coordinates. The input coordinates, consisting of the query time $t \in \mathbb{R}_+$ that determines the time for which the model's prediction is sought and the spatial coordinates $\boldsymbol{x_i} \in \mathbb{R}^D$, are represented in a latent space. For 1D PDEs, a linear layer is used for encoding, whereas, for 2D and 3D PDEs, the absolute positional encoding (PE; Vaswani et al. (2017)) to encode time t, similar to Ovadia et al. (2023) and learnable Fourier features (LFF; Li et al. (2021)) to encode the spatial coordinates are used.

$$\begin{aligned}
\text{1D: } \boldsymbol{c_i} &= (t \parallel \boldsymbol{x_i})\boldsymbol{W} + \boldsymbol{b} \\
\text{2D: } \boldsymbol{c_i} &= (\texttt{PE}(t) \parallel \texttt{LFF}(\boldsymbol{x_i}) \parallel \ldots \parallel \texttt{LFF}(\boldsymbol{x_{i+15}}))\boldsymbol{W} + \boldsymbol{b} \\
\text{3D: } \boldsymbol{c_i} &= (\texttt{PE}(t) \parallel \texttt{LFF}(\boldsymbol{x_i}) \parallel \ldots \parallel \texttt{LFF}(\boldsymbol{x_{i+63}}))\boldsymbol{W} + \boldsymbol{b} \\
\texttt{LFF}(\boldsymbol{x}) &= \texttt{MLP}(\frac{1}{\sqrt{d}}(\cos(\boldsymbol{x}\boldsymbol{W_r}) \parallel \sin(\boldsymbol{x}\boldsymbol{W_r}))^\top) \\
\boldsymbol{C} &= (\boldsymbol{c_1} \parallel \ldots \parallel \boldsymbol{c_s})^\top
\end{aligned} \tag{8}$$

where $\parallel$ stands for the concatenation of two vectors and MLP denotes a Multi-Layer Perceptron (MLP).

Latent Representation of IC. The input IC is mapped to a latent representation by either applying a shared linear layer to each solution point $u(t, \boldsymbol{x_i})$ or by dividing the spatial domain into non-overlapping patches and applying a linear layer to the patches, akin to Vision Transformers (ViTs; Dosovitskiy et al. (2020)). We divide the spatial domain into patches for 2D and 3D PDEs to reduce the computational costs. However, unlike a traditional ViT, our patch generation has two branches: patches of a smaller size ($p_S = 4$ or 4×4) and of a larger size ($p_L = 16$ or 16×16) as proposed in Chen et al. (2021) since we aim to capture the dynamics accurately at multiple scales.

$$\begin{aligned}
\text{1D: } \boldsymbol{z_i^{(0)}} &= (u(t, \boldsymbol{x_i}) \parallel \boldsymbol{x_i} \parallel \boldsymbol{p})\boldsymbol{W} + \boldsymbol{b} \\
\text{2D: } \boldsymbol{z_i^{(0)}} &= (u(t, \boldsymbol{x_i}) \parallel \ldots \parallel u(t, \boldsymbol{x_{i+15}}) \parallel \\
&\quad \boldsymbol{x_i} \parallel \ldots \parallel \boldsymbol{x_{i+15}} \parallel \boldsymbol{p})\boldsymbol{W} + \boldsymbol{b} \\
\text{3D: } \boldsymbol{z_i^{(0)}} &= (u(t, \boldsymbol{x_i}) \parallel \ldots \parallel u(t, \boldsymbol{x_{i+63}}) \parallel \\
&\quad \boldsymbol{x_i} \parallel \ldots \parallel \boldsymbol{x_{i+63}} \parallel \boldsymbol{p})\boldsymbol{W} + \boldsymbol{b} \\
\boldsymbol{Z^{(0)}} &= (\boldsymbol{z_1^{(0)}} \parallel \ldots \parallel \boldsymbol{z_s^{(0)}})^\top
\end{aligned} \tag{9}$$

A vector in the latent space (i.e., token) either represents the solution on a spatial point (for 1D) or the solution on a patch of spatial points (for 2D and 3D). The grid contains the coordinates where the solutions are sampled in the spatial domain. This information is used when generating the latent representations of the IC to ensure that each latent representation has information about the position. The PDE parameters $\boldsymbol{p}$ are also added to the latent representation. We neglect additional positional encodings to prevent length generalization problems (Ruoss et al., 2023) that could prevent changing the spatial resolution after training.

Linear Transformer Encoder for IC. We utilize a Linear Transformer (Katharopoulos et al., 2020) with self-attention in our VCNeF architecture to generate an attention-refined latent representation of the IC $\boldsymbol{Z^{(0)}}$. The global receptive field of the Transformer allows the proposed architecture to capture global spatial dependencies in the IC, although each token contains only local spatial information. Intuitively, the Transformer outputs latent representations that incorporate the entire spatial solution and not only a single spatial point or a subset of spatial points. We assume that this is beneficial to generate a better representation of the IC to condition the input coordinates accordingly.

$$\boldsymbol{Z^{(n+1)}} = \texttt{Transformer\_Block}\left(\boldsymbol{Z^{(n)}}\right) \tag{10}$$

where $\texttt{Transformer\_Block}(\cdot)$ is a Linear Transformer block with self-attention and n denotes the n^{th} block.

Modulation of Coordinates based on IC. The modulation blocks condition the input coordinates on the input IC $\boldsymbol{Z^{(3)}}$ by modulating the latent representation $\boldsymbol{C}$ of the coordinates. The block contains self-attention, a non-linearity σ, a modulation mechanism similar to Feature-wise Linear Modulation (FiLM; Perez et al. (2018)), layer normalization, residual connections, and an MLP. However, the conditioning mechanism uses only the scaling (i.e., pointwise multiplication) of FiLM and omits the shift (i.e., pointwise addition). A modulation block is expressed as

$$\begin{aligned}
\boldsymbol{Z^{(m+1)}} &= \texttt{Modulation_Block}\left(\boldsymbol{C}, \boldsymbol{Z^{(m)}}\right) \\
&= \texttt{MLP}\left(\sigma\left(\texttt{Self_Attn}\left(\boldsymbol{Z^{(m)}}\right)\right) \circ \texttt{MLP}(\boldsymbol{C})\right) \\
\sigma(\boldsymbol{X}) &= \texttt{ELU}(\boldsymbol{X}) + 1
\end{aligned} \tag{11}$$

where $\boldsymbol{Z^{(3)}} \in \mathbb{R}^{s \times d}$ represents the IC, $\boldsymbol{C} \in \mathbb{R}^{s \times d}$ denotes the latent representation of the input coordinates, $\circ$ represents the Hadamard product, and m is the m^{th} modulation block. The residual connections and layer normalization are omitted in Equation (11) for the sake of simplicity. The modulation blocks condition the spatio-temporal coordinates on the IC and PDE parameter values, and spatial self-attention incorporates dependencies between the queried spatial coordinates.

Decoding the Solution's Latent Representation. The solution's latent representation $\boldsymbol{Z^{(6)}}$ is mapped back to the physical space by either applying an MLP for 1D or by mapping the latent representations to small and large patches and outputting the weighted sum of the small and large patches for 2D and 3D.

5. Properties of VCNeFs

The proposed VCNeF model has the following properties.

Spatial and Temporal ZSSR. VCNeF can be trained on lower spatial and temporal resolutions and used for high-resolution spatial and temporal inference since the model is space and time continuous. To do so, the model can be queried with finer coordinates (i.e., intermediate spatial and temporal coordinates). Training on low-resolution data requires less computational resources and saves computing time, while inference at high-resolution data minimizes the risk of missing crucial dynamics.

Accelerated Training and Inference. The training and inference of VCNeF are accelerated by processing multiple temporal coordinates in parallel on the GPU. If the solution of multiple timesteps (e.g., $t \in \{t_1, t_2, \ldots, t_{N_t}\}$) is to be predicted, VCNeF can calculate the solution of the timesteps in parallel due to the fact that the predictions of $u(t, \cdot)$ are independent of each other. The proposed architecture uses linear attention. Nonetheless, linear attention can

be replaced with an arbitrary attention mechanism. The runtime and memory consumption are influenced by the spatial resolution $s = s_x \cdot s_y \cdot s_z \cdot \ldots$ and the cardinality N_t of the queried timesteps. For 1D, the model has a time and space complexity of $\mathcal{O}(s_x \cdot N_t)$. For 2D, the complexity is of $\mathcal{O}\left(\left(\frac{s_x \cdot s_y}{{p_S}^2} + \frac{s_x \cdot s_y}{{p_L}^2}\right) \cdot N_t\right)$ where s_x and s_y denote the spatial resolution of the x and y-axis, respectively, and p_S, p_L denote the patch sizes. We omit encoding and decoding, which include the channels c, for the sake of simplicity.

Physics-Informed VCNeF. The loss function of VCNeF can be easily extended with a physics-informed loss as in PINNs (Raissi et al., 2017) since VCNeF directly models the solution function u and therefore, the derivatives can be computed with automatic differentiation (Maclaurin et al., 2015; Paszke et al., 2017).

Randomized Starting Points Training. We suggest conditioning the model not only on the IC but also on randomly sampled timesteps along the trajectory in the training phase as data augmentation to improve the model's performance further.

6. Related Work

Physics-Informed Neural Networks and Neural Operators. A common approach for solving PDEs is Physics-Informed Neural Networks (Raissi et al., 2017) that model the underlying solution function u. Although PINNs are space-and-time continuous within the specified domain, they are finite-dimensional and hence cannot perform temporal extrapolation. Additionally, PINNs generate the solution for all input spatial coordinates independently without further exploitation of structural dependencies (Figure 1). A PDE-specific loss function allows the model to learn the underlying solution function which satisfies the PDE equation. However, PINNs can still fail to approximate the PDE solutions because of complex loss landscapes (Krishnapriyan et al., 2021). In contrast, neural operators (Li et al., 2020b) learn a mapping between two infinite-dimensional spaces or two functions where the function represents the solution function of the PDE. Consequently, neural operators are theoretically continuous in space and time, but current implementations usually have limited support for being continuous in space and time. Furthermore, neural operators generate the solution for all spatial coordinates in a single forward pass (Lu et al., 2019) and leverage the spatial dependencies of the solution by processing the spatial coordinates in one forward pass. The Fourier Neural Operator (FNO; Li et al. (2020a)) is a prevalent instantiation of a neural operator that is based on Fourier transforms. Physics-informed neural operators (Li et al., 2023c) extend neural operators with a PDE-specific loss to further improve the accuracy of the model. VCNeFs can be seen as a combination of PINNs and neural operators since VCNeFs can be queried continuously over time like PINNs, but process the spatial coordinates in a single forward pass, exploiting the spatial dependencies between the queried coordinates as existing neural operator implementations do.

PDE	Timesteps	Spatial res.	PDE parameters
1D Burgers	41	256	$\nu = 0.001$
1D Advection	41	256	$\beta = 0.1$
1D CNS	41	256	$\eta = \zeta = 0.007$
2D CNS	21	64×64	$\eta = \zeta = 0.01$
3D CNS	11	$32 \times 32 \times 32$	$\eta = \zeta = 10^{-8}$

Table 1: Fixed PDE parameters used in our experiments.

Transformers for Solving PDEs. Transformers are increasingly being utilized for modeling physical systems or PDEs. Previous works can be divided into using Transformers for applying temporal self-attention (Geneva & Zabaras, 2020) to model the temporal dependencies or applying spatial self-attention for capturing spatial dependencies of the PDE (Cao, 2021; Li et al., 2023a;b). Applying the spatial self-attention as in Fourier and Galerkin Transformer (Cao, 2021) or in OFormer (Li et al., 2023a) yields a neural operator (Kovachki et al., 2021) endowing the model with spatial ZSSR capabilities. Since these models do not consider time as an additional input, they are not time-continuous and, hence, fixed to a trained temporal discretization. To remedy the issue, the diffusion-inspired temporal Transformer operator (Ovadia et al., 2023) uses the time to condition the input solution, thereby supporting a flexible temporal discretization for inference. VCNeFs use spatial self-attention similar to the Fourier and Galerkin Transformers as well as the OFormer. However, our architecture is time-continuous by employing a conditional neural field that modulates spatio-temporal coordinates based on ICs and PDE parameters. DiTTO uses a UNet architecture that is enhanced with self-attention and conditioned on time by modulating the activations with scaling. In contrast, our architecture mainly relies on a transformer architecture with spatial (linear) self-attention and a modulated neural field.

Solving Parametric PDEs. Although ML-based methods have shown great success in solving PDEs, they often do not consider PDE parameters as input, resulting in failures to generalize to unseen parameter values. Recent works such as CAPE (Takamoto et al., 2023), PDERefiner (Lippe et al., 2023), and MP-PDE (Brandstetter et al., 2022) consider the PDE parameters as additional model input. Along the same lines, our proposed model also considers the PDE parameter to improve the generalization error of unseen PDE parameter values.

PDE	Model	nRMSE (↓)	bRMSE (↓)
Burgers	FNO	0.0987	0.0225
	MP-PDE	0.3046 (+208.7%)	0.0725 (+221.7%)
	UNet	**0.0566** (-42.6%)	0.0259 (+14.7%)
	CORAL	0.2221 (+125.1%)	0.0515 (+128.2%)
	Galerkin	0.1651 (+67.3%)	0.0366 (+62.3%)
	OFormer	0.1035 (+4.9%)	0.0215 (-4.5%)
	VCNeF	0.0824 (-16.5%)	0.0228 (+1.3%)
	VCNeF-R	0.0784 (-20.6%)	**0.0179** (-20.8%)
Advection	FNO	0.0190	0.0239
	MP-PDE	0.0195 (+2.7%)	0.0283 (+18.4%)
	UNet	**0.0079** (-58.4%)	0.0129 (-45.9%)
	CORAL	0.0198 (+4.3%)	0.0127 (-46.8%)
	Galerkin	0.0621 (+227.1%)	0.0349 (+46.2%)
	OFormer	0.0118 (-38.0%)	0.0073 (-69.6%)
	VCNeF	0.0165 (-13.0%)	0.0088 (-63.2%)
	VCNeF-R	0.0113 (-40.5%)	**0.0040** (-83.3%)
1D CNS	FNO	0.5722	1.9797
	UNet	0.2270 (-60.3%)	**1.0399** (-47.5%)
	CORAL	0.5993 (+4.7%)	1.5908 (-19.6%)
	Galerkin	0.7019 (+22.7%)	3.0143 (+52.3%)
	OFormer	0.4415 (-22.9%)	2.0478 (+3.4%)
	VCNeF	0.2943 (-48.6%)	1.3496 (-31.8%)
	VCNeF-R	**0.2029** (-64.5%)	1.1366 (-42.6%)
2D CNS	FNO	0.5625	0.2332
	UNet	1.4240 (+153.2%)	0.3703 (+58.8%)
	Galerkin	0.6702 (+19.2%)	0.8219 (+252.4%)
	VCNeF	**0.1994** (-64.6%)	**0.0904** (-61.2%)
3D CNS	FNO	0.8138	6.0407
	VCNeF	**0.7086** (-12.9%)	**4.8922** (-19.0%)

Table 2: Errors of surrogate models trained and tested on the same spatial and temporal resolution with a fixed PDE parameter. nRMSE and bRMSE denote the normalized and RMSE at the boundaries, respectively. Values in parentheses indicate the percentage deviation to the FNO as a strong baseline in terms of accuracy, memory consumption, and runtime. Underlined values indicate the second-best errors.

Implicit Neural Representations (INR). Neural Fields (NeFs) has become widely popular in signal processing (Sitzmann et al., 2020), computer vision (Mescheder et al., 2019), computer graphics (Chu et al., 2022), and recently in SciML for solving PDEs (Chen et al., 2023b; Yin et al., 2023; Chen et al., 2023a; Serrano et al., 2023). The prevalent INR models for PDE solving follow the "Encode-Process-Decode" paradigm. DiNO (Yin et al., 2023) has an encoder, a Neural ODE (NODE) to model dynamics, and a decoder. CORAL, an improvement to DiNO, has a two-step training procedure whereby the input and output INR modules with shared parameters are trained first, and subsequently, the dynamics modeling block is trained using the learned latent codes. DiNO and CORAL utilize an INR to encode and decode the PDE solution in a latent space and a NODE to propagate the dynamics in latent space, while our approach utilizes a neural field to represent the entire PDE solution encompassing both the spatial and temporal dependencies within a shared space.

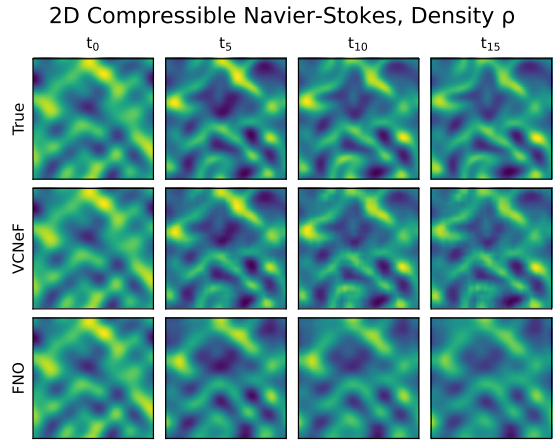


Figure 3: Example predictions for density of 2D CNS.

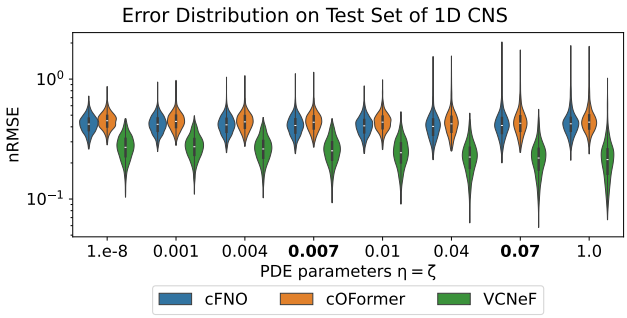


Figure 4: Error distribution of samples in the test set of 1D CNS. Boldfaced are the unseen PDE parameter values.

7. Experiments

We aim to answer the following research questions.

Q1: How effective are VCNeFs compared to the state-of-the-art (SOTA) methods when trained and tested for the same PDE parameter value?

Q2: How well can VCNeFs generalize to PDE parameter values not seen during training?

Q3: How well can VCNeFs do spatial and temporal zero-shot super-resolution?

Q4: Does training on initial conditions sampled from training trajectories improve the accuracy?

Q5: Does the vectorization provide a speed-up, and what is the model's scaling behavior?

7.1. Datasets

We conduct experiments on the following hydrodynamical equations of parametric PDE datasets from

PDE	Spatial res.	Model	nRMSE (↓)	bRMSE (↓)
Burgers	256	FNO	0.0987	0.0225
		OFormer	0.1035	**0.0215**
		VCNeF	**0.0824**	0.0228
	512	FNO	0.2557	0.0566
		OFormer	0.1092	**0.0228**
		VCNeF	**0.0832**	0.0229
	1024	FNO	0.3488	0.0766
		OFormer	0.1102	0.0233
		VCNeF	**0.0839**	**0.0230**
1D CNS	256	FNO	0.5722	1.9797
		OFormer	0.4415	2.0478
		VCNeF	**0.2943**	**1.3496**
	512	FNO	0.6610	2.7683
		OFormer	0.4657	2.5618
		VCNeF	**0.2943**	**1.3502**
	1024	FNO	0.7320	3.5258
		OFormer	0.4655	2.5526
		VCNeF	**0.2943**	**1.3510**
2D CNS	64×64	FNO	0.5625	0.2332
		VCNeF	**0.1994**	**0.0904**
	128×128	FNO	0.8693	2.3944
		VCNeF	**0.4016**	**0.2280**
3D CNS	$32 \times 32 \times 32$	FNO	0.8138	6.0407
		VCNeF	**0.7086**	**4.8922**
	$64 \times 64 \times 64$	FNO	0.9452	8.7068
		VCNeF	**0.7228**	**5.1495**
	$128 \times 128 \times 128$	FNO	1.0077	9.8633
		VCNeF	**0.7270**	**5.3208**

Table 3: Normalized RMSEs and RMSEs at the boundary for the spatial ZSSR experiments. The models are trained on the spatial resolutions given in the grey columns and are only tested on the additional spatial resolutions. The temporal resolution is the same as during training.

PDE	Temporal res.	Model	nRMSE (↓)	bRMSE (↓)
Burgers	41	FNO	0.0987	**0.0225**
		CORAL	0.2221	0.0515
		VCNeF	**0.0824**	0.0228
	101	FNO + Interp.	0.1116	0.0279
		CORAL	0.5298	0.1682
		VCNeF	**0.0829**	**0.0234**
	201	FNO + Interp.	0.1154	0.0294
		CORAL	0.6186	0.2013
		VCNeF	**0.0831**	**0.0236**
Advection	41	FNO	0.0190	0.0239
		CORAL	0.0198	0.0127
		VCNeF	**0.0165**	**0.0088**
	101	FNO + Interp.	0.0234	0.0242
		CORAL	0.8970	0.4770
		VCNeF	**0.0165**	**0.0088**
	201	FNO + Interp.	0.0258	0.0247
		CORAL	0.9656	0.5376
		VCNeF	**0.0165**	**0.0088**
1D CNS	41	FNO	0.5722	1.9797
		CORAL	0.5993	1.5908
		VCNeF	**0.2943**	**1.3496**
	82	FNO + Interp.	0.5667	1.9639
		CORAL	1.1524	3.7960
		VCNeF	**0.2965**	**1.3741**
3D CNS	11	FNO	0.8138	6.0407
		VCNeF	**0.7086**	**4.8922**
	21	FNO + Interp.	0.8099	6.1938
		VCNeF	**0.7106**	**5.1446**

Table 4: Error values for temporal ZSSR. The models are trained on the temporal resolution given in the grey columns and additionally tested on higher temporal resolutions. The spatial resolution is $s = 256$ for 1D and $s = 32 \times 32 \times 32$ for 3D (same as during training).

PDEBench (Takamoto et al., 2022): **1D Burgers'**, **1D Advection**, and **1D, 2D, and 3D compressible Navier-Stokes** (CNS).

7.2. Setup and Baselines

We train and test the models with a single timestep as an initial condition and predict multiple future steps. This setting is well-motivated by applications that need solutions to initial value problems. We use the PDE parameter values in Table 1 for the experiments where we train and test with one fixed PDE parameter value. Additionally, we conduct experiments on multiple PDE parameter values, including values not seen during training, to test the models' generalization capabilities.

We choose FNO (Li et al., 2020a), MP-PDE (Brandstetter et al., 2022), UNet from PDEArena (Gupta & Brandstetter, 2023), CORAL (Serrano et al., 2023), Galerkin Transformer (Cao, 2021), and OFormer (Li et al., 2023a) as baselines. The predictions of FNO, UNet, OFormer, and Galerkin Transformer are achieved in an autoregressive fashion, while VCNeF predicts the entire trajectory of the simulation directly in one forward pass or single shot.

The reported numbers are the mean values of two training runs with different initializations, and the full results are in Appendix F.

7.3. Results

Q1. We test VCNeF's generalization ability to different ICs by evaluating the models on the corresponding test sets of PDEBench. Table 2 shows the errors of the baseline models trained and tested on the selected PDEs and Figure 3 an example prediction for 2D CNS. The results demonstrate that our model performs competitively with SOTA methods for solving PDEs.

Q2. To evaluate the performance and effectiveness of VCNeF and its PDE parameter conditioning, we train VCNeF, a PDE parameter conditioned FNO (cFNO; Takamoto et al. (2023)), and OFormer that has been modified to accept PDE

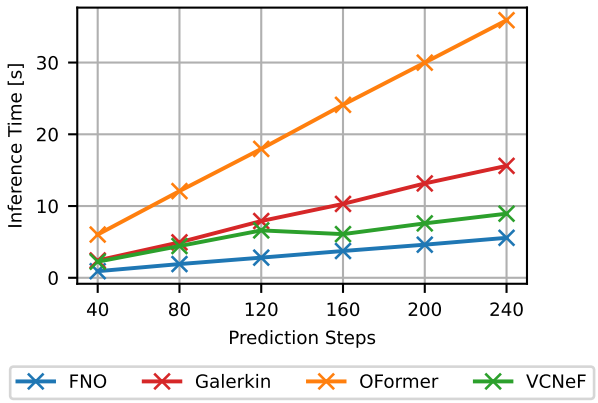


Figure 5: Inference times of models trained on Burgers with $s = 256$, predicting different numbers of timesteps in the future.

parameter as additional input[2] on a set of PDE parameter values and test them on another unseen set of parameter values. Figure 4 shows the error distribution for 1D CNS. We observe that VCNeF generalizes better to unseen PDE parameters than the other baseline models. See Appendix F.2 for results on other 1D PDEs (Burgers' and Advection).

Q3. We also evaluate the model's ability to do spatial and temporal ZSSR, by training the model with a reduced spatial and temporal resolution and testing for a higher resolution. The results in Table 3 show that the OFormer and VCNeF seem to have spatial ZSSR capabilities since there is no significant increase in error even when the spatial resolution is four times the training resolution. For the case of FNO on 1D Burgers, we observe a near multiplicative increase of error as the factor of resolution increases (2x and 4x).

Temporal ZSSR means that the model predicts a trajectory with a smaller temporal step size Δt than encountered during training (i.e., the trajectory is divided into more timesteps). This is possible due to VCNeF's support for continuous-time inference. We use FNO with linear interpolation between the time steps as a baseline since we train FNO in an autoregressive fashion, which fixes the temporal discretization of the trained model. Table 4 shows a negligible increase in error for VCNeF, meaning that the model has temporal ZSSR capabilities and learns the dependency between solution and time. Doing interpolation seems to be only effective for smooth targets such as CNS.

Q4. Conditioning VCNeF on random starting points sampled in the temporal extent of the trajectory $u(t, \boldsymbol{X})$ with $t \sim \mathcal{U}\{0, T\}$ in addition to the IC $u(0, \boldsymbol{X})$ during training further enhances the performance. Table 2 shows the performance improvement of the randomized VCNeF (VCNeF-R) over the non-randomized training of VCNeF. We attribute the boost in performance of VCNeF-R to the diversity of encountered ICs that the model processes during training.

[2]We encode the parameter values as an additional channel.

Q5. We also measure the inference times of the proposed and baseline models. The times are the raw inference times without measuring the time needed to transfer the data from the host device to the GPU on a single NVIDIA A100-SXM4 80GB GPU. We measure the time on the test set of 1D Burgers (i.e., 1k ICs, predicting 40 to 240 timesteps in the future) with a batch size of 64 and a spatial resolution of $s = 256$. The times in Figure 5 demonstrate that VCNeF is significantly faster than the other transformer-based counterparts. However, the speed-up is traded for a higher GPU memory consumption as shown in Table 20. To mitigate the high GPU memory consumption, the VCNeF can also be used to generate the solutions sequentially, drastically reducing the GPU memory consumption but resulting in a higher inference time for the trajectory. Thus, the proposed model allows for a trade-off between the inference time and GPU memory consumption when generating the solutions (see Appendix F.3).

8. Limitations

The limitations are two-fold. (i) Using VCNeF to generate trajectories consisting of multiple timesteps in parallel is limited to GPUs with larger memory and hence incurs higher costs. Nevertheless, the sequential VCNeF does not have this limitation (see Appendix F.3). (ii) VCNeF utilizes 2D and 3D convolutional layers for 2D and 3D PDEs to partition the spatial domain into (non-overlapping) patches, which restricts the model to regular grids. However, the VCNeF model can be used with a different encoder, such as a pointwise MLP or a graph neural network, which allows the application of the VCNeF model to irregular grids.

9. Conclusion

In this work, we have designed an effective Neural PDE Solver, *Vectorized Conditional Neural Field*, based on the conditional neural fields framework and demonstrated its generalization capabilities across multiple axes of desiderata: spatial, temporal, ICs, and PDE parameters. As a future work, we aim to experiment on turbulent simulations, improve the model design further with adaptive time-stepping, investigate sophisticated strategies for conditioning the neural fields (Rebain et al., 2023), and test physics-informed losses. Additionally, we plan to investigate the effect of the temporal discretization the model was exposed to during training on the temporal zero-shot super-resolution capabilities of the model.

Impact Statement

Accelerated and efficient Neural PDE Solvers help reduce cost in running cost-prohibitive simulations such as weather forecasting and cyclone predictions. As a consequence, disaster preparedness, the design and manufacturing timeline can be accelerated resulting in life and cost savings and decreased CO_2 emissions. As a negative side-effect, we cannot rule out the possibility of misuse by bad actors since fluid dynamic simulations are used to design military equipment.

Acknowledgements

Funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2075 – 390740016. We acknowledge the support of the Stuttgart Center for Simulation Science (SimTech). The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Marimuthu Kalimuthu, Daniel Musekamp, and Mathias Niepert. Additionally, we acknowledge the support of the German Federal Ministry of Education and Research (BMBF) as part of InnoPhase (funding code: 02NUK078). Furthermore, we acknowledge the support of the European Laboratory for Learning and Intelligent Systems (ELLIS) Unit Stuttgart. Lastly, we thank Makoto Takamoto for the insightful discussions.

References

Boussif, O., Bengio, Y., Benabbou, L., and Assouline, D. Magnet: Mesh agnostic neural pde solver. *Advances in Neural Information Processing Systems*, 35:31972–31985, 2022.

Brandstetter, J., Worrall, D. E., and Welling, M. Message passing neural PDE solvers. *CoRR*, abs/2202.03376, 2022. URL https://arxiv.org/abs/2202.03376.

Cao, S. Choose a transformer: Fourier or galerkin. *CoRR*, abs/2105.14995, 2021. URL https://arxiv.org/abs/2105.14995.

Chen, C., Fan, Q., and Panda, R. Crossvit: Cross-attention multi-scale vision transformer for image classification. *CoRR*, abs/2103.14899, 2021. URL https://arxiv.org/abs/2103.14899.

Chen, H., Wu, R., Grinspun, E., Zheng, C., and Chen, P. Y. Implicit neural spatial representations for time-dependent pdes. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 5162–5177. PMLR, 2023a. URL https://proceedings.mlr.press/v202/chen23af.html.

Chen, P. Y., Xiang, J., Cho, D. H., Chang, Y., Pershing, G. A., Maia, H. T., Chiaramonte, M. M., Carlberg, K. T., and Grinspun, E. CROM: continuous reduced-order modeling of pdes using implicit neural representations. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023b. URL https://openreview.net/pdf?id=FUORz1tG8Og.

Chu, M., Liu, L., Zheng, Q., Franz, E., Seidel, H., Theobalt, C., and Zayer, R. Physics informed neural fields for smoke reconstruction with sparse data. *ACM Trans. Graph.*, 41(4):119:1–119:14, 2022. doi: 10.1145/3528223.3530169. URL https://doi.org/10.1145/3528223.3530169.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL https://arxiv.org/abs/2010.11929.

Geneva, N. and Zabaras, N. Transformers for modeling physical systems. *CoRR*, abs/2010.03957, 2020. URL https://arxiv.org/abs/2010.03957.

Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., and Pang, R. Conformer: Convolution-augmented transformer for speech recognition, 2020.

Gupta, J. K. and Brandstetter, J. Towards multi-spatiotemporal-scale generalized PDE modeling. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=dPSTDbGtBY.

Hao, Z., Wang, Z., Su, H., Ying, C., Dong, Y., Liu, S., Cheng, Z., Song, J., and Zhu, J. GNOT: A general neural operator transformer for operator learning. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 12556–12569. PMLR, 2023. URL https://proceedings.mlr.press/v202/hao23c.html.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020. URL http://proceedings.mlr.press/v119/katharopoulos20a.html.

Kovachki, N. B., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A. M., and Anandkumar, A. Neural operator: Learning maps between function spaces. *CoRR*, abs/2108.08481, 2021. URL https://arxiv.org/abs/2108.08481.

Krishnapriyan, A. S., Gholami, A., Zhe, S., Kirby, R. M., and Mahoney, M. W. Characterizing possible failure modes in physics-informed neural networks. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 26548–26560, 2021. URL https://arxiv.org/abs/2109.01050.

Li, Y., Si, S., Li, G., Hsieh, C.-J., and Bengio, S. Learnable fourier features for multi-dimensional spatial positional encoding, 2021.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. M., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *CoRR*, abs/2010.08895, 2020a. URL https://arxiv.org/abs/2010.08895.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. M., and Anandkumar, A. Neural operator: Graph kernel network for partial differential equations. *CoRR*, abs/2003.03485, 2020b. URL https://arxiv.org/abs/2003.03485.

Li, Z., Meidani, K., and Farimani, A. B. Transformer for partial differential equations' operator learning. *Trans. Mach. Learn. Res.*, 2023, 2023a. URL https://openreview.net/forum?id=EPPqt3uERT.

Li, Z., Shu, D., and Farimani, A. B. Scalable transformer for PDE surrogate modeling. *CoRR*, abs/2305.17560, 2023b. doi: 10.48550/arXiv.2305.17560. URL https://doi.org/10.48550/arXiv.2305.17560.

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, 2023c.

Lippe, P., Veeling, B. S., Perdikaris, P., Turner, R. E., and Brandstetter, J. Pde-refiner: Achieving accurate long rollouts with neural PDE solvers. *CoRR*, abs/2308.05732, 2023. doi: 10.48550/ARXIV.2308.05732. URL https://doi.org/10.48550/arXiv.2308.05732.

Lu, L., Jin, P., and Karniadakis, G. E. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *CoRR*, abs/1910.03193, 2019. URL http://arxiv.org/abs/1910.03193.

Maclaurin, D., Duvenaud, D., and Adams, R. P. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML workshop*, volume 238, 2015.

McCabe, M., Blancard, B. R., Parker, L. H., Ohana, R., Cranmer, M. D., Bietti, A., Eickenberg, M., Golkar, S., Krawezik, G., Lanusse, F., Pettee, M., Tesileanu, T., Cho, K., and Ho, S. Multiple physics pretraining for physical surrogate models. *CoRR*, abs/2310.02994, 2023. doi: 10.48550/ARXIV.2310.02994. URL https://doi.org/10.48550/arXiv.2310.02994.

Mescheder, L. M., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. Occupancy networks: Learning 3d reconstruction in function space. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 4460–4470. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00459. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Mescheder_Occupancy_Networks_Learning_3D_Reconstruction_in_Function_Space_CVPR_2019_paper.html.

Ovadia, O., Turkel, E., Kahana, A., and Karniadakis, G. E. Ditto: Diffusion-inspired temporal transformer operator. *CoRR*, abs/2307.09072, 2023. doi: 10.48550/ARXIV.2307.09072. URL https://doi.org/10.48550/arXiv.2307.09072.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS 2017 Workshop on Autodiff*, 2017. URL https://openreview.net/forum?id=BJJsrmfCZ.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. C. Film: Visual reasoning with a general conditioning layer. In McIlraith, S. A. and Weinberger, K. Q. (eds.), *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana,*

USA, February 2-7, 2018, pp. 3942–3951. AAAI Press, 2018. doi: 10.1609/AAAI.V32I1.11671. URL https://doi.org/10.1609/aaai.v32i1.11671.

Rahman, M. A., Ross, Z. E., and Azizzadenesheli, K. U-NO: U-shaped neural operators. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=j3oQF9coJd.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations. *CoRR*, abs/1711.10561, 2017. URL http://arxiv.org/abs/1711.10561.

Rebain, D., Matthews, M. J., Yi, K. M., Sharma, G., Lagun, D., and Tagliasacchi, A. Attention beats concatenation for conditioning neural fields. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=GzqdMrFQsE.

Ruoss, A., Delétang, G., Genewein, T., Grau-Moya, J., Csordás, R., Bennani, M., Legg, S., and Veness, J. Randomized positional encodings boost length generalization of transformers, 2023.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. W. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8459–8468. PMLR, 2020. URL http://proceedings.mlr.press/v119/sanchez-gonzalez20a.html.

Serrano, L., Boudec, L. L., Koupaï, A. K., Wang, T. X., Yin, Y., Vittaut, J., and Gallinari, P. Operator learning with neural fields: Tackling pdes on general geometries. *CoRR*, abs/2306.07266, 2023. doi: 10.48550/ARXIV.2306.07266. URL https://doi.org/10.48550/arXiv.2306.07266.

Sitzmann, V., Martel, J. N. P., Bergman, A. W., Lindell, D. B., and Wetzstein, G. Implicit neural representations with periodic activation functions. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/53c04118df112c13a8c34b38343b9c10-Abstract.html.

Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., and Niepert, M. Pdebench: An extensive benchmark for scientific machine learning. *NeurIPS*, 2022.

Takamoto, M., Alesiani, F., and Niepert, M. Learning neural PDE solvers with parameter-guided channel attention. *ICML*, abs/2304.14118, 2023. doi: 10.48550/arXiv.2304.14118. URL https://doi.org/10.48550/arXiv.2304.14118.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Xie, Y., Takikawa, T., Saito, S., Litany, O., Yan, S., Khan, N., Tombari, F., Tompkin, J., Sitzmann, V., and Sridhar, S. Neural fields in visual computing and beyond. *CoRR*, abs/2111.11426, 2021. URL https://arxiv.org/abs/2111.11426.

Yin, Y., Kirchmeyer, M., Franceschi, J., Rakotomamonjy, A., and Gallinari, P. Continuous PDE dynamics forecasting with implicit neural representations. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/pdf?id=B73niNjbPs.

VECTORIZED CONDITIONAL NEURAL FIELDS: A FRAMEWORK FOR SOLVING TIME-DEPENDENT PARTIAL DIFFERENTIAL EQUATIONS APPENDIX

CODE: HTTPS://GITHUB.COM/JHAGNBERGER/VCNEF/

A Comparison of Neural Architectures for PDE Solving **14**

B Additional VCNeF Model Details **14**

B.1 Vectorized Conditional Neural Field as a Neural Operator 14

B.2 Neural Architecture 15

B.3 Linear Attention 17

B.4 Ablation Study 18

C PDE Dataset Details **19**

C.1 1D Burgers' Equation 19

C.2 1D Advection Equation 19

C.3 1D, 2D, and 3D Compressible Navier-Stokes (CNS) Equations 19

D Baseline Models **20**

D.1 Fourier Neural Operator Baseline 20

D.2 Graph Neural Network Baseline 21

D.3 UNet Baselines 21

D.4 Implicit Neural Representation Baseline 21

D.5 Transformer Baselines 22

E Additional Experimental Details **22**

E.1 Used PDE Parameters 22

E.2 Loss Function 23

E.3 Model's Hyperparameters 23

E.4 Evaluation Metrics 25

E.5 Randomized Starting Points Training 25

F Additional Experimental Results **25**

F.1 (Q1): Comparison to state-of-the-art baselines 26

F.2 (Q2): Generalization to Unseen PDE Parameter Values 29

F.3 (Q5): Inference Time and Memory Consumption 32

G Qualitative Results **32**

A. Comparison of Neural Architectures for PDE Solving

Table 5 shows the most important properties of ML models for solving PDEs and compares three families of models. The VCNeF combines both worlds of neural fields (e.g., PINNs) and neural operators. PINNs do not leverage the spatial dependencies among the queried coordinates since they produce the output for each queried coordinate independently. Meanwhile, neural operators map to a set of solution points and leverage the dependencies between the regressed points. However, neural operator implementations have usually limited support for time continuity, while PINNs are time-continuous. VCNeF, therefore, combines the advantages of both worlds by leveraging spatial dependencies by generating a set of solution points and being continuous in time.

Model family	Model	Initial value generalization	PDE parameter generalization	ZSSR Spatial	ZSSR Temporal	Models spatial dependencies with self-attention
Neural Operator	FNO[1]	✓	✗	✓	✗	✗
	OFormer	✓	✗	✓	✗	✓
	cFNO[1]	✓	✓	✓	✗	✗
	cOFormer	✓	✓	✓	✗	✓
Neural Field	PINN	✗	✗	✓	✓	✗
Conditional Neural Field	CORAL	✓	✗	✓	✓	✗
	VCNeF (ours)	✓	✓	✓	✓	✓

Table 5: Overview of distinct properties of benchmark baselines and our proposed VCNeF model. [1]Refers to FNO that applies the Fourier transform only to the spatial domain and is trained in an autoregressive fashion.

B. Additional VCNeF Model Details

B.1. Vectorized Conditional Neural Field as a Neural Operator

Neural operators are theoretically time-continuous. However, current neural operator implementations have limited support for being time continuous. Our proposed architecture for 1D PDEs can be considered as a neural operator implementation that is conditioned on time to be temporally continuous. As a time-continuous neural operator implementation, VCNeF learns a mapping between the initial condition (i.e., input function $u(0, \cdot)$) and the solution at time t (i.e., output function $u(t, \cdot)$). This can mathematically be expressed as

$$f_\theta(u(0, \boldsymbol{x}))(t) \approx u(t, \boldsymbol{x}) \tag{12}$$

where f_θ denotes the neural network or neural operator. The VCNeF model can be decomposed into the following layers that are identical to the Fourier Neural Operator of Li et al. (2020a).

Lifting. Encoding the input functions with a shared pointwise linear layer represents a lifting of the input function $u^{(0)}(\boldsymbol{x}) := u(0, \boldsymbol{x})$ that lifts the function to a higher dimensional space.

$$u^{(1)} = u^{(0)}\boldsymbol{W} + \boldsymbol{b} \tag{13}$$

which can be equivalently represented as a function

$$u^{(1)}(\boldsymbol{x}) = (u^{(0)}\boldsymbol{W} + \boldsymbol{b})(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \mathbb{X} \tag{14}$$

Iterative Updates. Following Cao (2021) and Kovachki et al. (2021) and interpreting the columns j of the matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ as learnable basis functions q_j, k_j, v_j yields that the encoding of the initial condition with self-attention of the Linear Transformer blocks can be seen as a kernel integral transformation as follows

$$
\begin{aligned}
Q = u^{(n)}W_Q + b_Q \qquad K = u^{(n)}W_K + b_K \qquad V = u^{(n)}W_V + b_V \\
u^{(n+1)} = \texttt{Linear\_Attn}(Q,K,V) = \frac{\Phi(Q)\Phi(K)^\top V}{\Phi(Q)^\top \Phi(K)} = \frac{1}{\Phi(Q)^\top \Phi(K)}\Phi(Q)\Phi(K)^\top V \\
u_{i,j}^{(n+1)} = \frac{1}{\sum_{l=1}^{s}\Phi(Q_i)^\top\Phi(K_l)}\sum_{l=1}^{s}\left(\Phi(Q_i)^\top\Phi(K_l)\right)V_{l,j} \approx \int_{\mathbb{X}}\frac{\kappa(\boldsymbol{x_i},\xi)}{\int_{\mathbb{X}}\kappa(\boldsymbol{x_i},\psi)d\psi}v_j(\xi)d\xi
\end{aligned}
\tag{15}
$$

where the learnable kernel $\kappa(\boldsymbol{x_i},\xi)$ is approximated by $\Phi(Q_i)^\top\Phi(K_l)$. It can also be expressed as a function

$$
u^{(n+1)}(\boldsymbol{x}) = \int_{\mathbb{X}}\frac{\kappa(\boldsymbol{x},\xi)}{\int_{\mathbb{X}}\kappa(\boldsymbol{x},\psi)d\psi}(u^{(n)}W_V + b_V)(\xi)d\xi \qquad \forall \boldsymbol{x}\in\mathbb{X} \tag{16}
$$

Iterative Time Injection. The modulation blocks can be considered as a time injection mechanism that performs the kernel integral transformation from Equation (15) and multiplies the latent representation $u^{(m+1)}$ with a spatial and temporal dependent function g.

$$
\begin{aligned}
g := g(t,\boldsymbol{x}) = \texttt{MLP}(t \parallel \boldsymbol{x}) \\
u^{(m+1)} = \texttt{Modulation\_Block}(g, u^{(m)}) = \sigma\left(\texttt{Linear\_Attn}(Q,K,V)\right)\circ g \\
\text{with } Q = u^{(m)}W_Q + b_Q \qquad K = u^{(m)}W_K + b_K \qquad V = u^{(m)}W_V + b_V \\
u_{i,j}^{(m+1)} = \sigma\left(\int_{\mathbb{X}}\frac{\kappa(\boldsymbol{x_i},\xi)}{\int_{\mathbb{X}}\kappa(\boldsymbol{x_i},\psi)d\psi}v_j(\xi)d\xi\right)g_j(t,\boldsymbol{x_i})
\end{aligned}
\tag{17}
$$

which can also be represented as a function

$$
\begin{aligned}
g := g(t,\boldsymbol{x}) = \texttt{MLP}(t \parallel \boldsymbol{x}) \\
u^{(m+1)}(t,\boldsymbol{x}) = \sigma\left(\int_{\mathbb{X}}\frac{\kappa(\boldsymbol{x},\xi)}{\int_{\mathbb{X}}\kappa(\boldsymbol{x},\psi)d\psi}(u^{(m)}W_V + b_V)(\xi)d\xi\right)g(t,\boldsymbol{x}) \qquad \forall \boldsymbol{x}\in\mathbb{X}, t\in(0,T]
\end{aligned}
\tag{18}
$$

Projection. The final hidden representation, which represents the solution $u(t,\boldsymbol{x})$, is projected back to the physical space with a pointwise MLP.

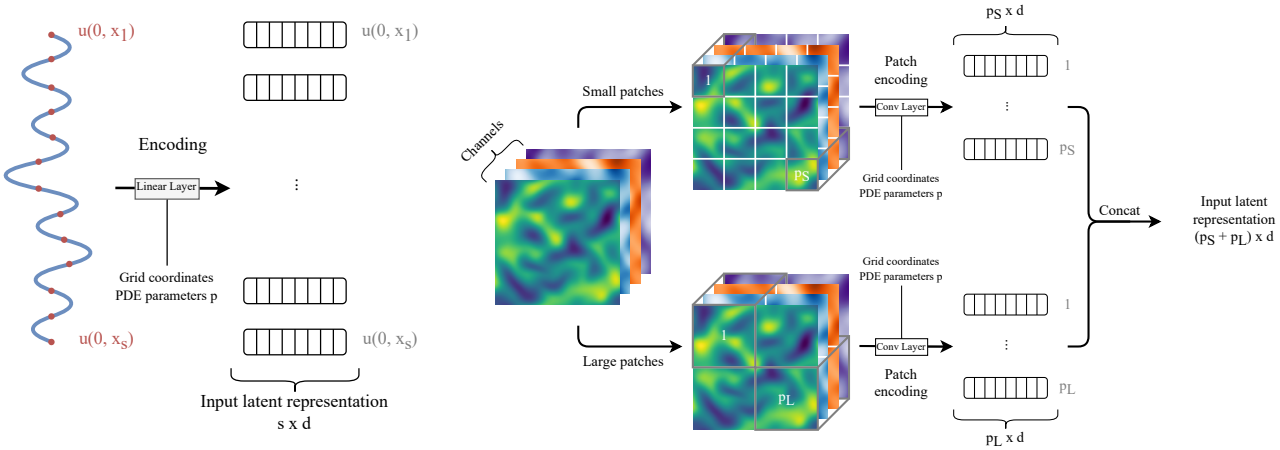
$$
u = u^{(m+n)}\boldsymbol{W} + \boldsymbol{b} \tag{19}
$$

which can be equivalently represented as a function

$$
u(\boldsymbol{x}) = (u^{(m+n)}\boldsymbol{W} + \boldsymbol{b})(\boldsymbol{x}) \qquad \forall \boldsymbol{x}\in\mathbb{X} \tag{20}
$$

B.2. Neural Architecture

Encoding of the Initial Condition. Depending on the dimensionality of the PDE, different mechanisms are utilized to generate latent representations of the IC. For 1D PDEs (Figure 6a), each solution point in space is projected to a latent representation by applying a linear layer that is shared across the spatial points. For 2D PDEs (Figure 6b), the initial condition is divided into patches of different sizes and each patch is projected to a latent representation by a 2D convolutional layer. The IC of 3D PDEs is also divided into non-overlapping patches to reduce the computational costs. The latent representations of the small and large patches are concatenated together. p_S denotes the number of small patches and depends on the spatial resolution of the input as well as the size of the patch. The same is true for p_L which denotes the large patches. In addition to the initial condition, the encoding mechanism takes the grid as positional information and the PDE parameters $\boldsymbol{p}$ as input. The 1D case could also be seen as a special case of the patches with a 1D input and only one patch size of 1×1.



(a) Latent representation for 1D PDEs

(b) Latent representation for 2D PDEs (multi-scale patches)

Figure 6: Encoding mechanisms for the initial condition of 1D and 2D PDEs.

Transformer-based Vectorized Conditional Neural Field. Figure 7 shows the detailed architecture of the proposed VCNeF with the modulation blocks that modulate the latent representation of the input coordinates based on the IC. Linear self-attention allows an information flow between different spatio-temporal coordinates to capture spatial dependencies.

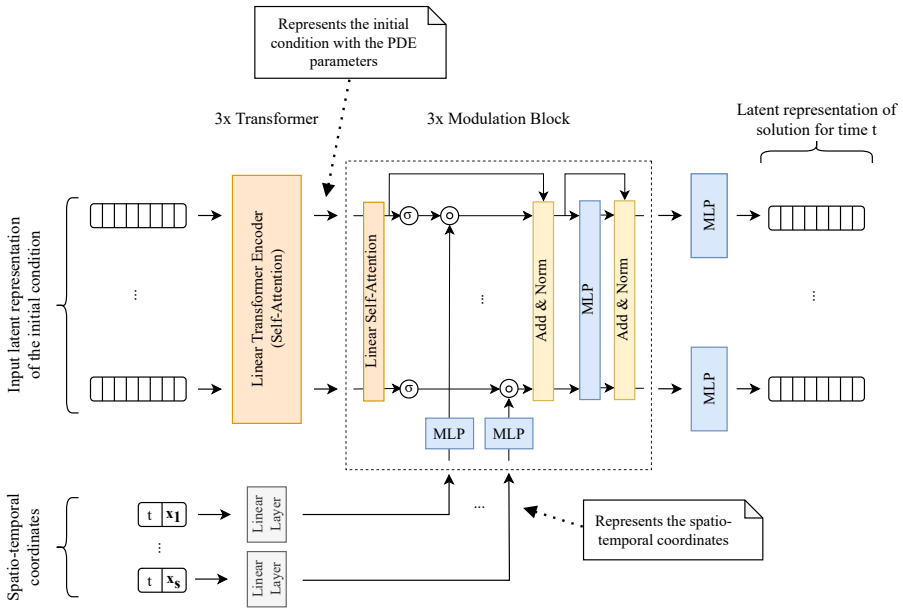


Figure 7: Architecture of VCNeF for solving time-dependent PDEs. The input latent representation is generated via the mechanisms in Figure 6. The modulation block (dashed rectangle) contains a non-linearity σ, linear self-attention, two shared MLPs, and a pointwise multiplication $\circ$ (scaling of FiLM). Additionally, it contains residual connections and layer normalization (Add & Norm). The solution's latent representation is mapped to physical space with the mechanism in Figure 8.

Decoding of Solution's Latent Representation. Similar to the encoding, the decoding of the solution's latent representation depends on the dimensionality of the PDE. For 1D PDEs, a shared MLP is applied to decode the latent representation into the physical representation. Figure 8 shows the mechanism to map the solution's latent representation back to the physical space for a 2D PDE. The mechanism for 3D PDEs is similar to the mechanism for 2D PDEs, except that it operates

on 3D patches instead of 2D patches.

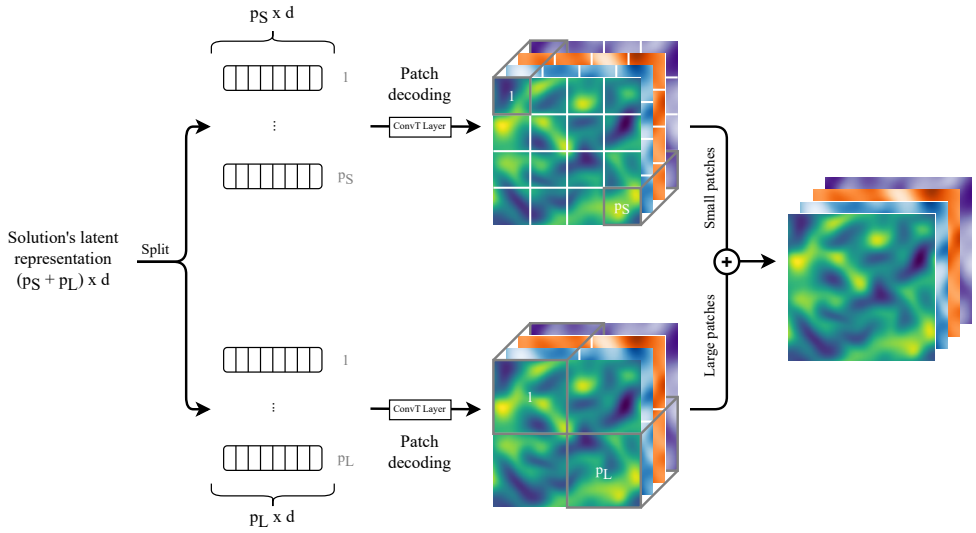


Figure 8: Solution decoding of the VCNeF for solving 2D PDEs. The solution's latent representation is split into the latent representation for the small and large patches. Thereafter, the latent representations are projected to patches with shared 2D convolution transposed layers. The final output is the weighted sum of the small and large patches.

B.3. Linear Attention

Katharopoulos et al. (2020) reformulate the attention calculation to

$$\texttt{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \texttt{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V} = \boldsymbol{V'} \Leftrightarrow \boldsymbol{V}_i' = \frac{\sum_{j=1}^{N} \texttt{sim}(\boldsymbol{Q}_i, \boldsymbol{K}_j)\boldsymbol{V}_j}{\sum_{j=1}^{N} \texttt{sim}(\boldsymbol{Q}_i, \boldsymbol{K}_j)} \tag{21}$$

where $\boldsymbol{V}_i'$ denotes the i-th row of matrix $\boldsymbol{V}$ and $\texttt{sim}(\boldsymbol{q}, \boldsymbol{k}) = \exp(\frac{\boldsymbol{q}^\top \boldsymbol{k}}{\sqrt{d}})$. $\texttt{sim}(\cdot, \cdot)$ is a similarity function that measures the similarity of two vectors. Each function that takes two vectors as an input and outputs a non-negative real value has an interpretation as a similarity function. All kernel functions $\kappa(\boldsymbol{x}, \boldsymbol{y}) : (\mathbb{R}^d \times \mathbb{R}^d) \to \mathbb{R}_+$ with non-negative output values also satisfies this property. Since kernel functions can be rewritten as the dot product of two output vectors of a feature function $\Phi(\cdot)$ that maps the input to some high-dimensional space (Mercer's theorem), the kernel function can be written as $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x})^\top \Phi(\boldsymbol{y})$. This leads to a new interpretation of the attention equation:

$$\boldsymbol{V}_i' = \frac{\sum_{j=1}^{N} \texttt{sim}(\boldsymbol{Q}_i, \boldsymbol{K}_j)\, \boldsymbol{V}_j}{\sum_{j=1}^{N} \texttt{sim}(\boldsymbol{Q}_i, \boldsymbol{K}_j)} = \frac{\sum_{j=1}^{N} \Phi(\boldsymbol{Q}_i)^\top \Phi(\boldsymbol{K}_j)\, \boldsymbol{V}_j}{\sum_{j=1}^{N} \Phi(\boldsymbol{Q}_i)^\top \Phi(\boldsymbol{K}_j)} = \frac{\Phi(\boldsymbol{Q}_i)^\top \sum_{j=1}^{N} \Phi(\boldsymbol{K}_j)\, \boldsymbol{V}_j^\top}{\Phi(\boldsymbol{Q}_i)^\top \sum_{j=1}^{N} \Phi(\boldsymbol{K}_j)} \tag{22a}$$

Since $\Phi(\boldsymbol{Q}_i)^\top$ is independent of the index i of the sum, the associative property of matrix multiplication can be applied and $\Phi(\boldsymbol{Q}_i)^\top$ can be pulled in front of the sum. The value of $\sum_{j=1}^{N} \Phi(\boldsymbol{K}_j)\boldsymbol{V}_j^\top$ and $\sum_{j=1}^{N} \Phi(\boldsymbol{K}_j)$ needs only to be calculated once because they can be reused (time complexity of $\mathcal{O}(N)$) and $\boldsymbol{V}_i$ needs to be calculated for all N tokens (time complexity of $\mathcal{O}(N)$). This results in linear time and space complexity. Equation (22a) can also be written in matrix notation:

$$\texttt{Linear\_Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \frac{\left(\Phi(\boldsymbol{Q})\Phi(\boldsymbol{K})^\top\right)\boldsymbol{V}}{\Phi(\boldsymbol{Q})^\top \Phi(\boldsymbol{K})} = \frac{\Phi(\boldsymbol{Q})\left(\Phi(\boldsymbol{K})^\top \boldsymbol{V}\right)}{\Phi(\boldsymbol{Q})^\top \Phi(\boldsymbol{K})} \tag{23}$$

where the feature function Φ is applied row-wise to the matrix.

Feature Function. We use the feature function

$$\Phi(x) = \texttt{ELU}(x) + 1$$
$$\texttt{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \exp(x) - 1, & \text{if } x \leq 0 \end{cases} \tag{24}$$

as proposed by Katharopoulos et al. (2020).

B.4. Ablation Study

We conduct an ablation study on the prominent parts of the proposed architecture. Namely, the self-attention mechanism that allows the model to capture spatial dependencies and the conditioning mechanism that is used to condition the neural field. For 2D PDEs, we also study the effect on the model's performance for patches of one size and the multi-scale patching mechanism with small and large patches. Additionally, we compare linear attention (Katharopoulos et al., 2020) with vanilla attention (Vaswani et al., 2017) in terms of training time and GPU memory consumption. For simplicity, we perform the ablation study mainly on the 1D Burgers' and 2D CNS datasets.

Self-Attention, Conditioning Mechanism, and Patch Generation. Table 6 shows the results of the proposed model with and without self-attention as well as with different modulation mechanisms to condition the neural field. Table 7 presents the different results for patches of one size vs multi-scale patching mechanism.

PDE	Self-attention	Conditioning mechanism	nRMSE (↓)	bRMSE (↓)
1D Burgers	✓	Modulation with scaling	**0.0824**	**0.0228**
	✗	Modulation with scaling	0.8890	0.3242
	✓	Modulation with scaling and shifting	0.0945	0.0291

Table 6: Ablation study for attention mechanism and conditioning mechanism of our proposed VCNeF model.

PDE	Patches	nRMSE (↓)	bRMSE (↓)
2D CNS	Small and large	**0.1994**	**0.0904**
	Only large	0.4569	0.1982

Table 7: Ablation study for the multi-scale mechanism of our proposed VCNeF model.

Vanilla Attention and Linear Attention. The Linear Transformer and the linear self-attention component in the proposed architecture can be replaced with vanilla attention or some arbitrary attention mechanism. We choose linear attention since it promises a speed-up for long sequences (i.e., fine resolution of the spatial domain) compared to vanilla attention. Table 8 shows empirical results for training the transformer-based VCNeF on the 1D Burgers' PDE. We observe that the memory of linear attention increases linearly and of vanilla attention quadratically. Double the spatial resolution corresponds to double the number of tokens yielding an increased memory and time consumption. Training the VCNeF with vanilla attention requires more than 640 GiB while the VCNeF with linear attention requires only 99.4 GiB. We use the vanilla attention implementation of Katharopoulos et al. (2020) for a fair comparison to the non-optimized linear attention implementation.

PDE	Spatial resolution (# tokens)	Attention type	GPU memory	Time per epoch
1D Burgers	256	Vanilla	72.6 GiB	28 s
		Linear	31.4 GiB	18 s
	512	Vanilla	223.4 GiB	78 s
		Linear	53.8 GiB	32 s
	1024	Vanilla	>640 GiB	N/A
		Linear	99.4 GiB	62 s

Table 8: GPU memory consumption and training time per epoch for the VCNeF with vanilla attention (scaled dot-product attention) and linear attention on the 1D Burgers train set. The values refer to training with a batch size of 64 on 4x NVIDIA A100-SXM4 80GB GPUs using data parallelism. The number of queried timesteps N_t is 40. Time per epoch includes the time that is needed to load the data and transfer it to the GPUs.

C. PDE Dataset Details

We conduct experiments on the following four challenging hydrodynamical equations of time-dependent parametric PDE datasets from PDEBench (Takamoto et al., 2022).

C.1. 1D Burgers' Equation

The Burgers' PDE models the non-linear behaviour and diffusion process in fluid dynamics and is expressed as

$$\partial_t u(t,x) + u(t,x)\partial_x u(t,x) = \frac{\nu}{\pi}\partial_{xx} u(t,x) \tag{25}$$

kinematic viscosity

where the PDE parameter ν denotes the diffusion coefficient. Our dataset contains solutions for x $\in$ (-1, 1) with a maximum resolution of 1024 spatial discretization points and t $\in$ (0, 2] with a maximum resolution of 201 temporal discretization steps including the initial condition. We subsample the data along the temporal and spatial domain yielding a trajectory of 41 time steps where each snapshot has a spatial resolution of 256.

C.2. 1D Advection Equation

The Advection PDE models pure advection behaviour without non-linearity. It is written as

$$\partial_t u(t,x) + \beta\, \partial_x u(t,x) = 0 \tag{26}$$

advection velocity

where the PDE parameter β denotes the advection velocity. Similar to 1D Burgers, we subsample the data to get a trajectory of 41 time steps each with a spatial resolution of 256.

C.3. 1D, 2D, and 3D Compressible Navier-Stokes (CNS) Equations

The Navier-Stokes equations (Equations 27a to 27c) are a compressible version of fluid dynamics equations that describe the flow of a fluid. Thus, the equations are important for Computational Fluid Dynamics (CFD) applications. Equation 27a refers to the mass continuity equation which is also called as transport equation or equation of conservation of mass, Equation 27b describes the conservation of momentum, and Equation 27c represents energy conservation.

mass density (mass per unit volume)

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{27a}$$

shear viscosity

$$\rho(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \eta \triangle \mathbf{v} + (\zeta + \frac{\eta}{3}) \nabla(\nabla \cdot \mathbf{v}), \tag{27b}$$

bulk viscosity

viscous stress tensor

$$\partial_t(\epsilon + \frac{\rho v^2}{2}) + \nabla \cdot [(p + \epsilon + \frac{\rho v^2}{2})\mathbf{v} - \mathbf{v} \cdot \sigma'] = \mathbf{0}, \tag{27c}$$

internal energy

where ρ represents the (mass) density of the fluid, $\mathbf{v}$ denotes the fluid velocity (in vector field), p stands for the gas pressure, ϵ describes the internal energy according to the equation of state, σ' is the viscous stress tensor, η and ζ are the PDE parameters which represent the shear and bulk viscosity, respectively. We subsample the data for the 1D, 2D and 3D equations. The original resolution of 1D simulation has 1024 spatial points and 101 timesteps. For training purposes, we subsample across both spatial and temporal resolutions by a factor of 4 and 2 respectively yielding a trajectory of length 51 time steps and a spatial resolution of 256. To be consistent with other 1D PDE trajectory lengths, we retain only the first 41 timesteps and perform experiments on this truncated data. The original resolution of 2D simulation is 128×128 for each channel (i.e., density, velocity-x, velocity-y, and pressure) and has 21 timesteps. For training, we have subsampled the data only for the spatial dimension resulting in a resolution of 64×64. For 3D data, the original spatial resolution is $128 \times 128 \times 128$ which was subsampled to a resolution of $32 \times 32 \times 32$ for training. The temporal resolution is 21 timesteps as for the 2D case.

Table 9 summarizes the spatial and temporal discretization, mach number, and PDE parameter values of the 2D CNS dataset.

Field Type	Mach	η	ζ	Δt	Δx	Δy
Rand	{0.1, 1.0}	0.01	0.01	0.05	0.0078125	0.0078125
Rand	{0.1, 1.0}	0.1	0.1	0.05	0.0078125	0.0078125
Rand	{0.1, 1.0}	10^{-8}	10^{-8}	0.05	0.001953125	0.001953125

Table 9: Original dataset configuration for 2D Compressible Navier-Stokes equations.

However, since we perform subsampling on the spatial axes before training the neural net models, the effective values of Δx and Δy are slightly higher (i.e., coarser resolution).

D. Baseline Models

We consider the following strong baselines spanning five different families of models for solving PDEs using neural networks: Fourier Neural Operators, GNNs, UNets, Neural Fields, and Transformers. Our proposed model is indicated in pink color in Figure 9.

D.1. Fourier Neural Operator Baseline

Fourier Neural Operator (FNO) (Li et al., 2020a) is an implementation of a neural operator that maps from one function $a(x)$ to another function $a'(x)$ (Kovachki et al., 2021). Traditionally, neural networks model a mapping between two finite-dimensional Euclidean spaces which leads to the problem that they are fixed to a spatial and temporal resolution when used for solving PDEs. Neural operators overcome this limitation by learning an operator that is a mapping between infinite-dimensional function spaces (i.e., mapping between functions). FNO, an instantiation of a neural operator, is based on spectral convolution layers which implement an integral transformation of the input function. The integral transformation is implemented with discrete Fourier transforms on the spatial or spatial and temporal domain allowing an efficient and expressive architecture. We use a FNO model, that applies the integral transformation on the spatial domain and is trained in an autoregressive fashion for 500 epochs, as a baseline. Thus, the FNO learns a mapping between the function $a(\boldsymbol{x}) := u(t_n, \boldsymbol{x})$ representing the solution for timestep t_n and $a'(\boldsymbol{x}) := u(t_{n+1}, \boldsymbol{x})$ denoting the solution for a future timestep t_{n+1}.

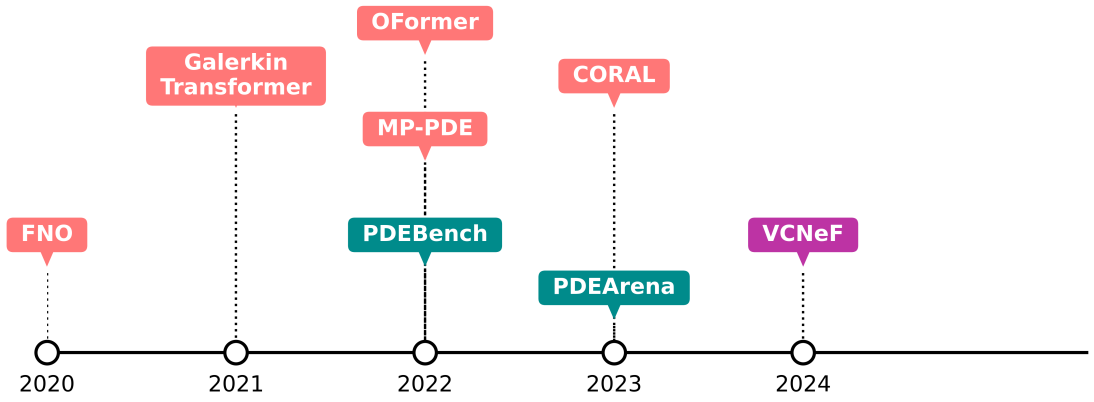


Figure 9: Timeline of Neural PDE Solvers and Benchmarks (PDEBench & PDEArena). Proposed model VCNeF.

cFNO. cFNO (Takamoto et al., 2023) is the adapted version of the FNO where the PDE parameters are added as an additional channel to condition the model on the PDE parameters.

D.2. Graph Neural Network Baseline

Several models exist that use Graph Neural Networks (GNNs) for solving PDEs (Brandstetter et al., 2022; Boussif et al., 2022)

Message Passing Neural PDE Solvers. MP-PDE (Brandstetter et al., 2022) follows the prevalent *Encode-Process-Decode* framework for simulating physical systems (Sanchez-Gonzalez et al., 2020). The MP-PDE model has an MLP as encoder, GNN as a processor, and a CNN as the decoder. Moreover, the model introduces several tricks such as pushforward, temporal bundling (time window with 5 timesteps), and random timesteps in the length of the trajectory as starting points during training for autoregressive PDE solving, while also considering the PDE parameter values as additional input, making it as a versatile choice for generalized neural PDE solving. Hence, we adopt it as a baseline. However, it has to be noted that we apply the model only to 1D PDEs. The configuration of our adaptation for 1D PDEs amounts to 614,929 model parameters which is comparable to the other baselines.

D.3. UNet Baselines

UNet-style models are increasingly becoming popular choices for weather forecasting and PDE solving due to their natural support for multi-scale data modeling (Takamoto et al., 2022; Rahman et al., 2023; Ovadia et al., 2023; Lippe et al., 2023), and have, thus, emerged as one of the competitive baseline models in SciML literature.

UNet-PDEArena. We use the modern UNet from PDEArena[3] (Gupta & Brandstetter, 2023). To match the number of parameters of the other models, we use an initial hidden dimension of 16 and two downsampling layers (3 for 2D). Following Lippe et al. (2023), we use the model to predict the residual instead of the next step directly and scale down the model output by a factor of 0.3. The training is performed autoregressively for 500 epochs using an initial learning rate of $3.e^{-3}$, which gets halved every 100 epochs.

D.4. Implicit Neural Representation Baseline

CORAL: Coordinate-based Model for Operator Learning. Considering that CORAL (Serrano et al., 2023), to the best of our knowledge, is the current state-of-the-art INR-based method for solving PDEs, we benchmark our proposed VCNeF model against it on 1D Advection and Burgers' as well as on the challenging 1D and 2D compressible Navier-Stokes PDEs (Takamoto et al., 2022). The CORAL model is trained purely in a data-driven manner and involves two stages: (i) INR

[3] https://microsoft.github.io/pdearena/

training, and (ii) Dynamics modeling training. Due to the sequential nature of this two phase training, first we train the INR model and dynamics model is trained after the completion of INR model training. CORAL authors conducted experiments on a small dataset of 256 training and 16 test samples. We, on the other hand, conduct experiments on PDEBench which consists of 9000 train and 1000 test samples. Hence, we train the CORAL baseline model for 1000 epochs for INR training and 500 epochs for dynamics modeling optimization, unlike the original authors' suggested setting of 10000 epochs of optimization for both INR and dynamics modeling training.

As in the case of other baseline models, we train and test the CORAL baseline on the subsampled data yielding a spatial and temporal resolution of 256 and 41 respectively. We report results on 1D Advection, Burgers', and CNS. The training of 2D CNS resulted in very high errors in the INR training phase and the loss diverged to NaN values in dynamics modeling training. We encode both the single and multiple channel inputs of PDEs in a single latent space of dimension 256 with the aim to keep the model simple and match the number of parameters to other baseline models. For other hyperparameter values such as the learning rate, NODE depth and width, we use the default values suggested by Serrano et al. (2023).

D.5. Transformer Baselines

Transformer-based models are increasingly used to solve PDEs (Cao, 2021; Li et al., 2023a;b; Hao et al., 2023). Hence, we use Galerkin Transformer and Operator Transformer as state-of-the-art transformer-based models.

Galerkin Transformer. Cao (2021) introduces the novel application of self-attention for learning a neural operator. The author provides an alternative way to interpret the matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ by interpreting them column-wise as the evaluation of learned basis functions instead of row-wise as the latent representation of the tokens. This new interpretation allows the author to improve the effectiveness of the attention mechanism by linearizing it, yielding Fourier and Galerkin-type attention. The author employs the proposed attention mechanisms in a transformer-based neural operator for solving PDEs. We choose Galerkin Transformer as a baseline because it is transformer-based and uses self-attention on the spatial domain of the PDE. The baseline model is trained for 500 epochs in an autoregressive fashion using the hyperparameters suggested by Cao (2021).

Operator Transformer. OFormer (Li et al., 2023a) is a transformer-based neural operator which is based on the attention types proposed in Galerkin and Fourier Transformer (Cao, 2021). Existing approaches such as FNO and Galerkin or Fourier Transformer are restricted in having the same grid for the input and output. Consequently, it is not possible to query the model (i.e., output function) on arbitrary spatial points that are different or partially disjoint from the input points. OFormer solves this problem by adding cross-attention to the model to allow querying for arbitrary spatial points. In addition, the authors suggest further improvements to the Galerkin or Fourier Transformer and name the resulting model Operator Transformer (OFormer). We train the OFormer model in an autoregressive fashion with the curriculum learning strategy of Takamoto et al. (2023) for 500 epochs.

cOFormer. Inspired by cFNO (Takamoto et al., 2023) we adapt OFormer to take the PDE parameter as an additional input. The PDE parameter values are appended to the input as an additional channel to condition the model on the PDE parameter.

E. Additional Experimental Details

E.1. Used PDE Parameters

Table 10 shows the combinations of PDE parameter values used in our experiments for the multiple parameters setting. In this case, we train the models on a set of PDE parameter values (**seen**) and test it on a different set of PDE parameter values (**unseen**) with the aim to test the model's generalization capabilities on this aspect.

PDE	Training Set Parameters (seen)	Test Set Parameters (unseen)
1D Burgers	$\nu = (0.002, 0.004, 0.02, 0.04, 0.2, 0.4, 2.0)$	$\nu = (0.001, 0.01, 0.1, 1.0, 4.0)$
1D Advection	$\beta = (0.2, 0.4, 0.7, 2.0, 4.0)$	$\beta = (0.1, 1.0, 7.0)$
1D CNS	$\eta = \zeta = (10^{-8}, 0.001, 0.004, 0.01, 0.04, 0.1)$	$\eta = \zeta = (0.007, 0.07)$

Table 10: Exemplary set of PDE parameters used in our experiments with multiple PDE parameters.

E.2. Loss Function

We use the Mean Squared Error (MSE) loss function as optimization criterion for FNO, UNet, Galerkin Transformer, OFormer, and the proposed VCNeF. Let $\boldsymbol{Y} \in \mathbb{R}^{N_b \times N_t \times s \times c}$ be a batch of ground truth trajectories and $\hat{\boldsymbol{Y}} \in \mathbb{R}^{N_b \times N_t \times s \times c}$ the corresponding batch of model's predictions where N_b denotes the batch size, N_t is the length of the trajectories, $s = s_x \cdot s_y \cdot \ldots$ the spatial points per timestep, and c the number of channels of the PDE. Then, the MSE loss function is defined as

$$\text{MSE}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = \frac{1}{N_b \cdot N_t \cdot s \cdot c} \sum_{b=1}^{N_b} \sum_{t=1}^{N_t} \sum_{i=1}^{s} \sum_{j=1}^{c} (\hat{\boldsymbol{Y}}_{b,t,i,j} - \boldsymbol{Y}_{b,t,i,j})^2 \tag{28}$$

Whereas MP-PDE and CORAL are trained with the loss functions suggested by the authors.

E.3. Model's Hyperparameters

Tables 11, 12, 13, 14 list the used hyperparameters for the baselines and our proposed VCNeF.

PDE	Model	Epochs	Batch size	Fourier width	# Fourier Modes	# Layers	Learning rate	LR Scheduler	# Parameters	Curriculum learning
1D Burgers	FNO	500	64	64	16	4	1.e-4	Step Scheduler	549,569	✗
1D Advection	FNO	500	64	64	16	4	1.e-4	Step Scheduler	549,569	✗
1D CNS	FNO	500	64	64	16	4	6.e-5	Step Scheduler	549,955	✗
2D CNS	FNO	500	64	32	12	4	3.e-4	Step Scheduler	9,453,716	✗
3D CNS	FNO	1000	4	20	12	4	3.e-4	Step Scheduler	22,123,753	✗

Table 11: Hyperparameters for the FNO used in the single PDE parameter experiments. The Step Scheduler was configured with a step size of 100 and gamma of 0.5.

PDE	Model	Epochs	Batch size	Embedding size	# Layers	Time window	Learning rate	# Parameters	Curriculum learning
1D Burgers	MP-PDE	20	64	128	6	5	1.e-4	614,929	✗
1D Advection	MP-PDE	20	64	128	6	5	1.e-4	614,929	✗

Table 12: Hyperparameters for the MP-PDE used in the single PDE parameter experiments.

PDE	Model	Epochs	Batch size	Embedding size	# Heads	# Layers	Learning rate	LR Scheduler	# Parameters	Curriculum learning
1D Burgers	OFormer	500	64	96	1	4+3	6.e-5	One Cycle Scheduler	660,814	✓
	Galerkin Transformer	500	64	96	1	4+2	1.e-5	One Cycle Scheduler	530,305	✗
	VCNeF	500	32	96	8	3+3	3.e-4	One Cycle Scheduler	793,825	✗
1D Advection	OFormer	500	64	96	1	4+3	6.e-5	One Cycle Scheduler	660,814	✓
	Galerkin Transformer	500	64	96	1	4+2	1.e-5	One Cycle Scheduler	530,305	✗
	VCNeF	500	64	96	8	3+3	6.e-4	One Cycle Scheduler	793,825	✗
1D CNS	OFormer	500	64	96	1	4+3	6.e-5	One Cycle Scheduler	662,733	✓
	Galerkin Transformer	500	64	96	1	4+2	1.e-5	One Cycle Scheduler	530,595	✗
	VCNeF	500	64	96	8	3+3	4.e-4	One Cycle Scheduler	794,307	✗
2D CNS	Galerkin Transformer	500	64	384	1	6+2	8.e-5	One Cycle Scheduler	8,053,091	✗
	VCNeF	1000	64	256	8	1+6	3.e-4	One Cycle Scheduler	11,779,436	✗
3D CNS	VCNeF	1000	4	256	8	1+6	3.e-4	One Cycle Scheduler	27,335,041	✗

Table 13: Hyperparameters for the transformer-based models used in the single PDE parameter experiments. The One Cycle Scheduler was configured to reach the maximum learning rate at 0.2, start division factor 1.e-3 and final division factor 1.e-4.

PDE	Model	Epochs	Batch size	# Downsample	# Upsample	# Layers	Learning rate	LR Scheduler	# Parameters	Curriculum learning
1D Burgers	UNet	500	256	64	16	4	1.e-4	Step Scheduler	557,137	✗
1D Advection	UNet	500	256	64	16	4	1.e-4	Step Scheduler	557,137	✗
1D CNS	UNet	500	256	-	16	4	6.e-5	Step Scheduler	562,579	✗
2D CNS	UNet	500	256	246	12	4	3.e-4	Step Scheduler	9,187,284	✗

Table 14: Hyperparameters of UNets from PDEArena for single PDE parameter experiments. The Step Scheduler was configured with a step size of 80 and gamma of 0.5.

E.4. Evaluation Metrics

We use the normalized RMSE (nRSME) and boundary RMSE (bRMSE) from PDEBench (Takamoto et al., 2022) as metrics to evaluate the models.

Normalized RMSE (nRMSE). The normalized RMSE ensures the independence of the different scales of field variables. The channels of PDEs with multiple channels are often on different scales (e.g., one channel consists of values with small magnitudes while another channel consists of values with large magnitudes). Additionally, the scale of a single channel usually changes when the time-dependent PDE evolves in time (e.g., large magnitudes at the beginning of the trajectory decaying to small magnitudes at the end). nRMSE is independent of these scaling effects and provides a good metric for the global and local performance of the ML model. Let $\boldsymbol{Y} \in \mathbb{R}^{N_t \times s \times c}$ be the ground truth trajectory and $\hat{\boldsymbol{Y}} \in \mathbb{R}^{N_t \times s \times c}$ the model's prediction where N_t denotes the length, $s = s_x \cdot s_y \cdot \ldots$ the spatial points per timestep, and c the number of channels of the PDE. Then, the per-sample nRMSE is defined as

$$\begin{aligned} \text{relativeError}(t, c') &= \frac{\left\| \boldsymbol{Y}_{t,\cdot,c'} - \hat{\boldsymbol{Y}}_{t,\cdot,c'} \right\|_2}{\| \boldsymbol{Y}_{t,\cdot,c'} \|_2} \qquad \in \mathbb{R} \\ \text{nRMSE} &= \frac{1}{N_t \cdot c} \sum_{t=1}^{N_t} \sum_{i=1}^{c} \text{relativeError}(t, i) \qquad \in \mathbb{R} \end{aligned} \tag{29}$$

Boundary RMSE (bRMSE). The RMSE on the boundaries of the spatial domain quantifies whether the boundary condition can be learned or not. Let $\boldsymbol{Y} \in \mathbb{R}^{N_t \times s_x \times c}$ be the ground truth trajectory of a 1D PDE and $\hat{\boldsymbol{Y}} \in \mathbb{R}^{N_t \times s_x \times c}$ the model's prediction where N_t denotes the length, s_x denotes the number of points for the x-axis, and c the number of channels of the PDE under consideration. Then, the per-sample bRMSE is defined as

$$\begin{aligned} \text{boundaryError}(t, c') &= \sqrt{\frac{(\boldsymbol{Y}_{t,1,c'} - \hat{\boldsymbol{Y}}_{t,1,c'})^2 + (\boldsymbol{Y}_{t,s,c'} - \hat{\boldsymbol{Y}}_{t,s,c'})^2}{2}} \qquad \in \mathbb{R} \\ \text{bRMSE} &= \frac{1}{N_t \cdot c} \sum_{t=1}^{N_t} \sum_{i=1}^{c} \text{boundaryError}(t, i) \qquad \in \mathbb{R} \end{aligned} \tag{30}$$

E.5. Randomized Starting Points Training

Algorithm 1 describes the training with randomized starting points for the VCNeF-R. During training, the model is conditioned on the initial value $u(0, \boldsymbol{x})$ as well as on randomly sampled $u(t, \boldsymbol{x})$ with $t \sim \mathcal{U}\{0, T\}$ along the trajectory as starting points.

Algorithm 1 Randomized Starting Points as Initial Conditions

```
Input: training data set 𝒟, training trajectories length N_t, number of epochs Epochs
for epoch = 1 to Epochs do
  starting_points = [0] ∪ random_shuffle([1, ···, N_t − 1])[:10]
  for starting_point in starting_points do
    for (x, y) in 𝒟 do
      model.train(x[starting_point], y[starting_point:])
    end for
  end for
end for
```

F. Additional Experimental Results

This section contains additional results of the experiments. We train all models on two different initializations and provide the mean and standard deviations of the runs. Similar to the experiments section in the main paper, we structure the results to answer the following five research questions.

Q1: How effective are VCNeFs compared to the state-of-the-art (SOTA) methods when trained and tested for the same PDE parameter value?

Q2: How well can VCNeFs generalize to PDE parameter values not seen during training?

Q3: How well can VCNeFs do spatial and temporal zero-shot super-resolution?

Q4: Does training on initial conditions sampled from training trajectories improve the accuracy?

Q5: Does the vectorization provide a speed-up, and what is the model's scaling behavior?

F.1. (Q1): Comparison to state-of-the-art baselines

F.1.1. Detailed Metrics

The Tables 15, 16, 17, 18, and 19 show the metrics with standard deviations for the chosen PDEs and models.

Model	nRMSE (↓)	bRMSE (↓)
FNO	$0.0987^{\pm 0.0004}$	$0.0225^{\pm 0.0006}$
MP-PDE	$0.3046^{\pm 0.0004}$	$0.0725^{\pm 0.0014}$
UNet	$\mathbf{0.0566}^{\pm 0.0004}$	$0.0259^{\pm 0.0019}$
CORAL	$0.2221^{\pm 0.0108}$	$0.0515^{\pm 0.0001}$
Galerkin	$0.1651^{\pm 0.0044}$	$0.0366^{\pm 0.0012}$
OFormer	$0.1035^{\pm 0.0059}$	$0.0215^{\pm 0.0009}$
VCNeF	$0.0824^{\pm 0.0004}$	$0.0228^{\pm 0.0003}$
VCNeF-R	$0.0784^{\pm 0.0001}$	$\mathbf{0.0179}^{\pm 0.0001}$

Table 15: Normalized RMSE (nRMSE) and RMSE at the boundaries (bRMSE) of baselines and proposed model for the 1D Burgers' equation with $\nu = 0.001$.

Model	nRMSE (↓)	bRMSE (↓)
FNO	$0.0190^{\pm 0.0003}$	$0.0239^{\pm 0.0002}$
MP-PDE	$0.0195^{\pm 0.0011}$	$0.0283^{\pm 0.0022}$
UNet	$\mathbf{0.0079}^{\pm 0.0024}$	$0.0129^{\pm 0.0043}$
CORAL	$0.0198^{\pm 0.0031}$	$0.0127^{\pm 0.0014}$
Galerkin	$0.0621^{\pm 0.0024}$	$0.0349^{\pm 0.0011}$
OFormer	$0.0118^{\pm 0.0012}$	$0.0073^{\pm 0.0008}$
VCNeF	$0.0165^{\pm 0.0007}$	$0.0088^{\pm 0.0003}$
VCNeF-R	$0.0113^{\pm 0.0003}$	$\mathbf{0.0040}^{\pm 0.0005}$

Table 16: Normalized RMSE (nRMSE) and RMSE at the boundaries (bRMSE) of baselines and proposed model for the 1D Advection equation with $\beta = 0.1$.

Model	nRMSE (↓)	bRMSE (↓)
FNO	$0.5722^{\pm 0.0244}$	$1.9797^{\pm 0.0029}$
UNet	$0.2270^{\pm 0.0133}$	$\mathbf{1.0399}^{\pm 0.0863}$
CORAL	$0.5993^{\pm 0.1014}$	$1.5908^{\pm 0.1341}$
Galerkin	$0.7019^{\pm 0.0002}$	$3.0143^{\pm 0.0112}$
OFormer	$0.4415^{\pm 0.0115}$	$2.0478^{\pm 0.0581}$
VCNeF	$0.2943^{\pm 0.0034}$	$1.3496^{\pm 0.0254}$
VCNeF-R	$\mathbf{0.2029}^{\pm 0.0227}$	$1.1366^{\pm 0.0589}$

Table 17: Normalized RMSE (nRMSE) and RMSE at the boundaries (bRMSE) of baselines and proposed model for the 1D CNS equation with $\eta = \zeta = 0.007$.

Model	nRMSE (↓)	bRMSE (↓)
FNO	$0.5625^{\pm 0.0015}$	$0.2332^{\pm 0.0001}$
UNet	$1.4240^{\pm 0.5018}$	$0.3703^{\pm 0.0432}$
Galerkin	$0.6702^{\pm 0.0036}$	$0.8219^{\pm 0.0043}$
VCNeF	$\mathbf{0.1994}^{\pm 0.0086}$	$\mathbf{0.0904}^{\pm 0.0036}$

Table 18: Normalized RMSE (nRMSE) and RMSE at the boundaries (bRMSE) of baselines and proposed model for the 2D CNS equation with $\eta = \zeta = 0.01$.

Model	nRMSE (↓)	bRMSE (↓)
FNO	$0.8138^{\pm 0.0007}$	$6.0407^{\pm 0.0493}$
VCNeF	$\mathbf{0.7086}^{\pm 0.0005}$	$\mathbf{4.8922}^{\pm 0.0077}$

Table 19: Normalized RMSE (nRMSE) and RMSE at the boundaries (bRMSE) of baselines and proposed model for the 3D CNS equation with $\eta = \zeta = 10^{-8}$.

F.1.2. Error Visualization

We visualize the error along the spatial and temporal domains for 1D PDEs using a heatmap. For a given timestep t, let $\boldsymbol{y_t} \in \mathbb{R}^s$ be the ground truth and $\hat{\boldsymbol{y}}_{\boldsymbol{t}} \in \mathbb{R}^s$ the model's prediction. Then, we calculate the point-wise error for the 2D error visualization as

$$e(t) = \sqrt{\frac{(\boldsymbol{y_t} - \hat{\boldsymbol{y}}_{\boldsymbol{t}})^2}{\boldsymbol{y_t}^2}} = \frac{|\boldsymbol{y_t} - \hat{\boldsymbol{y}}_{\boldsymbol{t}}|}{|\boldsymbol{y_t}|} \in \mathbb{R}^s \tag{31}$$

Each of the operations in Equation (31) is applied in a pointwise manner to the elements of $\boldsymbol{y_t}$ and $\hat{\boldsymbol{y}}_{\boldsymbol{t}}$. We calculate the mean value over all test samples to get the final error heatmap. The error heatmap for 1D Burgers' equation in Figure 10 shows that all models, except for UNet and VCNeF, exhibit a very high error at the first and second timesteps (white area). Figures 11 and 12 show that FNO and Galerkin Transformer have a very high error on a few spatial coordinates for 1D Advection and 1D CNS.

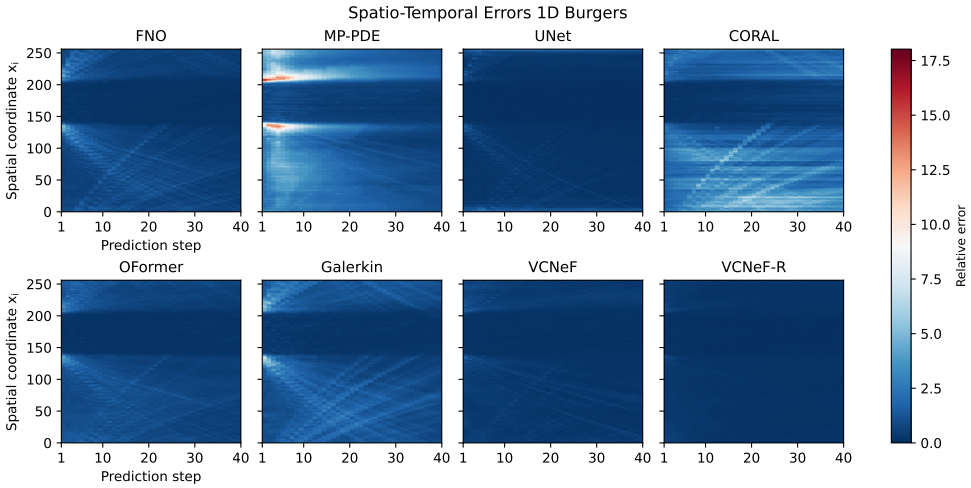


Figure 10: Error heatmap for 1D Burger's equation with $\nu = 0.001$. Models are trained and tested on spatial resolution $s = 256$ and temporal resolution $N_t = 41$. t_0 (not depicted above) is the initial condition.

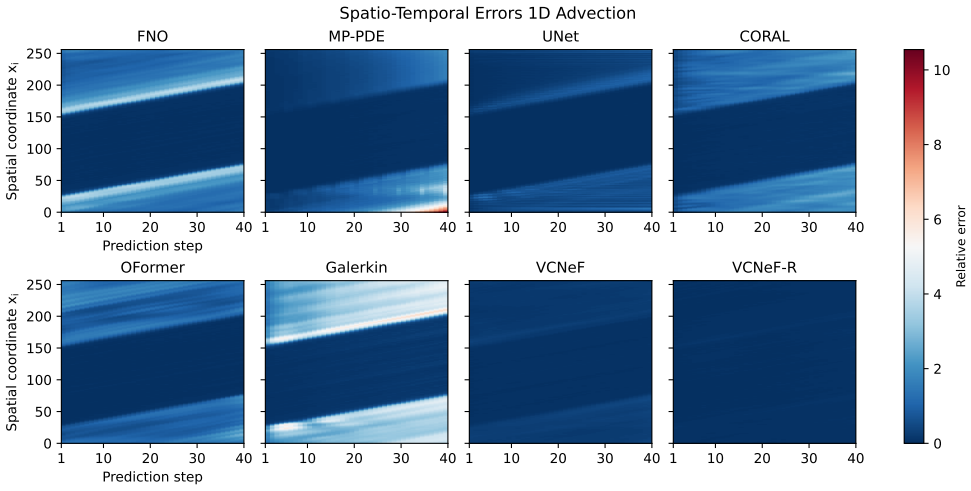


Figure 11: Error heatmap for 1D Advection with $\beta = 0.1$. Models are trained and evaluated on spatial resolution $s = 256$ and temporal resolution $N_t = 41$. t_0 (not shown above) is the initial condition.

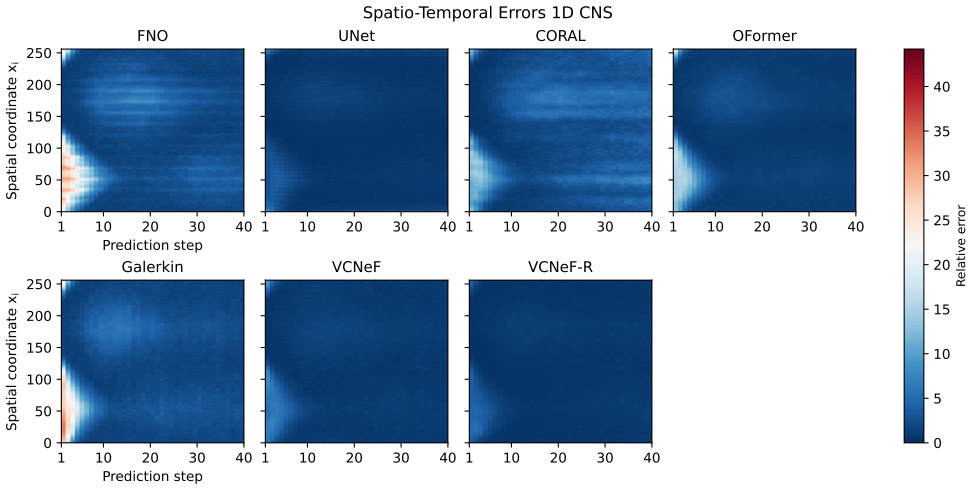


Figure 12: Error heatmap for 1D CNS with $\eta = \zeta = 0.007$. Models are trained and evaluated on spatial resolution $s = 256$ and temporal resolution $N_t = 41$. t_0 (not depicted above) is the initial condition.

F.1.3. Temporal Error

The following section shows the temporal error of the baselines and proposed model. Figure 13 shows the temporal error of the models for 1D Burgers, 1D Advection, and 1D CNS.

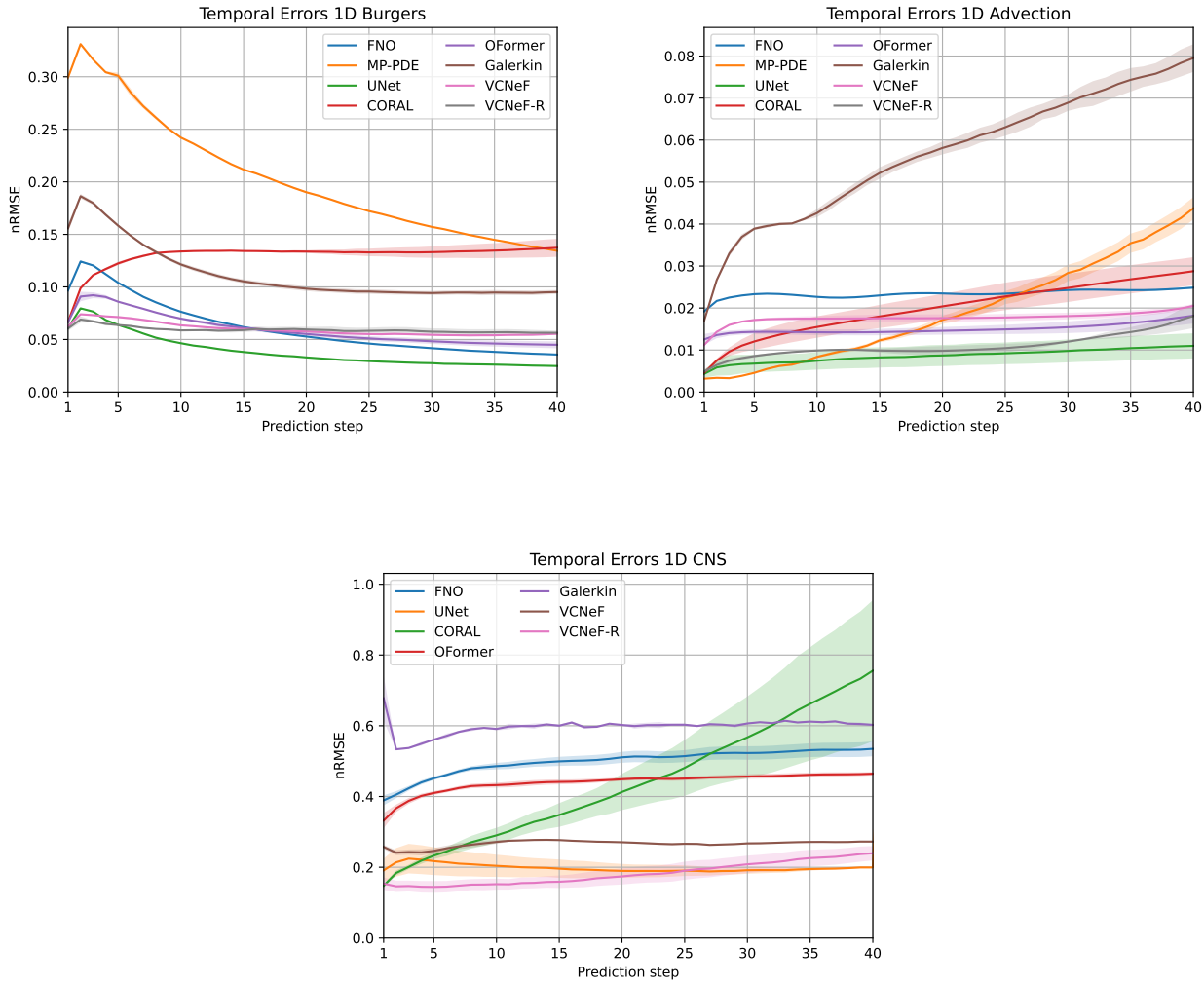


Figure 13: Temporal error of the models considered. The confidence band shows the standard deviation. t_0 (not visualized above) is the initial condition.

F.2. (Q2): Generalization to Unseen PDE Parameter Values

We test VCNeF's generalization capabilities to unseen PDE parameter values by training it on a set of PDE parameter values and testing it on a different set of unseen PDE parameter values. We use cFNO (Takamoto et al., 2023) and cOFormer as the state-of-the-art baselines. Both models have been adapted to encode the PDE parameter as an additional input channel. Figures 14, 15, 16 show the error distribution over the corresponding PDE parameter values and test sets.

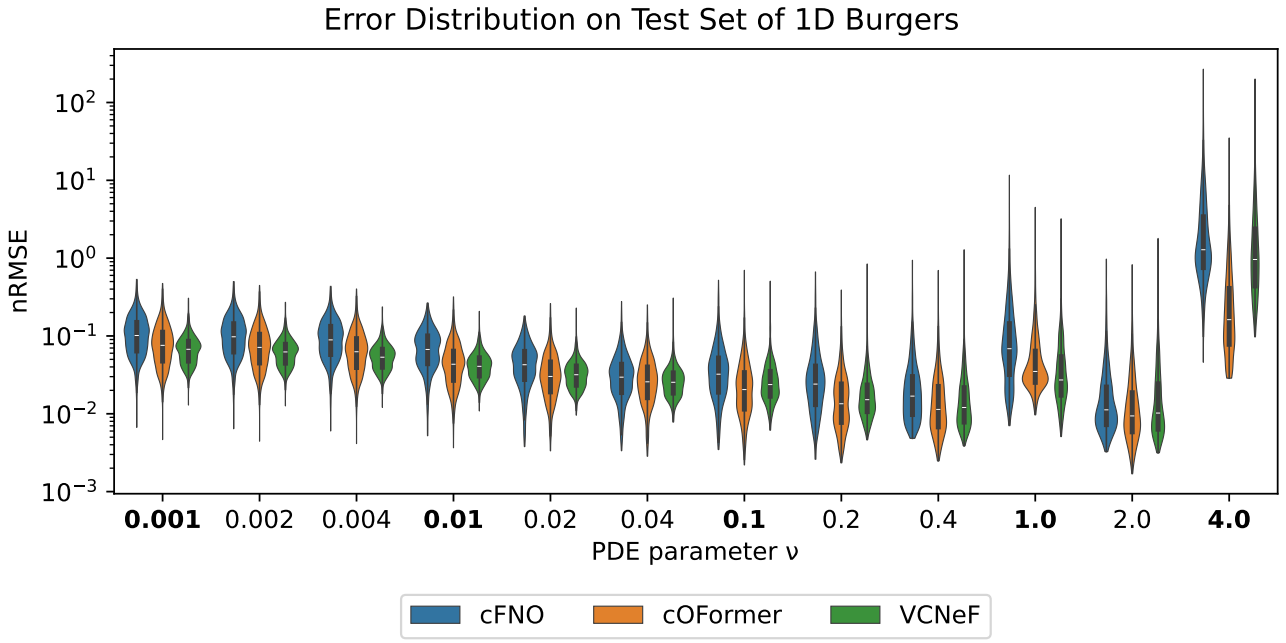


Figure 14: Error distribution of samples in the test set of 1D Burgers. Boldfaced are the unseen PDE parameter values.

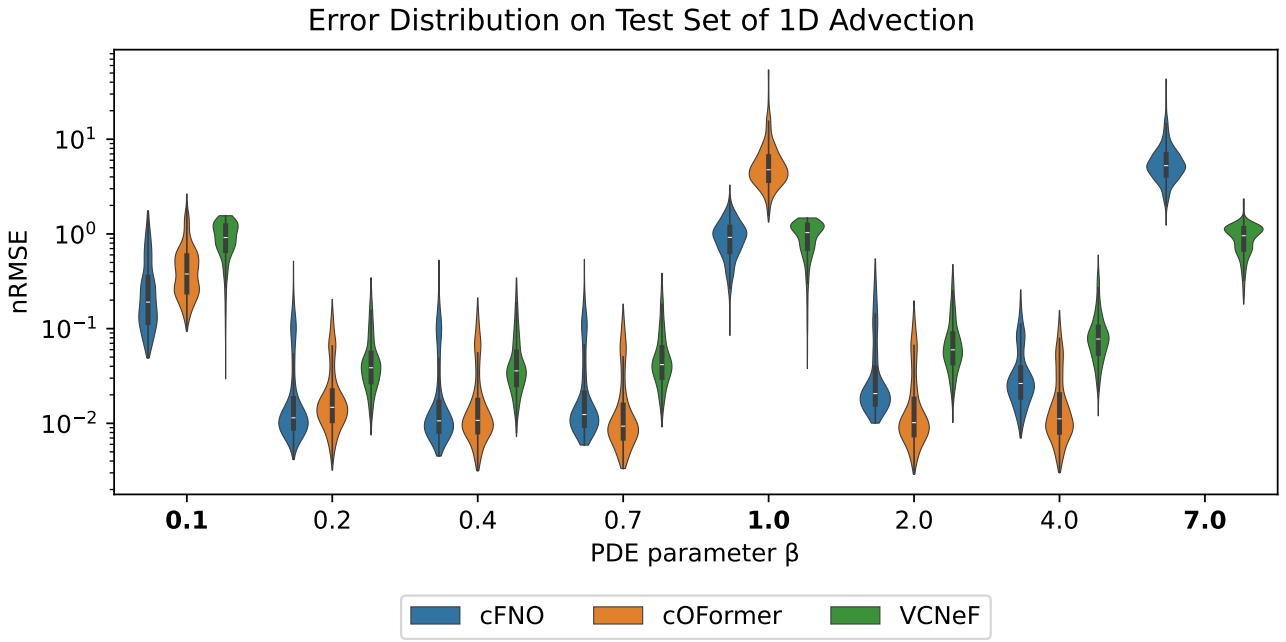


Figure 15: Error distribution of samples in the test set of 1D Advection. Boldfaced are the unseen PDE parameter values. Values for cOFormer and $\beta = 7.0$ are missing since the model produced NaN at inference time.

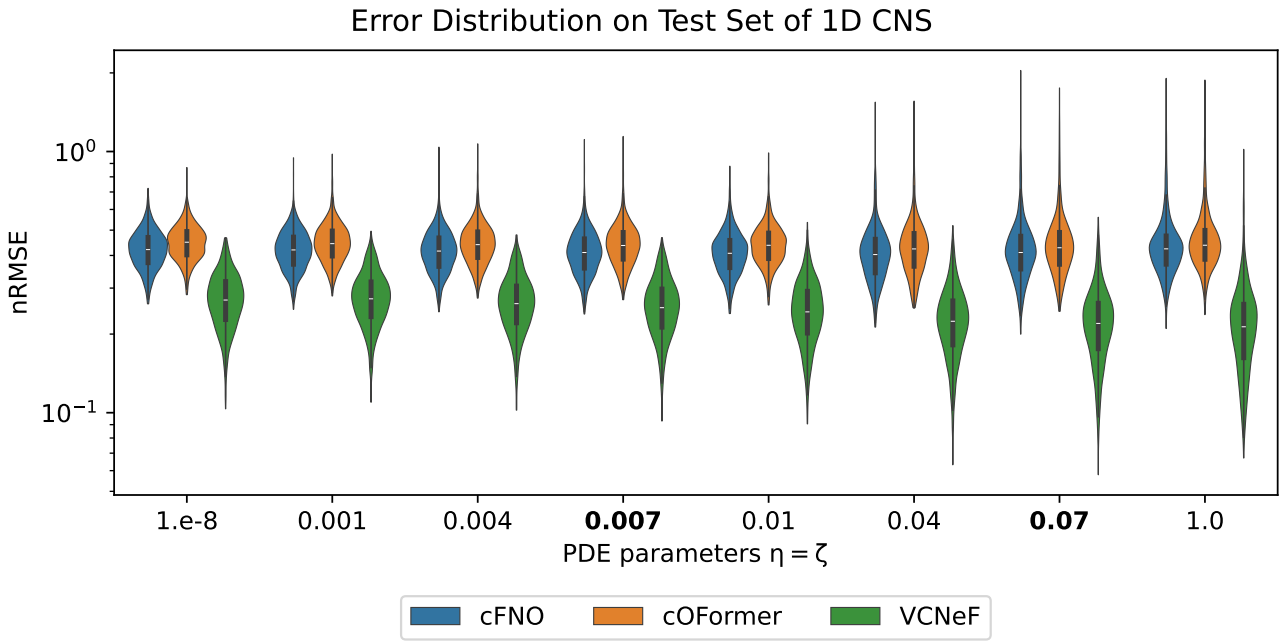


Figure 16: Error distribution of samples in the test set of 1D CNS. Boldfaced are the unseen PDE parameter values.

F.3. (Q5): Inference Time and Memory Consumption

In traditional numerical solvers, the simulation time of trajectories of a given PDE is influenced by several factors, such as the value of the PDE parameter, efficiency of software implementation, the type and order of the numerical algorithm, discretization mesh, etc. On the contrary, the inference (simulation) time of ML models is agnostic to factors such as the PDE parameter value or order of numerical algorithm, which is one of the huge advantages of Neural PDE surrogates. Table 20 demonstrates that the inference times of the proposed model scale better when compared to other transformer-based baselines. However, the speed-up results in a higher memory consumption. The model can also be used to do inference in a sequential fashion, which reduces the memory consumption but increases the inference time. Nevertheless, it is still faster than OFormer, and the memory requirement remains the same even for extended rollout durations.

Prediction steps	Model	Inference time [ms]		GPU memory consumption [MiB]
40	FNO	917.77	± 2.51	716
	Galerkin	2415.99	± 54.56	632
	OFormer	6025.75	± 12.75	990
	VCNeF	2244.04	± 6.65	4724
	VCNeF sequential	4853.17	± 75.29	644
80	FNO	1912.19	± 56.03	716
	Galerkin	4940.80	± 89.44	632
	OFormer	12081.98	± 19.39	990
	VCNeF	4422.65	± 4.11	9284
	VCNeF sequential	9701.80	± 84.48	644
120	FNO	2808.04	± 82.22	716
	Galerkin	7908.18	± 96.52	644
	OFormer	17965.47	± 14.19	988
	VCNeF	6606.41	± 3.00	13638
	VCNeF sequential	14577.00	± 112.83	644
160	FNO	3733.10	± 62.94	716
	Galerkin	10295.78	± 116.50	644
	OFormer	24108.24	± 6.45	990
	VCNeF	6084.04	± 9.37	18871
	VCNeF sequential	19449.80	± 113.73	644
200	FNO	4614.21	± 97.52	718
	Galerkin	13151.47	± 93.95	644
	OFormer	29986.81	± 6.35	990
	VCNeF	7584.48	± 1.86	22328
	VCNeF sequential	24252.38	± 101.41	644
240	FNO	5572.07	± 109.23	716
	Galerkin	15600.60	± 262.51	644
	OFormer	35900.51	± 6.71	988
	VCNeF	8935.28	± 7.08	26662
	VCNeF sequential	29063.89	± 79.58	668

Table 20: Inference times and GPU memory consumptions of different models trained and evaluated on the 1D Burgers' equation with a spatial resolution of 256, predicting different numbers of timesteps in future. "VCNeF sequential" means a VCNeF model that computes the solutions for all timesteps sequentially.

G. Qualitative Results

Here we provide a comparison of visualizations of the predictions vs ground truth for 1D Advection, Burgers, and 2D Compressible Navier-Stokes PDEs. The 2D CNS dataset has four channels, namely density velocity-x and velocity-y, and pressure, and we visualize the predictions of our VCNeF model with the ground truth data.

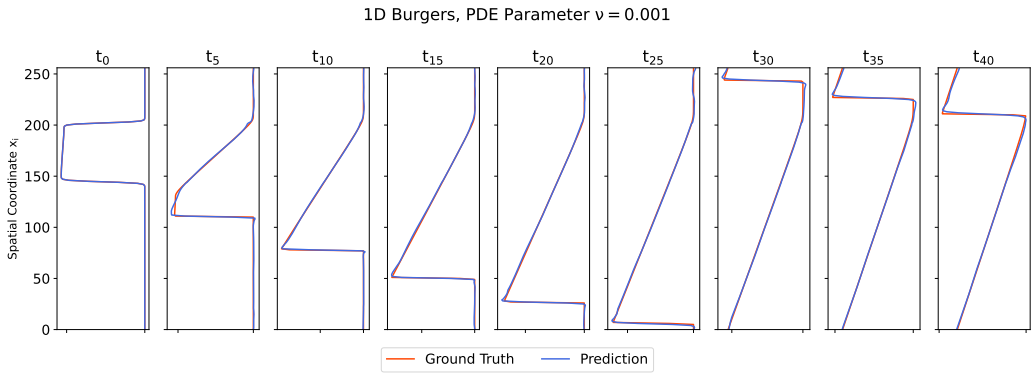


Figure 17: Example prediction's of VCNeF for 1D Burgers with $N_t = 41$.

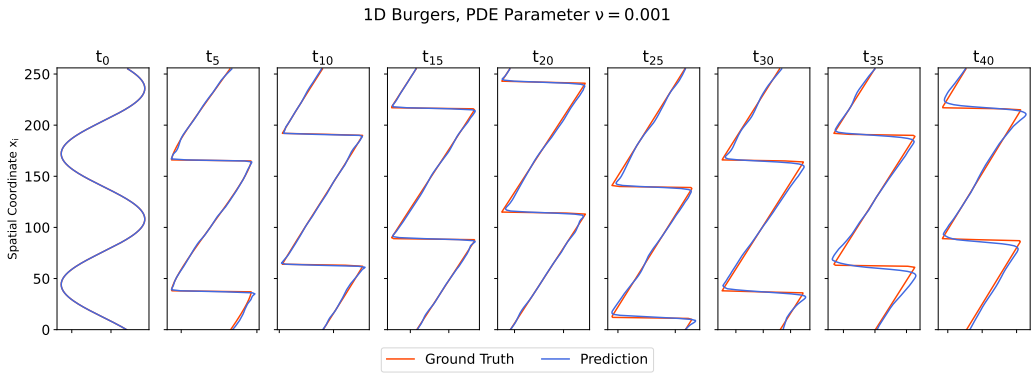


Figure 18: Example prediction of VCNeF for 1D Burgers with $N_t = 41$.

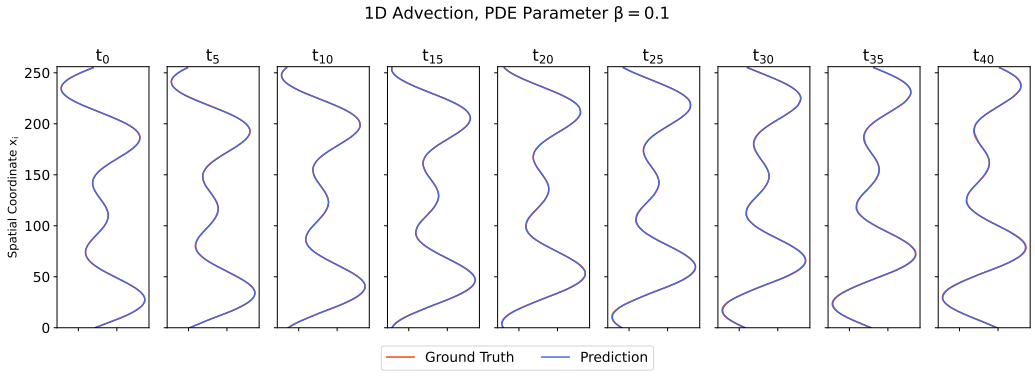


Figure 19: Example prediction of VCNeF for 1D Advection with $N_t = 41$.

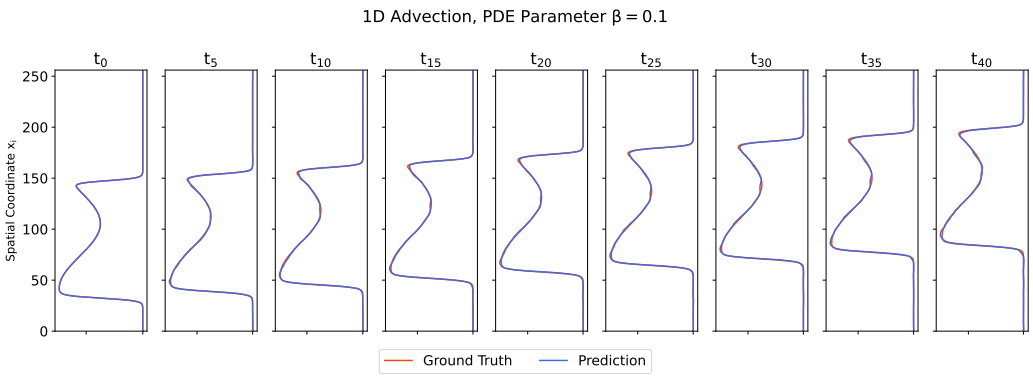


Figure 20: Example prediction of VCNeF for 1D Advection with $N_t = 41$.

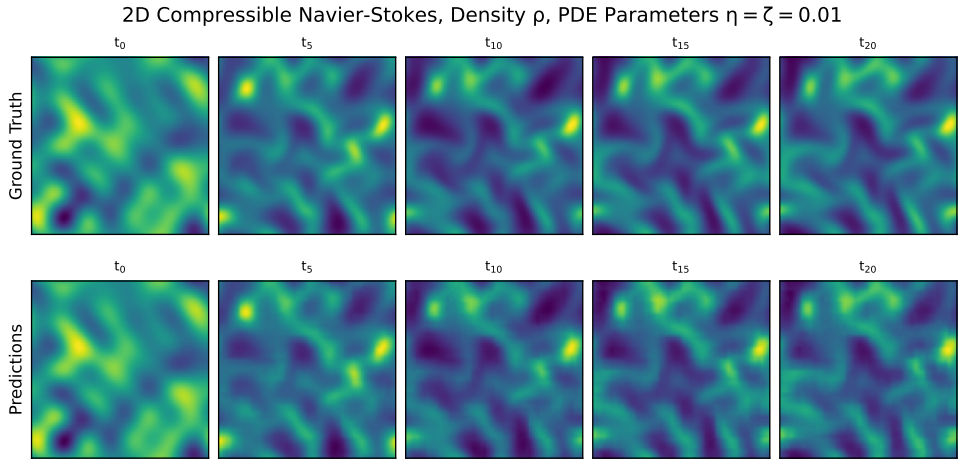


Figure 21: Example prediction of VCNeF for the density channel of 2D compressible Navier-Stokes with $N_t = 21$.

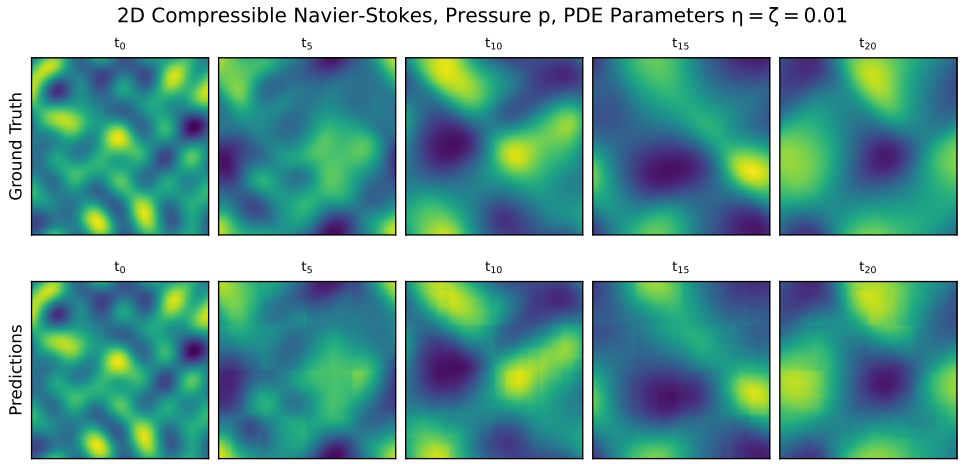


Figure 22: Example prediction of VCNeF for the pressure channel of 2D compressible Navier-Stokes with $N_t = 21$.

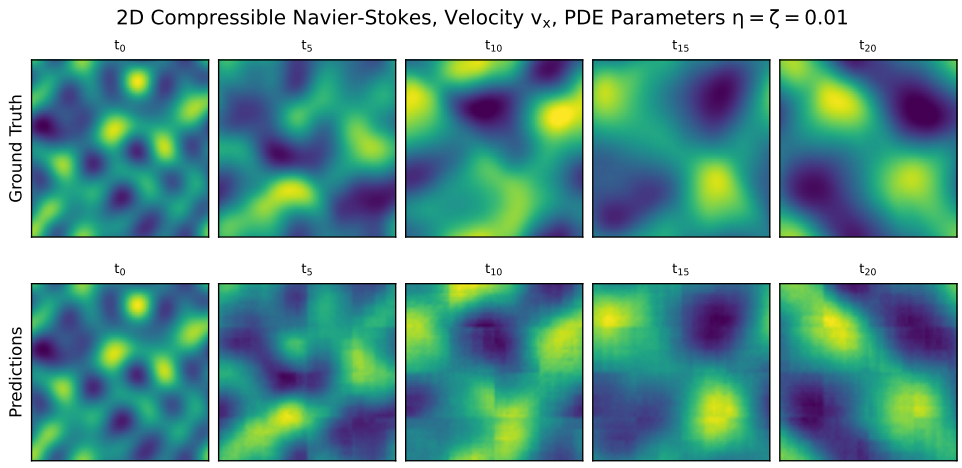


Figure 23: Example prediction of VCNeF for the velocity x-axis channel of 2D compressible Navier-Stokes with $N_t = 21$.