Institute for Artificial Intelligence

University of Stuttgart
Universitätsstraße 32
D–70569 Stuttgart

Bachelorarbeit

Linear Transformers for Solving Parametric Partial Differential Equations

Jan Hagnberger

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Mathias Niepert
Supervisor:	Marimuthu Kalimuthu, M.Sc.
Commenced:	September 1, 2023
Completed:	March 1, 2024

Abstract

The simulation of physical phenomena relies on solving Partial Differential Equations (PDEs), and Machine Learning models have increasingly addressed this task in recent years. PDEs often involve parameters influencing their evolution, prompting the development of models that consider these parameters as additional input. These parameter-conditioned models aim to generalize across different PDE parameters, replacing the need for multiple models trained on specific ones.

Transformer models have been achieving great success in Natural Language Processing (NLP), Speech Processing, and even in domains such as Computer Vision. Due to their ability to effectively model long-range dependencies in sequential data, their field of application is steadily increasing. Calculating attention via Scaled Dot-Product Attention in Vanilla Transformers is computationally expensive and scales quadratically with the input length. This leads to a bottleneck for very long sequences. To address this challenge, Linear Transformers have been introduced, substituting the Scaled Dot-Product Attention to achieve linear time and space complexity. Consequently, Linear Transformers have shown promising potential for processing very long sequences efficiently.

We investigate two approaches of utilizing Linear Transformers for solving PDEs and their associated problems. Moreover, we conduct a comprehensive comparison between our proposed transformer-based models and state-of-the-art models for solving parametric PDEs. The evaluation criteria include accuracy for short and long rollouts, memory consumption, and inference times. The results demonstrate that our proposed models perform competitively with the current state-of-the-art models, providing an efficient solution for PDE solving.

Kurzfassung

Die Simulation physikalischer Phänomene beruht auf dem Lösen von partiellen Differentialgleichungen (engl. Partial Differential Equations, kurz PDEs). In den letzten Jahren wurden zunehmend Machine Learning Modelle entwickelt, die partielle Differentialgleichungen lösen. PDEs beinhalten oft Parameter, die ihre Lösungen beeinflussen, was die Entwicklung von Machine Learning Modellen, die diese Parameter als zusätzlichen Input berücksichtigen, vorangetrieben hat. Diese parameterkonditionierten Modelle zielen darauf ab, über verschiedene PDE-Parameter hinweg zu generalisieren, sodass nicht mehr mehrere Modelle benötigt werden, die für bestimmte Parameter trainiert wurden.

Transformer-Modelle haben sich in der Verarbeitung natürlicher Sprache (engl. Natural Language Processing, kurz NLP), der Verarbeitung von Audiosequenzen und sogar in Bereichen wie der Computer Vision als sehr erfolgreich erwiesen. Aufgrund ihrer Fähigkeit, Abhängigkeiten über große Distanzen hinweg in sequenziellen Daten effektiv zu modellieren, wird das Anwendungsgebiet von Transformer zunehmend größer. Die Berechnung von Attention mittels Scaled Dot-Product Attention in Vanilla Transformer ist rechenintensiv und skaliert quadratisch mit der Eingabelänge. Dies führt bei sehr langen Sequenzen zu einem Engpass. Um dieses Problem zu lösen, wurden Linear Transformer eingeführt, die die Berechnung von Scaled Dot-Product Attention ersetzen, um eine lineare Zeit- und Speicherkomplexität zu erreichen. Folglich haben Linear Transformer ein großes Potential für die effiziente Verarbeitung sehr langer Sequenzen.

Wir untersuchen zwei Ansätze zur Verwendung von Linear Transformer für das Lösen von PDEs und den damit verbundenen Problemen. Darüber hinaus führen wir einen umfassenden Vergleich zwischen den von uns entwickelten Transformer-Modellen und anderen Modellen zur Lösung parametrischer PDEs durch. Zu den Vergleichskriterien gehören die Genauigkeit für kurze und lange Rollouts, der Speicherverbrauch und die Inferenzzeiten. Die Ergebnisse zeigen, dass die von uns entwickelten Modelle mit aktuellen Modellen konkurrieren können und eine effiziente Lösung für das Lösen von PDEs bieten.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

CFD Computational Fluid Dynamics. 77

CNN Convolutional Neural Network. 29

DFFT Discrete Fast Fourier Transform. 35

FiLM Feature-wise Linear Modulation. 74

FNO Fourier Neural Operator. 39

LSTM Long Short-Term Memory. 25

MLP Multi-Layer Perceptron. 30

NLP Natural Language Processing. 21

OFormer Operator Transformer. 40

PDE Partial Differential Equation. 21

PINN Physics Informed Neural Network. 25

RLT Recurrent Linear Transformer. 80

RNN Recurrent Neural Network. 25

ViViT Video Vision Transformer. 47

ZSSR Zero-Shot Super-Resolution. 87

1 Introduction

The simulation of physical phenomena (e.g., atmospheric processes or fluid dynamics) relies on solving Partial Differential Equations (PDEs) [Olv+14]. Everyone implicitly deals with PDEs in their daily lives since weather forecasting is based on numerical simulations of the atmospheric processes which include solving a system of PDEs [Lyn08]. Optimizing the aerodynamics of vehicles also relies on solving PDEs for simulating the airflow around the vehicle [PTA12].

In recent years, Machine Learning models have been developed to address the task of solving PDEs ([LKA+20b], [Cao21], [BWW22]). Utilizing Machine Learning for solving PDEs has several advantages. For instance, Machine Learning models are often faster than classical numerical PDE solvers [TPL+22] and the models require no knowledge of the underlying PDE, only data to train the models is required [LKA+20b]. However, if knowledge about the underlying PDE is available, it can be added to the model by adding a PDE-specific term to the loss function [LZK+21]. Current models typically use multiple timesteps as the initial condition which isn't well-suited for real-world applications. In practical scenarios, there is often only one timestep as the initial condition available and a numerical PDE solver must be utilized to generate the next timesteps to use the Machine Learning model.

PDEs often contain a parameter which influences the evolution of the PDE. The current models typically do not consider the PDE parameters which makes it hard for the models to generalize to different PDE parameters [TAN23]. This encourages the development of models which take the PDE parameter as additional input. These parameter-conditioned models preferably generalize to different PDE parameters to replace multiple models trained on a single PDE parameter with one single model.

The Transformer model, proposed by Vaswani et al. [VSP+17] in 2017, and its many variants have achieved great success in Natural Language Processing (NLP) [DCLT18], Speech Processing [GQC+20] and even in domains such as Computer Vision [DBK+20] for a few years now. Transformers are also used in Reinforcement Learning [CLR+21] and Natural Sciences [JEP+21]. Due to their remarkable ability to effectively model long-range dependencies in sequential data, their field of application is steadily increasing. Crucial to the ability to handle long-range dependencies is the computation of attention which is achieved via the Scaled Dot-Product calculation in Vanilla Transformers. However, this operation is computationally expensive and grows quadratically with the length of the input sequence. Consequently, this leads to a bottleneck for very long sequences. To remedy this problem, several improvements that primarily aim to linearize the attention computation have been proposed. "Transformers are RNNs" proposed by Katharopoulos et al. [KVPF20] linearizes the computational complexity for computing attention, while also retaining the accuracy of the original softmax-based attention computation, by reformulating the attention mechanism with a kernel function. "FNet" proposed by Lee-Thorp et al. [LAEO22] reduces the quadratic

complexity of the softmax-based attention to a log-linear complexity by replacing it with a 2D Fourier transform. The reduced time and space complexity makes the linearized transformers promising for processing very long sequences.

Linearized transformers are a suitable choice for solving PDEs since the temporal domain (i.e., timesteps) and spatial domain (i.e., solution points per timestep) are essentially long sequences. Therefore, this bachelor's thesis investigates two approaches of utilizing (log) linear Transformers for solving PDEs and their associated problems. Moreover, we propose four transformer-based models for solving parametric PDEs and conduct a comprehensive comparison between our proposed models and state-of-the-art models. The evaluation criteria include accuracy for short and long rollouts, memory consumption, and inference times. The results demonstrate that our proposed models perform competitively with the current state-of-the-art models, providing an efficient solution for PDE solving. We focus on using only one timestep as the initial condition and predicting multiple timesteps in future since this scenario is better suited for real-world applications.

We use the benchmark "PDEBench" proposed by Takamoto et al. [TPL+22] for the training and testing of the models. PDEBench provides a large variety of datasets with PDE data, state-of-the-art baseline models for solving PDEs and a framework to easily train and test custom models. Therefore, we implement our models in PDEBench and use it for our experiments.

2 Background Knowledge

This section briefly defines Partial Differential Equations (PDEs) and our notation of discretized PDE data. We also illustrate three ways in which PDEs can be solved with Machine Learning. Furthermore, we explain the fundamentals of Transformers and introduce Linear Transformers and FNet Transformers.

2.1 Partial Differential Equations (PDEs)

Following the notations introduced in Brandstetter et al. [BWW22], Partial Differential Equations over one time dimension, denoted as $t = [0, T]$, and multiple spatial dimensions, denoted as $\boldsymbol{x} = (x_x, x_y, x_z, \ldots)^T \in \mathbb{X} \subseteq \mathbb{R}^D$ with D dimension of the PDE, can be expressed as

$$\begin{aligned} \partial_t u &= F(t, \boldsymbol{x}, u, \partial_x u, \partial_{xx} u, \ldots) \text{ with } (t, \boldsymbol{x}) \in [0, T] \times \mathbb{X} \\ u(0, \boldsymbol{x}) &= u(0, \cdot) = u^0(\boldsymbol{x}) = u^0 \text{ with } \boldsymbol{x} \in \mathbb{X} \\ B[u](t, \boldsymbol{x}) &= 0 \text{ with } (t, \boldsymbol{x}) \in [0, T] \times \partial\mathbb{X} \end{aligned} \tag{2.1}$$

where $u : [0, T] \times \mathbb{X} \to \mathbb{R}^c$ represents the solution function of the PDE that satisfies the initial condition $u(0, \boldsymbol{x})$ for time $t = 0$ and the boundary conditions $B[u](t, \boldsymbol{x})$ if $\boldsymbol{x}$ is on the boundary $\partial\mathbb{X}$ of the domain $\mathbb{X}$. c denotes the number of output channels of the PDE. In Equation (2.1), the notation $\partial_x u, \partial_{xx} u, \ldots$ represents the i-th order (where $i \in \{1, 2, \ldots, n\}$) partial derivatives $\frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, \ldots, \frac{\partial^n u}{\partial x^n}$. Scalars are represented as a small letter (e.g., a), vectors as a small and bold letter (e.g., $\boldsymbol{a}$) and matrices and tensors are represented with a capital letter (e.g., A). This work focuses on solving 1D PDEs (i.e. only one spatial dimension and it holds $D = 1$) only. Solving a PDE means finding the function $u(t, \boldsymbol{x})$ which satisfies the equation, the initial condition, and the boundary conditions.

2.1.1 Discretized PDE Data

The temporal and spatial domains are usually discretized to solve PDEs numerically. We introduce a few notations in this section since we use discretized data, generated by a numerical PDE solver, to train and test the models. The temporal domain $[0, T]$ is discretized into N timesteps yielding a sequence $(u(t_0, \cdot), u(t_1, \cdot), \ldots, u(t_{N-1}, \cdot))^T$ which describes the evolution of the PDE. $\Delta t = t_{i+1} - t_i$ denotes the temporal step size or resolution. We denote the number of timesteps used as the initial condition as N_I. Since we focus on predictions with one timestep as the initial condition it holds $N_I = 1$. The spatial domain $\mathbb{X}$ is also discretized into a grid X by discretizing each spatial dimension. Each point in the grid localizes a point in the spatial domain of the PDE. For 1D PDEs, the grid $X = (\left(\boldsymbol{x_i} = (x_{x_i})\right)_{i=1}^{s_x})^T \in \mathbb{R}^{s_x}$ and s_x denotes the

spatial resolution (i.e., number of spatial points) for the x-axis. Similarly, for 2D PDEs the grid $X = ((\boldsymbol{x_i} = (x_{x_i}, x_{y_i}))_{i=1}^{s_x \cdot s_y})^T \in \mathbb{R}^{(s_x \cdot s_y) \times 2}$ and s_x, s_y denote the spatial resolutions for the x and y axis, respectively. $u(t_i, X) = (u(t_i, \boldsymbol{x_1}), u(t_i, \boldsymbol{x_2}), \ldots, u(t_i, \boldsymbol{x_s}))^T \in \mathbb{R}^{s \times c}$ with $s = s_x \cdot s_y \cdot s_z \cdot \ldots$ contains the solutions at different spatial locations on the grid X. The PDE parameters are stacked to a vector $\boldsymbol{p} = (p_1, \ldots, p_j)^T \in \mathbb{R}^j$ where p_i represents the value of a PDE parameter. Figure 2.1 shows an example evolution of the 1D Burgers' equation.

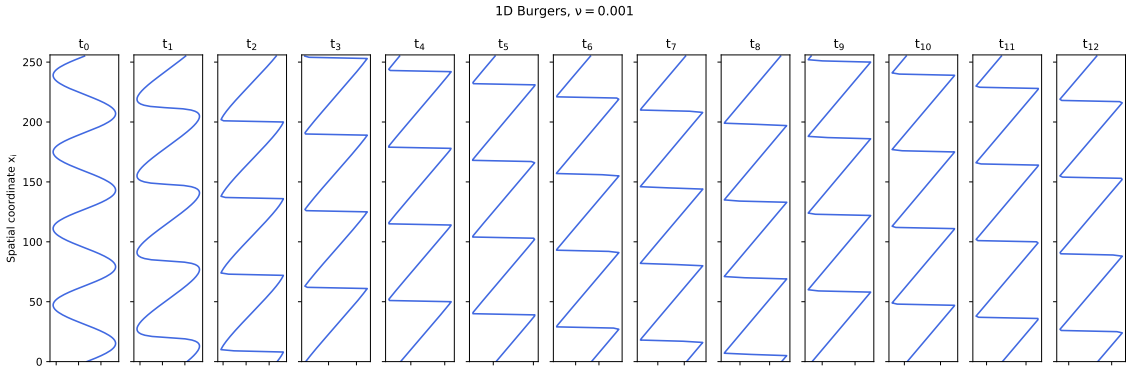


Figure 2.1: Example trajectory with 13 timesteps of 1D Burgers' PDE with parameter $\nu = 0.001$. Timestep $t_0 = 0$ is the initial condition that evolves.

2.2 Solving PDEs with Machine Learning

There are several ways of solving PDEs with Machine Learning and we introduce three main classes of models that are used to solve PDEs. Furthermore, we briefly explain how training and testing data could be generated. Using Machine Learning for solving PDEs has several advantages. For instance, Machine Learning models are often faster than classical numerical PDE solvers [TPL+22] and the models require no knowledge of the underlying PDE, only data to train the models is required [LKA+20b]. However, if knowledge about the underlying PDE is available, it can be added to the model by adding a PDE-specific term to the loss function [LZK+21].

2.2.1 Emulate Numerical PDE Solver

Solving PDEs can be considered as a high-dimensional time series prediction problem, where the solution $u(t_n, X)$ at timestep t_n for a fixed grid X is given and the solution $u(t_{n+1}, X)$ at timestep t_{n+1} needs to be predicted. A model f_θ could be expressed as

$$f_\theta : \mathbb{R}^{s \times c} \to \mathbb{R}^{s \times c} \text{ with } u(t_n, X) \mapsto u(t_{n+1}, X) \tag{2.2}$$

where θ denotes the parameters (i.e., weights and biases) of the model and c is the number of channels of the PDE. This class of models emulate a numerical PDE solver by mapping the timestep t_n to the future timestep t_{n+1}. The input $u(t_n, X) \in \mathbb{R}^{s \times c}$ and the output $u(t_{n+1}, X) \in \mathbb{R}^{s \times c}$ are vectors of a fixed size. Consequently, the spatial discretization (i.e., input and output vector size of the model) cannot be changed after training the model.

2.2.2 Neural Operators

Neural Operators, introduced by Li et al. [LKA+20a], solve the issue of fixed spatial discretization by learning a mapping between two infinite-dimensional spaces $\mathcal{A}$ and $\mathcal{B}$ (i.e., a mapping between two functions). Mathematically, Neural Operators can be expressed as

$$f_\theta : \mathcal{A} \rightarrow \mathcal{B} \text{ with } u(t_n, \boldsymbol{x}) \mapsto u(t_{n+1}, \boldsymbol{x}) \tag{2.3}$$

where θ denotes the parameters (i.e., weights and biases) of the model. Note that the input and output of the model are functions which can be evaluated at arbitrary spatial points $\boldsymbol{x}$. Therefore, the spatial discretization can be changed after training the model.

2.2.3 Learning the Solution Function

Instead of emulating a numerical PDE solver or learning a mapping between two functions, the model could directly learn the solution function $u(t, \boldsymbol{x})$. An example of this class of models are Physics Informed Neural Networks (PINNs) proposed by Raissi et al. [RPK19]. The model f_θ can be written as

$$f_\theta : (\mathbb{R} \times \mathbb{R}^D) \rightarrow \mathbb{R}^c \text{ with } (t, \boldsymbol{x}) \mapsto u(t, \boldsymbol{x}) \tag{2.4}$$

where θ denotes the parameters (i.e., weights and biases) of the model, $\boldsymbol{x}$ the spatial coordinates, t the temporal dimension and c the number of channels of the PDE.

2.2.4 Training and Testing Data

The training and testing data for the models could be generated by applying a numerical PDE solver or by observing a physical system. For instance, measuring parameters such as pressure, velocity and density in the atmosphere every hour could be used to train a model that evolves the state of the atmosphere (i.e., solves the underlying PDE). We use training and testing data that was generated by numerical PDE solvers. Figure 2.2 shows an example process of training a simple model to solve PDEs.

2.3 Vanilla Transformers

Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) networks proposed by Hochreiter and Schmidhuber [HS97] show great success in processing sequential data but struggle if the input sequences become longer. They cannot capture dependencies in long input sequences due to vanishing or exploding gradient [LLC+18]. Figure 2.3 shows an illustration of the problem. The RNN cell is iteratively applied to a token x_i of the input sequence $x = (x_0, \ldots, x_N)^T$ and outputs a corresponding hidden state h_i. The cell also uses the previous hidden state h_{i-1} as an input in addition to the current token x_i. Unfolding the iterative application of the RNN cell shows that the path between the first input token x_0 and the last token x_N grows with the sequence length N.

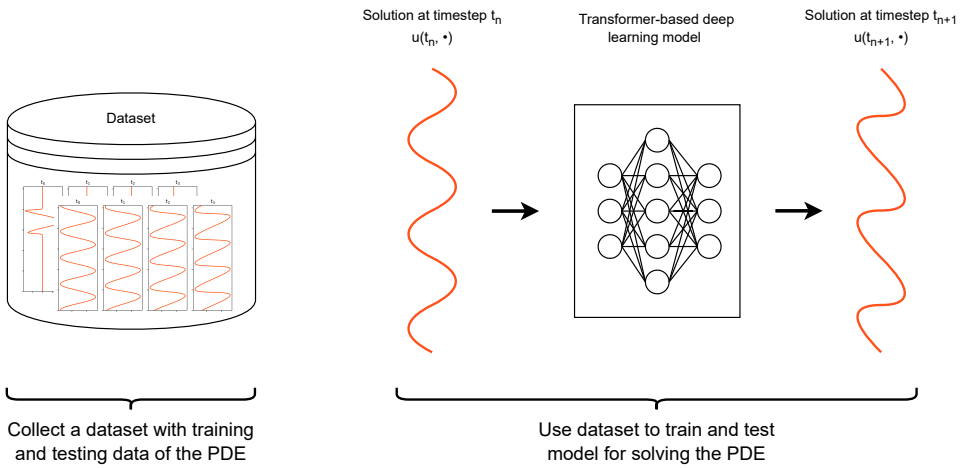


Figure 2.2: Example process of training a model for solving PDEs. A dataset with train and test data of the PDE must be collected to train and test the model that solves the PDE. The dataset could be generated by either utilizing a numerical PDE solver or by observing a physical system.

If the sequence is too long, the path between x_0 and x_N is too long such that h_N contains only little information of x_0. Therefore, h_N can barely represent the entire input sequence $x = (x_0, \ldots, x_N)^T$ which hinders to model from capturing all dependencies in the input sequence.

However, Transformers which were proposed by Vaswani et al. [VSP+17] can extract dependencies in very long sequences and successfully work on text data [DCLT18], audio sequences for speech recognition [GQC+20], and even on images which are divided into a sequence of smaller fields or patches [DBK+20].

In the following sections, we first explain the intuition of Transformers and attention and discuss Vanilla Attention (Scaled Dot-Product Attention or softmax-based attention) in detail. In addition, we introduce Linear Transformers and explain how they provide a speed-up for autoregressive training and inference. We also introduce FNet, a Transformer model that replaces Vanilla Attention with a 2D Fourier transform. Linear Transformers and FNet Transformers reduce the quadratic time and space complexity of Vanilla Attenion by reformulating or reinterpreting it.

2.3.1 Intuition

Transformers were originally proposed by Vaswani et al. [VSP+17] for translating text from one language into another language. The text is represented as a sequence of words or tokens and each token is represented as a vector in a latent space or embedding space. As a first step, the tokens of the text in the original language are processed by a Transformer block. This Transformer block takes a sequence $X \in \mathbb{R}^{N \times d}$ (N represents the number of tokens and d the dimension of the latent space) as an input and outputs an attention-refined latent representation $X' \in \mathbb{R}^{N \times d}$. We are using the terms *attention-refined latent representation*, *context vector* and *hidden state* interchangeably and refer to the output of the Transformer block.

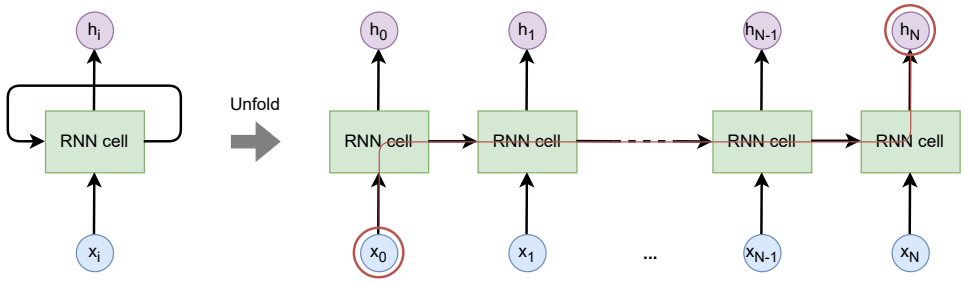


Figure 2.3: The RNN cell (e.g., simple linear layer with a non-linearity or an LSTM cell) is applied iteratively to the input tokens x_i of the input sequence $X \in \mathbb{R}^{N \times d}$. The cell takes the current token x_i and previous hidden state h_{i-1} as input and outputs a new hidden state h_i. Unfolding the application of the RNN cell shows that the path (red in the illustration) between the first token x_0 and the last token x_N grows with the sequence length. Consequently, h_N contains only a little information about x_0 and can barely represent the entire sequence.

The context vectors X' are a more meaningful representation of the tokens than the original latent representation X used as input. The input latent representation (input of Transformer) represents each token independently while the context vectors (output of Transformer) also take the entire sequence into account (i.e., other tokens). So, the context vector of a token represents not only the token itself but also depends on all other tokens in the sequence. Taking the entire sequence into account, when generating the context vectors, means that the model can capture long-range dependencies in the sequence.

When generating a context vector for one input token, the model assigns weights to all input tokens. This mechanism could be interpreted as a filter. Intuitively, the weights allow the model to focus on important dependencies (tokens) and ignore unimportant tokens in the sequence. This can be compared to how humans read and understand text. Some words in the sentence are very important for the meaning of a word (pay attention) and some words in the sentence are unimportant (pay less attention). Figure 2.4 illustrates the attention for generating a context vector for the word "solving".

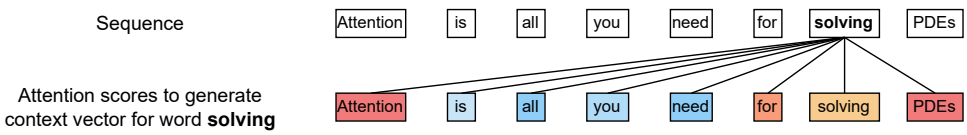


Figure 2.4: The Transformer would pay *more attention* to the words "Attention", "for", "solving" and "PDEs" to generate the context vector of the word "solving" since these words determine the meaning of the word "solving" in this sentence. The remaining words have a lower attention score because they are less important for the meaning of the word "solving".

Each context vector is essentially the weighted sum of all input tokens and the weights depend on the token for which a context vector is to be created. This can be described as

$$x'_i = x_0 \cdot w_{i,0} + x_1 \cdot w_{i,1} + \ldots + x_N \cdot w_{i,N} \qquad \forall i \in \{0, \ldots, N-1\} \tag{2.5}$$

where x'_i is the context vector of a token x_i and $w_{i,j}$ are the attention weights for the given token. The Transformer calculates the weights (attention weights) with Scaled Dot-Product Attention. Equation (2.5) also shows that each input token directly influences the context vector in contrast to the RNN in Figure 2.3, where the first token x_0 has less influence on the final hidden state. The path length of the first token x_0 and the final hidden state h_N of an RNN is N, while the path length of a token x_i and a context vector x'_j of a Transformer is always 1. This implies that Transformers can capture dependencies in sequences without struggling if the sequence length grows drastically.

2.3.2 Attention

The mechanism called "Scaled Dot-Product Attention" implements Equation (2.5). It calculates the weights or attention scores for the tokens and outputs the weighted sums x'_i for all input tokens.

Scaled Dot-Product Attention

The input sequence $X \in \mathbb{R}^{N \times d}$ is projected to three new representations (Q, K, V) by multiplying it with three weight matrices (W_Q, W_K, W_V). The weights of the matrices will be learned during the training. This means that the latent representation x_i of each token is projected to three new vectors (q_i, k_i, v_i). The query matrix Q and key matrix K are used to calculate the attention between the tokens and the value matrix V determines the value of the tokens (cf. querying a value from a database that can be accessed with a key). The context vectors are calculated by applying Equation (2.6).

$$\begin{aligned} Q &= XW_Q + b_Q \\ K &= XW_K + b_K \\ V &= XW_V + b_V \\ \text{Attention}(Q, K, V) &= softmax(\frac{QK^T}{\sqrt{d}})V \end{aligned} \tag{2.6}$$

The matrix dot product of Q and K^T is equivalent to calculating the dot products of all query vectors with all key vectors (N^2 combinations). The dot product of two vectors represents a similarity measure. Therefore, QK^T contains the similarities between the query and key vectors and has a resulting shape of $N \times N$. Each entry in the matrix is scaled by dividing it by $\sqrt{d}$. The row-wise application of the $softmax$ function is used to normalize the weights, to ensure they sum up to 1. It is similar to applying $softmax$ at the output layer of multi-class classification models [NIGM18]. Multiplying the attention matrix (result after applying the $softmax$ function) with the value matrix V is equivalent to calculating the weighted sum for all input tokens.

This calculation has a quadratic time and space complexity because N context vectors will be generated and each calculation requires the calculation of N dot products (for each input token). Alternatively, the complexity can also be recognised by the matrix $QK^T \in \mathbb{R}^{N \times N}$, which contains N^2 values and requires at least N^2 computation steps. The quadratic time and space complexity can lead to a bottleneck for long sequences (i.e., N very large).

Multi-Head Attention

$Softmax$ has the property to be very non-linear and to emphasize high values and shrink small values ("winner takes most"). Since $softmax$ is used in Scaled Dot-Product Attention to normalize the attention scores (Equation (2.6)), a few tokens get a very high attention score and the remaining tokens get a very low attention score. Thus, models with Scaled Dot-Product Attention proposed in Section 2.3.2 (called Single-Head Attention) focus only on a few tokens. To allow the model to focus on multiple tokens or positions, Vaswani et al. [VSP+17] proposed Multi-Head Attention. Instead of applying one Scaled Dot-Product Attention, the model performs h Scaled Dot-Product Attention calculations on smaller projections of Q, K, V. The output of each Scaled Dot-Product Attention (Attention-Head) is concatenated and fed into a linear layer. This adds more non-linearity to the model and allows the model to focus on different dependencies and positions in the sequence. The Attention-Heads in Multi-Head Attention can be compared to filters in Convolutional Neural Networks (CNNs). Multi-Head Attention is defined as

$$\begin{aligned} \text{Multi-Head-Attn}(Q, K, V) &= (\text{head}_1 || \dots || \text{head}_h) W_O \\ \text{head}_i &= \text{Attention}(QW_Q^i, KW_K^i, VW_V^i) \end{aligned} \tag{2.7}$$

The symbol $||$ represents the concatenation and $W_O \in \mathbb{R}^{(d_V \cdot h) \times d}$, $W_Q^i, W_K^i \in \mathbb{R}^{d \times d_Q}$ and $W_V^i \in \mathbb{R}^{d \times d_V}$ denotes learnable weight matrices. d_Q determines the dimension of the query and key vectors and d_V is the dimension of the value vectors.

Cross-Attention

The Scaled Dot-Product Attention from Section 2.3.2 and Multi-Head Attention described in the previous Section 2.3.2 are used to calculate Self-Attention in the Transformer block. Self-Attention means that the three representations Q, K, V are generated from the same sequence. This allows the model to generate a context vector for each token that also takes the entire sequence into account (i.e., captures dependencies in the sequence). Intuitively, this is what happens if humans read a text that is written in one language.

However, in translation tasks, the model operates on the sequence in the original language (sequence X) and on the sequence in the target language (sequence Y). Cross-Attention allows the model to couple both different sequences or representations by calculating attention between the sequences. This is essentially what happens if humans translate sentences from one language into another. We look at the sentences in the target language we already translated and at the sentences in the original language to determine which word to translate next. Paying attention allows us to ignore unimportant words that are already translated or unimportant as for now.

Cross-Attention is calculated by generating K, V from one sequence X (the sequence in the original language in translation tasks) and Q from another sequence Y (the sequence in the target language). The mechanism to calculate Cross-Attention is still Scaled Dot-Product Attention (Equation (2.6)) which is also used for Self-Attention, but Q is generated from Y and the matrices V, K are generated from X. Self-Attention and Cross-Attention are defined as

$$
\begin{aligned}
Q_X &= XW_Q + b_Q & Q_Y &= YW_Q + b_Q \\
V_X &= XW_V + b_V & V_X &= XW_V + b_V \\
K_X &= XW_K + b_K & K_X &= XW_K + b_K \\
\text{Self-Attn}(X) &= \text{Attention}(Q_X, K_X, V_X) & \text{Cross-Attn}(X, Y) &= \text{Attention}(Q_Y, K_X, V_X)
\end{aligned} \tag{2.8}
$$

where $X \in \mathbb{R}^{N \times d}$ represents a sequence and $Y \in \mathbb{R}^{N' \times d}$ represents another sequence. N and N' denote the sequence lengths and do not necessarily have to be equal, and d represents the dimension of the latent space. W_Q, W_V, Q_K and b_Q, b_V, b_K denote the corresponding weights and biases of the linear layers. The left side shows the calculation for Self-Attention and the right side for Cross-Attention. Cross-Attention can be used to mix two different sequences by calculating attention between the sequences.

2.3.3 Positional Encoding

The position information of the tokens gets lost in the attention mechanism. This implies that the attention mechanism does not consider whether a token is at the start or end of a sequence, even though the token's position in the sequence holds valuable information. Consequently, it is necessary to incorporate additional positional information into the latent representation of the tokens. A suggested approach by Vaswani et al. [VSP+17] involves utilizing an absolute positional encoding utilizing sine and cosine functions. This absolute positional encoding is defined as

$$
PE_{pos,2i} = sin\left(\frac{pos}{1000^{\frac{2i}{d}}}\right) \qquad PE_{pos,2i+1} = cos\left(\frac{pos}{1000^{\frac{2i}{d}}}\right) \tag{2.9}
$$

where pos denotes the position of the token in the sequence and i is the dimension index in the embedding vector with dimension d. The vector PE_{pos}, containing the positional encoding for position pos, is added to the input latent representation of the token at position pos.

2.3.4 Transformer Block

Combining the attention mechanism introduced in the previous sections, and a few other components yields a Transformer block. Figure 2.5 shows the Transformer block proposed by Vaswani et al. [VSP+17] with all internal components. A Transformer consists of the following components:

- Positional Encoding: Adds positional information to the tokens since the attention mechanism doesn't consider the order of the tokens.
- Multi-Head Self-Attention or Multi-Head Cross-Attention: Allows information flow between the tokens by calculating the weighted sum of the input tokens.
- Pointwise Multi-Layer Perceptron (MLP): The MLP processes the output of the attention mechanism and is shared across all tokens.

- Residual Connection (**Add** & Norm): Residual connections help the model to converge by keeping the loss landscape smooth, similar to ResNets proposed by He et al. [HZRS15]. The attention mechanism can be interpreted as a filter that aggressively blocks information, meaning that small changes in the input data may not result in a change in the output ("dead spot") [Roh21]. The residual connections enable data to bypass certain components, such as the attention mechanism or the MLP, in order to ensure gradient flow during backpropagation, even if there is a "dead spot" in the gradient. They allow data to bypass components by adding the input of a component to their output.
- Layer Normalization (Add & **Norm**): Since the components in the Transformer are heavily non-linear, a small change in the input data or model's parameters could lead to high output values which could lead to problems since subsequent components are sensitive to the inputs' magnitude and distribution [Roh21]. Layer normalization [BKH16] transforms the values by shifting (i.e., changing the mean value $\bar{x}$) and scaling (i.e., changing the standard deviation σ) of the values. This avoids the internal covariate shift and improves training stability and efficiency.

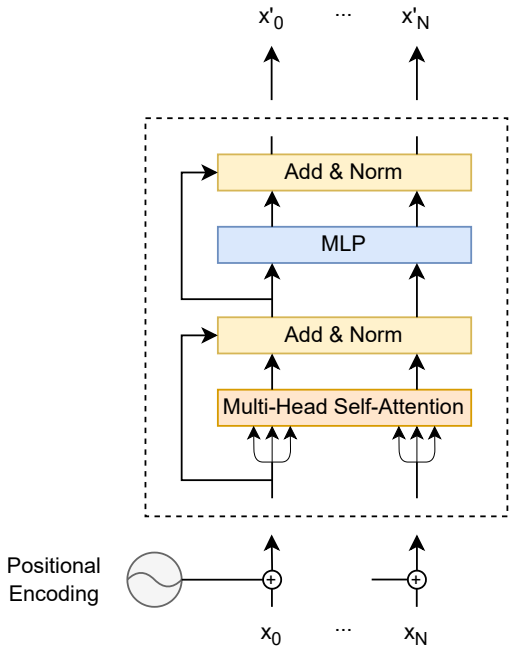


Figure 2.5: Transformer block with Multi-Head Self-Attention. The positional encoding is applied to all input tokens to add positional information to the latent representation of the tokens. Applying the Transformer block, consisting of Multi-Head Self-Attention, residual connections and layer normalizations (Add & Norm) as well as an MLP which is shared across all tokens, produces an attention-refined latent representation X' of the tokens which is more meaningful than the input latent representation X.

2.3.5 Transformer Encoder-Decoder Architecture

The original Transformer architecture was introduced to translate sentences (text) from one language into another. First, the text in the original language is processed by several Transformer blocks with Multi-Head Self-Attention that generate context vectors X' for the input latent representation X. The context vectors are more meaningful than the input representation since the context vectors consider the word and the meaning of the word in the entire sequence (i.e., dependencies in the sequence). Essentially, the Transformer blocks act as an encoder which "encodes" the input text in the original language.

The translation or the generation of the next token in the target language happens in Transformer blocks with Multi-Head Cross-Attention. These blocks take the previously translated tokens Y as input, apply Self-Attention to it to generate context vectors and use Multi-Head Cross-Attention to "make queries" from Y to the encoded input text X', yielding the next token in the target language. Intuitively, these Transformer blocks predict the next word in the target language by "looking" at the encoded input text and the previously translated words. It essentially acts as a decoder. Figure 2.6 illustrates the proposed Transformer Encoder-Decoder architecture with an example translation of the English input text "Transformers for solving PDEs" to the German text "Transformers zum Lösen von PDEs". Several Transformer encoder and decoder blocks are stacked together to build the Transformer Encoder-Decoder architecture.

2.3.6 Transformer Sequence Length Generalization Problem

In theory, the length of the input sequence (e.g., number of words) of the Transformer can be arbitrarily increased after the training. The increase of the Transformers' context window after the training is possible since all components are shared between the tokens (e.g., the mechanism to generate the latent representation or MLP) and the attention calculation is also independent of the input sequence length. But in practice, the Transformer model usually applies additional positional encoding to the input tokens which limits the maximum length of the context window. This means that the context window length is limited to the maximum context window length seen during training. If the sequence at inference is longer than the maximum sequence seen during training the model will most likely produce poor predictions. This problem occurs because the positional encoding is out-of-distribution for sequences which are longer than the sequences seen during training [RDG+23]. Techniques such as random positional encoding proposed by Ruoss et al. [RDG+23] show success in solving the length generalization problem for specific tasks.

2.4 Linear Transformers

The quadratic time and space complexity of the attention mechanism yields a bottleneck for long sequences. To address this problem, Katharopoulos et al. [KVPF20] proposed Linear Transformers that reformulate the Scaled Dot-Product Attention by using a kernel function. The new formulation yields an attention calculation with a linear time and space complexity. First of all, they reformulate the attention calculation to

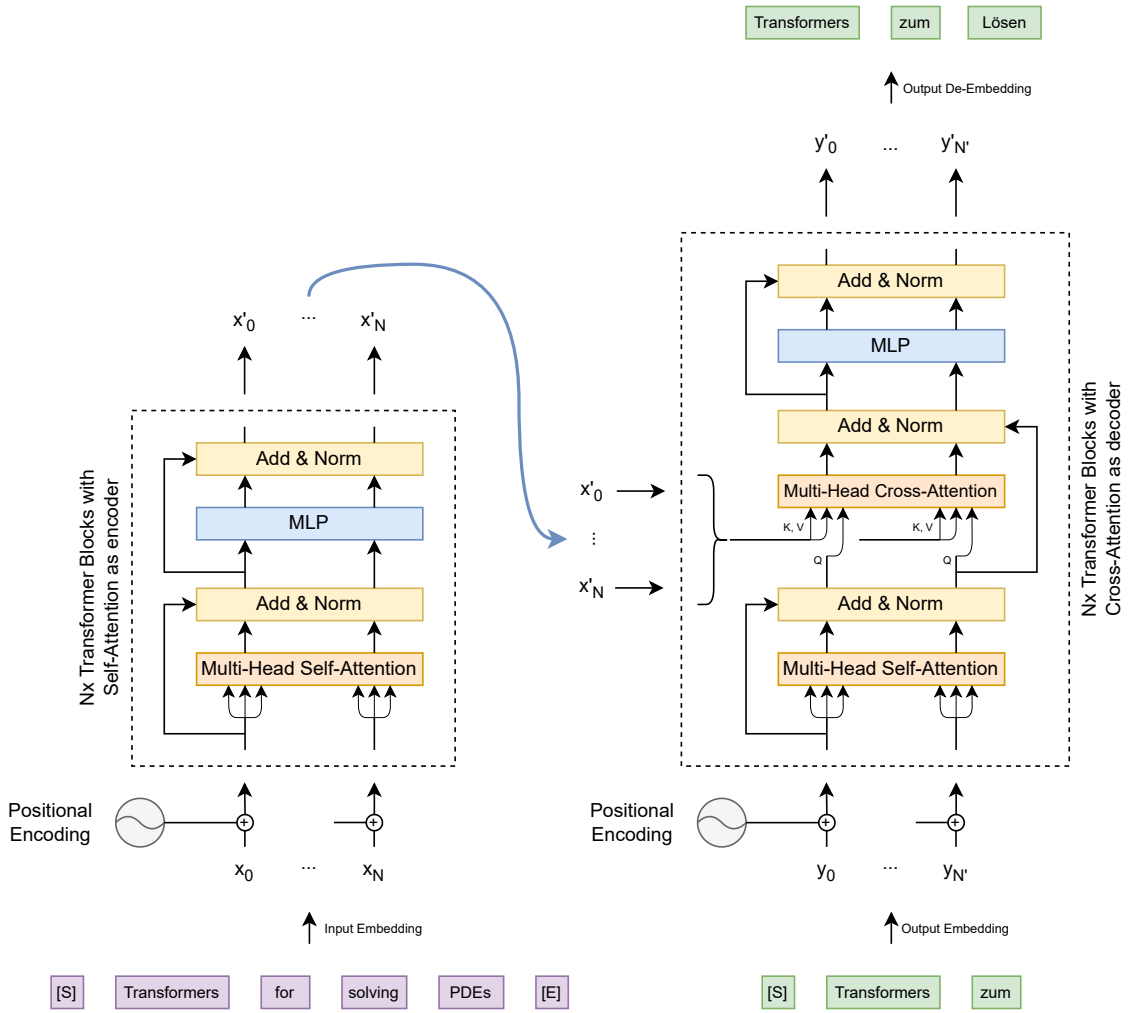


Figure 2.6: Transformer Encoder-Decoder architecture for translation tasks. The input words are embedded in an embedding space or latent space. The tokens $[S]$ and $[E]$ are special tokens which mark the start and end of a text, respectively. The encoder encodes the input tokens and the decoder predicts the next token by using the encoded input tokens and the previously translated tokens. Usually, a suitable number of encoder blocks and a suitable number of decoder blocks are stacked together. Depending on the task, encoder and decoder can be used separately.

$$\text{Attention}(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V = V' \Leftrightarrow V'_i = \frac{\sum_{j=1}^{N} \text{sim}(Q_i,K_j)V_j}{\sum_{j=1}^{N} \text{sim}(Q_i,K_j)} \tag{2.10}$$

where V'_i denotes the i-th row of matrix V and $\text{sim}(q,k) = \exp(\frac{q^T k}{\sqrt{D}})$. $\text{sim}(\cdot,\cdot)$ is a similarity function which measures the similarity of two vectors.

2.4.1 Kernel Function as a Similarity Measure

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 $k(x, y) : (\mathbb{R}^d \times \mathbb{R}^d) \rightarrow \mathbb{R}_+$ 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 map the input to some high-dimensional space (Mercer's theorem), we can rewrite the kernel function as $k(x, y) = \Phi(x)^T \Phi(y)$. This leads to a new interpretation of the attention equation:

$$V_i' = \frac{\sum_{j=1}^{N} \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^{N} \text{sim}(Q_i, K_j)} = \frac{\sum_{j=1}^{N} \Phi(Q_i)^T \Phi(K_j) V_j}{\sum_{j=1}^{N} \Phi(Q_i)^T \Phi(K_j)} = \frac{\Phi(Q_i)^T \sum_{j=1}^{N} \Phi(K_j) V_j^T}{\Phi(Q_i)^T \sum_{j=1}^{N} \Phi(K_j)} \tag{2.11}$$

Since $\Phi(Q_i)^T$ is independent of the index i of the sum, we can make use of the associative property of matrix multiplication and pull $\Phi(Q_i)^T$ in front of the sum. The value of $\sum_{j=1}^{N} \Phi(K_j) V_j^T$ and $\sum_{j=1}^{N} \Phi(K_j)$ needs only to be calculated once because they can be reused (time complexity of $O(N)$) and V_i needs to be calculated for all N tokens (time complexity of $O(N)$). This results in linear time and space complexity.

Used Feature Function

Katharopoulos et al. [KVPF20] use

$$\begin{aligned} \Phi(x) &= elu(x) + 1 \\ elu(x) &= \begin{cases} x & \text{if } x > 0 \\ exp(x) - 1 & \text{otherwise} \end{cases} \end{aligned} \tag{2.12}$$

as the feature function in their experiments. The output of this function is always non-negative and thus fulfils the required property of a similarity measure. They are using elu [CUH15] instead of $ReLU$ [Aga18] as a non-linearity to avoid having a gradient of 0 for negative inputs.

2.4.2 Linear Attention for Autoregressive Predictions

Katharopoulos et al. [KVPF20] show that their Linear Attention also provides a speed-up for autoregressive tasks. When making autoregressive predictions, the input sequence of the model grows with every prediction step since the previous model's output is appended to the input sequence. This implies that the attention from Equation (2.10) must be recalculated in each prediction step. With Vanilla Attention there is no other choice than recalculating the attention in each prediction step since the sum in Equation (2.10) depends on Q_i which changes in each prediction step. Whereas the sum in Linear Attention of Katharopoulos et al. [KVPF20] is independent of Q_i which allows us to split the sum into two parts:

$$V_i' = \frac{\Phi(Q_i)^T \sum_{j=1}^{N} \Phi(K_j) V_j^T}{\Phi(Q_i)^T \sum_{j=1}^{N} \Phi(K_j)} = \frac{\Phi(Q_i)^T (\sum_{j=1}^{N-1} \Phi(K_j) V_j^T + \Phi(K_N) V_N^T)}{\Phi(Q_i)^T (\sum_{j=1}^{N-1} \Phi(K_j) + \Phi(K_N))} \tag{2.13}$$

By introducing two variables $s_i = \sum_{j=1}^{N-1} \Phi(K_j)V_j^T + \Phi(K_N)V_N^T) = s_{i-1} + \Phi(K_i)V_i^T)$ and $z_i = \sum_{j=1}^{N-1} \Phi(K_j) + \Phi(K_N) = z_{i-1} + \Phi(K_i)$ it becomes visible that we can reformulate the attention calculation to

$$
\begin{aligned}
s_i &= \begin{cases} 0 & \text{if } i = 0 \\ s_{i-1} + \Phi(K_i)V_i^T & \text{otherwise} \end{cases} \\
z_i &= \begin{cases} 0 & \text{if } i = 0 \\ z_{i-1} + \Phi(K_i) & \text{otherwise} \end{cases} \\
V_i' &= \frac{\Phi(Q_i)^T s_i}{\Phi(Q_i)^T z_i}
\end{aligned}
\tag{2.14}
$$

This results in a speed-up for autoregressive predictions since the model only needs to maintain two states s_i and z_i and does a simple addition instead of recalculating attention from scratch in every prediction step. Katharopoulos et al. [KVPF20] call this model a "Recurrent Linear Transformer" because the two states s_i and z_i can be compared to hidden states in an RNN.

2.5 FNet: Transformers with Fourier Transforms

Lee-Thorp et al. [LAEO22] interpret the Scaled Dot-Product Attention as a mechanism that mixes tokens by calculating the weighted sum of the input tokens. Therefore, they experiment with different mechanisms to mix tokens and propose a model called "FNet". Their proposed model replaces the Scaled Dot-Product Attention with two Discrete Fast Fourier Transforms (DFFTs) to mix the input tokens. The FNet "attention" can be described with the following equation

$$
\text{Attention}_{\text{FNet}}(X) = Re(DFFT_t(DFFT_s(X))) \tag{2.15}
$$

where $DFFT$ denotes a function that calculates the discrete Fourier transform and Re is a function to return the real part of each entry in a complex-valued tensor. $X \in \mathbb{R}^{N \times d}$ represents the input latent representation. The FNet "attention" first applies a DFFT along the spatial or embedding dimension ($DFFT_s$) and then a second DFFT ($DFFT_t$) along the temporal domain (i.e., sequence length). They only keep the real part of the complex-valued output of the two DFFTs to avoid adapting the remaining parts of the Transformer (e.g., multi-layer perceptron and Add & Norm). Note that they introduced only a replacement for Self-Attention, but not a replacement for Cross-Attention.

2.5.1 Discrete Fourier Transform

The discrete Fourier transform takes a complex-valued sequence $a = (a_0, \ldots, a_{N-1}) = (a_n)_{0 \leq n \leq N}$ with $a_n \in \mathbb{C}$ and a length of N as input and decomposes it in its frequency components $c_k \in \mathbb{C}$. Each frequency component or mode represents the amplitude and phase of the corresponding frequency. The discrete Fourier transform essentially transforms the sequence from the physical domain to the frequency domain (a.k.a Fourier space). It is defined as

$$c = DFT(a) = (c_k)_{k=0}^{N-1} \text{ and } c_k = \sum_{n=0}^{N-1} a_n \cdot e^{-2\pi i \cdot \frac{nk}{N}} \in \mathbb{C} \text{ for } k = 0, \ldots, N-1 \tag{2.16}$$

where $e^{i\varphi} = cos(x) + i \cdot sin(x)$ represents Euler's formula. The inverse Fourier transform reconstructs the sequence in the physical domain from its frequency components c_k. This can be expressed as

$$a = IDFT(c) = (a_n)_{n=0}^{N-1} \text{ and } a_n = \frac{1}{N} \sum_{k=0}^{N-1} c_k \cdot e^{2\pi i \cdot \frac{nk}{N}} \in \mathbb{C} \text{ for } n = 0, \ldots, N-1 \tag{2.17}$$

Primarily, the discrete Fourier transform is calculated with the Discrete Fast Fourier Transform algorithm (DFFT) of Cooley and Tukey [CT65] or by multiplying the Fourier matrix $M = (e^{-2\pi i \cdot \frac{nk}{N}})_{k=0}^{N-1} = ((e^{\frac{-2\pi i}{N}})^{nk})_{k=0}^{N-1} \in \mathbb{C}^{N \times N}$ (Vandermonde matrix containing the unit roots) with the input sequence. The DFFT algorithm has a time complexity of $O(N \log(N))$ and the matrix multiplication has a complexity of $O(N^2)$. The IDFT can be calculated with $IDFT(c) = \frac{1}{N} \cdot \overline{DFT(\overline{c})}$ and has therefore the same time complexity as the DFT.

The discrete Fourier transform can be interpreted as a token mixing mechanism since each frequency component c_k depends on the entire sequence and therefore, each component c_k contains information about the entire sequence.

2.5.2 FNet Transformer Block

The FNet Transformer block is identical to a Vanilla Transformer block with Self-Attention, except that the first one uses a 2D Fourier transform instead of Scaled Dot-Product Attention. Figure 2.7 shows an FNet Transformer block with the Fourier transform, MLP as well as the residual connections and layer normalizations (Add & Norm).

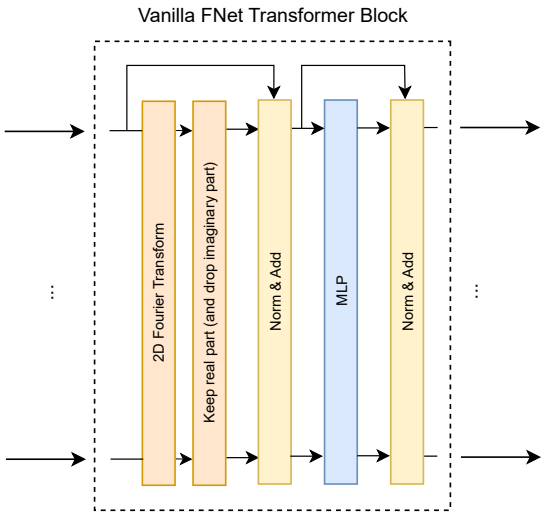


Figure 2.7: Architecture of the Vanilla FNet Transformer. The two residual connections and layer normalizations (Add & Norm) as well as the MLP are identical to the Vanilla Transformer. Only the token mixing mechanism is different (2D Fourier transform instead of Scaled Dot-Product Attention).

3 Related Work

The goal of this section is to provide an overview of related research. We examine Machine Learning models that solve PDEs by introducing three popular models and their results. In addition, we present PDEBench, an open-source framework that allows the training and testing of Machine Learning models for solving PDEs. In the end, we investigate previous work that aims to solve parametric PDEs by considering the PDE parameter as model input.

3.1 Solving PDEs with Machine Learning

3.1.1 Fourier Neural Operator (FNO)

This paper of Li et al. [LKA+20b] introduces a novel architecture called Fourier Neural Operator (FNO). Neural Networks usually model a mapping between two finite-dimensional Euclidian spaces which leads to the problem that they are fixed to a spatial resolution when used for solving PDEs. The proposed Fourier Neural Operator is a Neural Operator that learns a mapping between infinite-dimensional function spaces (i.e., mapping between functions). Thus, the spatial discretization of the input and output functions can be arbitrary which allows changing the spatial resolution after the training.

Their proposed method 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 domain allowing for an efficient and expressive architecture. Their approach differs from our introduced transformer-based methods since our models either encode a timestep in a fixed latent space and emulate a numerical PDE solver or apply Linear Attention on the spatial domain and directly learn the underlying solution function.

The paper demonstrates the effectiveness of the Fourier Neural Operator on the 1D Burgers' equation, Darcy flow, and 2D CFD (Navier-Stokes equation). They take multiple timesteps as the initial condition and map it to multiple future timesteps. Thus, the experiment setup differs from ours since we use one timestep as the initial condition and predict multiple timesteps in an autoregressive fashion. The benchmarks show that the Fourier Neural Operator achieves higher accuracy compared to previous Machine Learning models such as U-Net [RFB15] and ResNet [HZRS15]. They also demonstrate the zero-shot super-resolution capabilities (i.e., increasing spatial resolution after the training) of the FNO on 2D CFD.

Additionally, they measure the inference times and show that the Fourier Neural Operator is significantly faster compared to numerical PDE solvers. The paper also discusses the limitations of traditional numerical solvers (e.g., the tradeoff between computational time and discretization) and the potential of data-driven methods for solving PDEs. Overall, the Fourier Neural Operator provides a promising approach for solving PDEs efficiently and accurately.

3.1.2 Galerkin and Fourier Transformer

In the paper, Cao [Cao21] introduces the novel application of Self-Attention for learning a Neural Operator. The author improves the effectiveness of the Scaled Dot-Product Attention by re-interpreting and linearizing it. These modified attention mechanisms are referred to as Fourier and Galerkin Attention. He employs the proposed attention mechanisms in a transformer-based architecture that maps from one function to another for solving PDEs.

First, he provides an alternative way to interpret the matrices $Q, K, V \in \mathbb{R}^{N \times d}$ 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 him to improve the effectiveness of the attention mechanism by linearizing it, yielding Fourier and Galerkin Attention. Furthermore, he suggests replacing the post-layer normalization with pre-layer normalization for the K and V matrices. Putting the components together yields a Galerkin or Fourier Transformer for solving PDEs. The Fourier and Galerkin Transformer apply the Self-Attention on the spatial domain. One of our architectures (TFormer) draws inspiration from his approach of using the Transformer to apply spatial Self-Attention, which facilitates zero-shot super-resolution. Our other models are very different to the introduced approach since they apply temporal Self-Attention instead of spatial Self-Attention.

The paper presents several PDE-solving experiments, including the 1D Burgers' equation and Darcy flow. The results of these experiments show that the newly proposed Fourier and Galerkin Transformers outperform a Vanilla Transformer in terms of computational time and memory as well as accuracy. He also compares the accuracy of the Fourier and Galerkin Transformer against the FNO and shows that his models outperform FNO.

3.1.3 Operator Transformer (OFormer)

The research of Li et al. [LMF23] is based on the Galerkin and Fourier Transformer. Existing approaches such as FNO and Galerkin or Fourier Transformer are restricted in having the same grid X 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 from the input points. The authors solve this problem by adding Cross-Attention to the model to allow querying for arbitrary spatial points. In addition, they suggest further improvements and call the resulting model Operator Transformer (OFormer).

One crucial component of OFormer is a modified Galerkin or Fourier Transformer with Self-Attention. Similar to the work of Cao [Cao21], the Transformer is applied to the spatial domain of the input solution to update it iteratively. The authors extend the Fourier and Galerkin Attention from Self-Attention to Cross-Attention which is used to capture the relationships between the input solutions at some locations and the query locations. This decouples the output spatial domain from the input spatial domain and enables querying at different locations.

The authors demonstrate that the model achieves state-of-the-art performance on standard PDE benchmark problems such as 1D Burgers, Darcy flow and CFD (Navier-Stokes equation). They include benchmark values for regular and irregular grids. In addition, the paper also encompasses a benchmark where the input grid is different from the output grid to show the ability to process different input and output spatial points. In general, the model achieves competitive performance

with the advantage of decoupling the input and output grids. The authors highlight that Linear Attention is still computationally expensive for high-resolution grids since each point is treated as a token.

Furthermore, they provide a comprehensive ablation study to investigate the effect of the type of attention (Galerkin or Fourier), the number of parameters of the model, and the use of additional positional encoding. The results show that the additional positional encoding improves the performance compared to omitting it. Galerkin Attention also provides some improvement compared to Fourier Attention.

3.2 PDEBench: Scientific Machine Learning Benchmark

With PDEBench, Takamoto et al. [TPL+22] has introduced an open-source benchmark framework which is specifically designed for Machine Learning models that address simulation tasks based on PDEs. The authors highlight that the utilization of Machine Learning for simulations is increasing, but there is currently a lack of user-friendly benchmark frameworks that encompass a comprehensive collection of datasets.

PDEBench includes the datasets and the code for the training and testing of Machine Learning models for solving PDEs. The datasets encompass a wide range of PDEs, varying from simple toy examples to more complex and practical problems. The testing code provided by PDEBench entails standard metrics such as RMSE and custom metrics that incorporate properties of the underlying physics. It also includes the code of recent baseline models for solving PDEs. The authors emphasize that PDEBench allows researchers to easily extend the benchmark framework with new models or datasets.

The authors have already included benchmark values in their paper for popular Machine Learning models, including FNO [LKA+20b], PINN [RPK19], and U-Net [RFB15]. The models were trained to predict multiple timesteps in future in an autoregressive fashion which is similar to our benchmark setup. However, the models were trained using multiple timesteps as initial condition and, therefore, it is necessary for us to re-compute the benchmark values, making it impossible to directly compare their results to ours.

We use PDEBench as a benchmark framework in our work and extend it with our proposed transformer-based models.

3.3 Solving Parametric PDEs

The paper of Takamoto et al. [TAN23] discusses the use of Machine Learning models for solving parametric PDEs. The authors note that although ML-based methods have shown great success in solving PDEs, they often do not consider the PDE parameter. Thus, it makes it hard for the models to generalize to different PDE parameters. To tackle this problem, the authors introduce a Channel Attention mechanism guided by PDE Parameter Embeddings (CAPE) module that considers the PDE parameter. In addition, they suggest training the model with a curriculum learning strategy.

By incorporating their CAPE module into existing Machine Learning models, the models improve their capabilities to generalize to unseen PDE parameters. The curriculum learning strategy starts with teacher-forcing training and smoothly moves to fully autoregressive training which helps to further improve the training stability and generalization capabilities.

To evaluate their proposed CAPE module, the authors conduct a benchmark on 1D Burgers, 1D Advection, and 2D CFD (Navier-Stokes equation). They train the models using a set of PDE parameters and test their performance on set with the seen and unseen PDE parameters. The benchmark shows the effectiveness of the proposed CAPE module and curriculum strategy and compares the errors to base models that do not consider the PDE parameter or to PDE parameter-conditioned models. For our work, we adopt their PDE parameter combinations and benchmark setup. However, we use slightly different hyperparameters, such as the number of modes of the FNO, to improve the expressivity of the FNO. Hence, a direct comparison of their results with ours is not feasible.

The authors emphasize several benefits of their method, including an improved generalization to unseen PDE parameters without a significant increase in the inference time and the model's parameter count. To conclude, this paper presents a method for solving parametric PDEs that considers the PDE parameter as input and improves the generalization to unseen PDE parameters.

4 Methodology

We investigate two ways of utilizing Transformers for solving 1D PDEs. Furthermore, we propose four transformer-based models for solving 1D parametric PDEs. The first two models solve PDEs by emulating a numerical PDE solver and utilize a Linear and a Vanilla Transformer. The third architecture also emulates a numerical PDE solver and is based on the idea of FNet using Fourier transforms instead of Scaled Dot-Product or Linear Attention. The fourth model unites the idea of Neural Operators and Physics Informed Neural Networks into a model that is continuous in the temporal domain and spatial domain. This architecture uses Linear Transformer blocks and Linear Attention since Linear Attention promises a speed-up for processing long sequences.

4.1 Solving PDEs with Transformers

There are essentially two ways of employing Transformers for solving 1D PDEs. The temporal domain (i.e., timesteps) and the spatial domain (i.e., solutions per timestep) are usually discretized. Thus, the temporal domain and the spatial domain are represented as a sequence to which attention could be applied. Since the sequences are very long (e.g., ≥ 40 timesteps = tokens for the temporal domain and ≥ 256 spatial points = tokens on the spatial domain) Transformers are a suitable choice for processing the sequences.

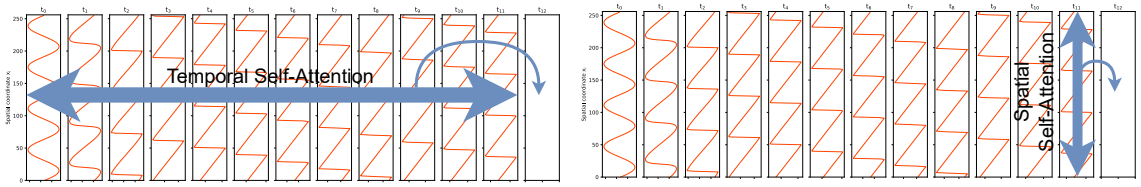
4.1.1 Temporal Attention

Applying *attention* to the temporal dimension allows the model to capture temporal dependencies which could help the model to predict the solution for the next timestep ([GZ22], [HGP+22]). One option could be to provide the model the solutions of the latest 10 timesteps (for example) as input and apply attention to it. Another option would be to provide the model the entire historical context, the solutions from the initial condition to the latest timestep, as input and apply attention to this sequence. The latter has the advantage that the model gets as much information as possible as input and that we don't have to decide whether the model should take the solutions of the latest 10 or latest 20 timesteps. So, we don't need to choose a suitable hyperparameter for the context window of the model. Figure 4.1a illustrates temporal attention.

4.1.2 Spatial Attention

Alternatively, *attention* could be applied to the spatial domain allowing the model to capture spatial dependencies for predicting the next timestep ([Cao21], [LMF23]). For instance, the solution of the latest timestep could be used as the model's input and attention could be applied to the spatial

dimension of the latest timestep. The global receptive field of Transformers, particularly of attention, is also useful since the spatial points at the boundary should be able to exchange information (e.g., could be helpful for periodic boundary conditions). Figure 4.1b illustrates spatial attention.



(a) Temporal Self-Attention **(b)** Spatial Self-Attention

Figure 4.1: Comparison of temporal and spatial Self-Attention. Temporal Self-Attention (Figure 4.1a) is applied to the temporal domain (i.e., timesteps) to capture temporal dependencies which could be helpful for the model to predict the solution for the future timestep t_{12}. Spatial Self-Attention (Figure 4.1b) is applied to the spatial domain of the solution of the latest timestep to capture spatial dependencies for predicting the solution of the next timestep t_{12}.

4.2 Recurrent Linear Transformer for Solving 1D PDEs

We introduce our first model that solves 1D parametric PDEs by emulating a numerical PDE solver and by applying temporal Self-Attention. Thus, the model is fixed to a temporal and spatial resolution. Instead of using only the latest timestep t_n as the model's input, we provide the entire history of timesteps $(u(t_0, X), u(t_1, X), \ldots, u(t_n, X))^T \in \mathbb{R}^{(n+1)\times s\times c}$ to the model to enable it to capture information from all previous timesteps (i.e., modelling temporal dependency). Since we are using Transformers to predict the evolution of PDEs, we can provide the entire history (a.k.a context) to the model without risking that the model will not capture the dependencies as the input sequence gets longer and longer. Mathematically, the model can be expressed as the following function

$$f_\theta : (\mathbb{R}^{(n+1)\times s\times c} \times \mathbb{R}^s \times \mathbb{R}^j) \to \mathbb{R}^{s\times c} \text{ with } ((u(t_0, X), \ldots, u(t_n, X)), X, \boldsymbol{p}) \mapsto u(t_{n+1}, X) \quad (4.1)$$

where θ represents the parameters or weights and biases of the neural network. s denotes the spatial resolution of the grid X and c is the number of channels. To predict multiple timesteps or to do longer rollouts, previous predictions are appended to the input sequence (i.e., autoregressive prediction). Figure 4.2 illustrates the autoregressive predictions of the model.

This idea is similar to the idea of Geneva and Zabaras [GZ22] or to transformer architectures for next-word predictions in NLP (e.g., GPT-1 of Radford et al. [RNSS+18]) except that next-word predictions are classifications while we solve a regression problem. We also use a Recurrent Linear Transformer instead of a Vanilla Transformer to make use of the autoregressive speed-up. The Recurrent Linear Transformer with Multi-Head Self-Attention takes the latent representations of the previous timesteps as input and predicts the solution for the future timestep.

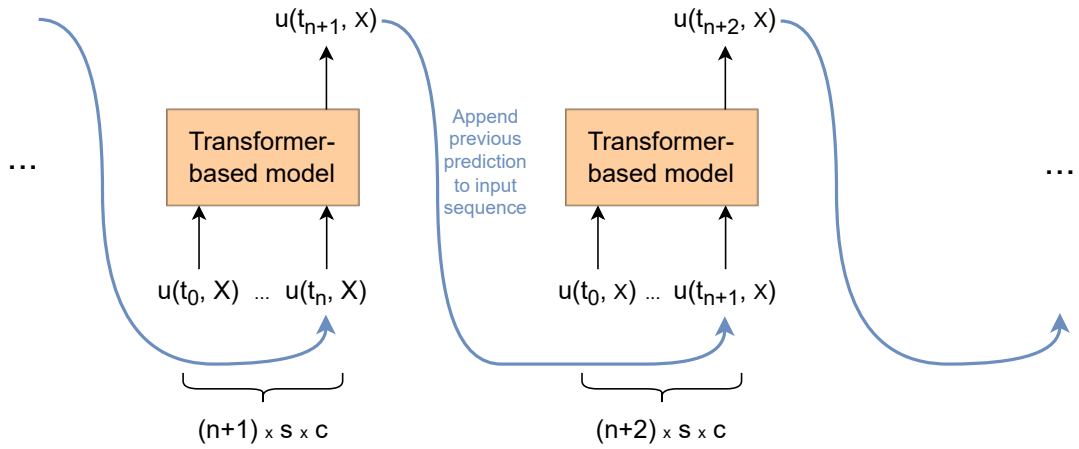


Figure 4.2: Autoregressive predictions to predict multiple timesteps in the future. The model takes an input sequence of solutions as input and predicts the next timestep. $u(t, X)$ is a vector of size $s \times c$ (s represents the spatial resolution and c the number of channels) and denotes the solutions of all spatial points at timestep t.

4.2.1 Model Architecture

In this section, we provide a detailed explanation of all components of the model. The model interprets the solutions of each timestep $u(t_i, X)$ as a token. These tokens are then passed through either a linear layer (for PDEs with a single channel) or a CNN (for PDEs with multiple channels) to generate a latent representation for each timestep or token. The latent representations are then processed by a Recurrent Linear Transformer, which applies temporal Self-Attention to effectively model the dynamics of the PDE. The latent representation outputted by the Transformer depicts the solution for the next timestep which is then passed through an MLP to project it back to the physical space. Figure 4.3 shows the detailed architecture of the model for PDEs with a single channel.

Latent Representation

Each solution $u(t_i, X) = (u(t_i, \boldsymbol{x_1}), u(t_i, \boldsymbol{x_2}), \ldots, u(t_i, \boldsymbol{x_s}))^T \in \mathbb{R}^{s \times c}$ of a timestep t_i is interpreted as a token. First of all, the tokens or timesteps must be mapped to a latent space with dimension $d = 96$. Intuitively, the representation of a token (solution $u(t_i, X)$ at timestep t_i) in the latent space should encode the "state of the physical system" at timestep t_i. This intuition is similar to the intuition of Geneva and Zabaras [GZ22]. Creating a latent representation for $u(t_i, X)$ also ensures that each component of the vector in the latent representation may contain information of $u(t_i, \boldsymbol{x_j})$ from all spatial coordinates x_j (i.e., collect information of neighbouring solutions in the spatial domain to allow the model to capture spatial dependencies). Depending on the number of channels c of the PDE, we apply different mechanisms to create the latent representation.

PDEs with a Single Channel If the PDE has only one channel (i.e., $c = 1$), we apply a single-layer perceptron with the identity function as the activation function (i.e., no activation function) to map the solutions of the timestep to a vector in the latent space. The single-layer perceptron

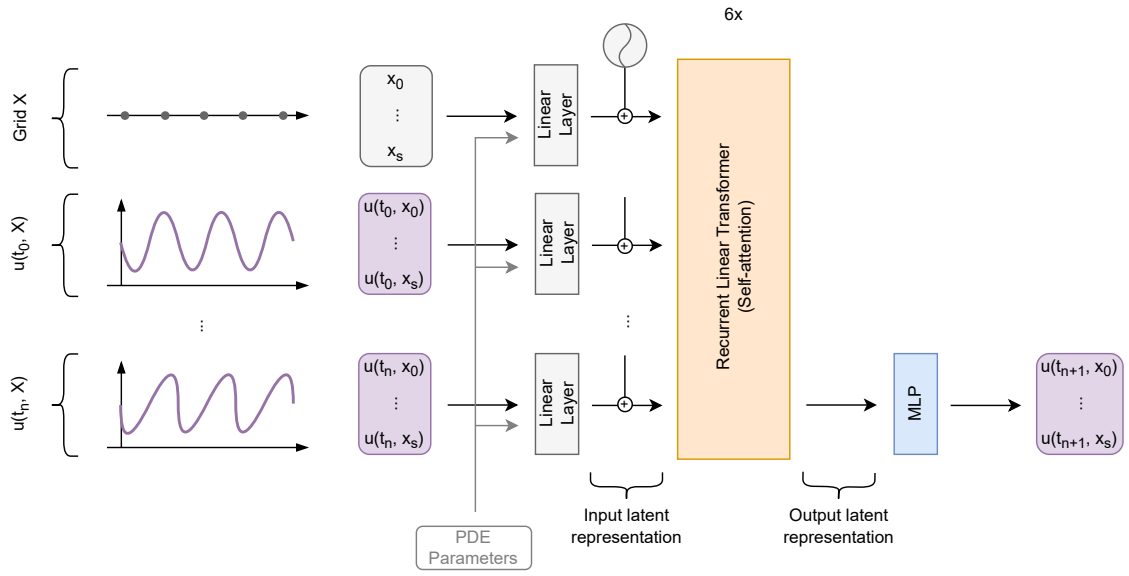


Figure 4.3: Architecture of the Recurrent Linear Transformer for solving 1D PDEs with a single channel. The grid X and the PDE parameters $\boldsymbol{p}$ are used as an additional input feature. The PDE parameters are fed into the pointwise linear layer (single-layer perceptron) to encode each timestep w.r.t to the PDE parameters and an MLP is used to map the solution's latent representation back to physical space.

or linear layer is shared across all timesteps. The grid $X = (\boldsymbol{x_0}, \ldots, \boldsymbol{x_s})^T$ with s spatial resolution represents the points at which a solution $u(t, \boldsymbol{x_i})$ was collected. X is treated as a regular timestep which means that it is processed by the linear layer to get a latent representation and then is fed into the Transformer blocks. The latent representation for a timestep can be expressed as

$$\begin{aligned} \text{Latent}(u(t_i, X), \boldsymbol{p}) &= \text{Latent}((u(t_i, \boldsymbol{x_1}), \ldots, u(t_i, \boldsymbol{x_s}))^T, (p_1, \ldots, p_j)^T) \\ &= \text{Linear-Layer}((u(t_i, \boldsymbol{x_1}), \ldots, u(t_i, \boldsymbol{x_s}))^T || (p_1, \ldots, p_j)^T) \\ &= (u(t_i, \boldsymbol{x_1}), \ldots, u(t_i, \boldsymbol{x_s}), p_1, \ldots, p_j))^T W + b \end{aligned} \tag{4.2}$$

where $W \in \mathbb{R}^{(s+j)\times d}$ denotes the learnable weight matrix and $b \in \mathbb{R}^d$ the corresponding bias. The operator $||$ represents the concatenation of two vectors (e.g., $\boldsymbol{c} = (\boldsymbol{a}||\boldsymbol{b})$ is the concatenation of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$). $\boldsymbol{x} \in \mathbb{R}^j$ is the vector with the PDE parameters. Figure 4.4 shows the mechanism.

PDEs with Multiple Channels For PDEs with multiple channels (i.e., $c \geq 2$) such as 1D CFD, we propose using a 1D Convolutional Neural Network (CNN) consisting of convolution layers, pooling layers, and non-linearities to generate the latent representation for the Transformer. The CNN and linear layer are shared across all timesteps. The grid is used as an additional channel for the CNN, in contrast to the mechanism for single-channel PDEs, where the grid is used as an additional input token. The approach can be expressed as

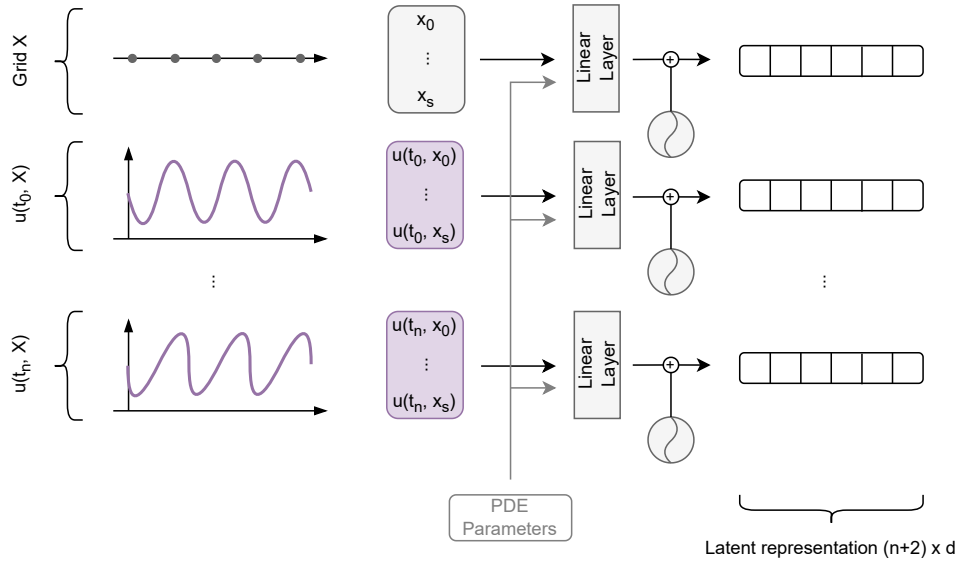


Figure 4.4: Mechanism to generate the latent representation of the PDE solutions for PDEs with one single channel.

$$\begin{aligned} \text{Latent}(u(t_i, X), \boldsymbol{p}) &= \text{Linear-Layer}(\text{CNN}((u(t_i, \boldsymbol{x_1}), \ldots, u(t_i, \boldsymbol{x_s}))^T) || (p_1, \ldots, p_j)^T) \\ &= (\text{CNN}((u(t_i, \boldsymbol{x_1}), \ldots, u(t_i, \boldsymbol{x_s})), (p_1, \ldots, p_j))^T W + b \end{aligned} \tag{4.3}$$

where CNN denotes the 1D Convolutional Neural Network and W and b the learnable weight matrix and bias for the final linear layer. Figure 4.5 illustrates the mechanism to generate a latent representation for $u(t_i, X)$ with multiple channels. Alternatively, the CNN could be replaced with a second Transformer, that applies attention on the different PDE channels, to generate the input vectors for the Transformer that applies temporal attention. This approach would be similar to Video Vision Transformers (ViViTs) proposed by Arnab et al. [ADH+21] whereas the attention over the PDE channels can be compared to spatial attention in ViViTs.

PDE Parameter Embedding To enable the model to use the information provided by the PDE parameters, we add the PDE parameters as an additional input to the model. Using the PDE parameters should enable the model to generalize to unseen PDE parameters (desideratum). The PDE parameters are added to the linear layer and enable it to encode a solution $u(t_i, X)$ w.r.t. to the PDE parameters. It can be easily shown that this mechanism ensures that the PDE parameters $\boldsymbol{p}$ influence all components of the vectors in the latent space. By reformulating Equation (4.2) to

$$\begin{aligned} \text{Latent}(u(t_i, X), \boldsymbol{p}) &= (u(t_i, \boldsymbol{x_1}), \ldots, u(t_i, \boldsymbol{x_s}), p_1, \ldots, p_j))^T W + b \\ &= (u(t_i, \boldsymbol{x_1}), \ldots, u(t_i, \boldsymbol{x_s}))^T W_s + (p_1, \ldots, p_j)^T W_p + b \end{aligned} \tag{4.4}$$

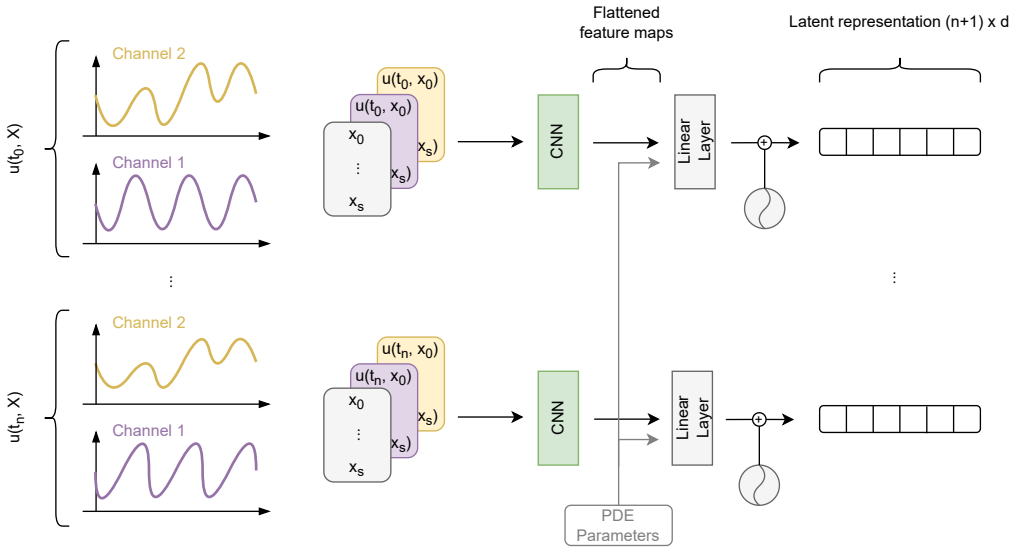


Figure 4.5: Mechanism to generate the latent representation of the PDE solutions for PDEs with multiple channels.

where symbol || denotes the concatenation of two vectors and $W \in \mathbb{R}^{(s+j)\times d}$ is a learnable weight matrix and $b \in \mathbb{R}^d$ the corresponding bias vector (it holds $W = \begin{pmatrix} W_s \\ W_p \end{pmatrix}$), it becomes visible that an increase or decrease in $\boldsymbol{p}$ results in an increase or decrease of the components of the output latent representation. So, the PDE parameters influence the entire latent vector and make it hard for the subsequent layers to ignore the PDE parameters.

Positional Encoding Since the temporal order gets lost when computing Self-Attention in the Transformer, we have to utilize some additional positional encoding. We add absolute positional encodings [VSP+17] to the latent representations of the tokens (i.e., solutions at some timestep). In this case, the position of the tokens in the sequence represents some temporal information (a.k.a time). Thus, it is crucial to add this information to the tokens.

Recurrent Linear Transformer

In the second stage, the latent representations of the timesteps are fed into Recurrent Linear Transformer blocks with Multi-Head Self-Attention that generate attention-refined latent representations. Using a Recurrent Linear Transformer promises a speed-up since the model is trained and tested in an autoregressive fashion. The Self-Attention is applied along the temporal dimension which allows the Transformer to capture temporal dependencies in the input sequence. Intuitively, the Transformer "looks" at all timesteps to capture the temporal dependencies and outputs a latent representation of the "state of the physical system" for the next timestep. This output latent representation is then mapped to the physical space by a following multi-layer perceptron outputting the solution $u(t_{n+1}, X)$ for the future timestep t_{n+1}.

Map Latent Representation back to the Physical Space

To map the solution's latent representation (i.e., the output of the Recurrent Linear Transformer) back to the physical space, we apply a multi-layer perceptron (MLP) with three layers and $GeLU$ as the activation function. For PDEs with a single channel, the output of the MLP is simply in $\mathbb{R}^s$ and for PDEs with multiple channels, the MLP outputs $\mathbb{R}^{(s \cdot c)}$ which is reshaped to $\mathbb{R}^{s \times c}$. Figure 4.6 shows the mechanism to map the solution's latent representation back to the physical space for a PDE with one single channel.

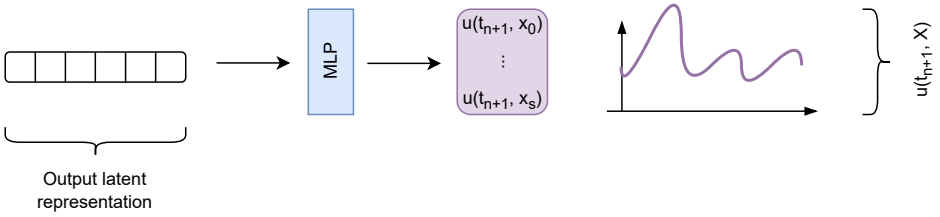


Figure 4.6: Mechanism to map the solution's latent representation back to the physical space for PDEs with one single channel.

4.2.2 Variable Number of Timesteps as the Initial Condition

The proposed model can handle a different number of timesteps N_I as the initial condition. This means that the model can be trained on N_I timesteps as the initial condition and used to do inference with $N_I' \neq N_I$ timesteps as the initial condition. It is also possible to train the model on trajectories that have different initial condition sizes. Later, we make use of the variable initial condition sizes for the proposed sliding window approach, to do longer rollouts than done during training.

4.3 Vanilla Transformer for Solving 1D PDEs

The Recurrent Linear Transformer in the architecture introduced above (orange box in Figure 4.3) could be replaced with some arbitrary Transformer. Thus, we replace the Recurrent Linear Transformer in the architecture with an identical Vanilla Transformer to get a baseline and a comparison of Linear Attention and Scaled Dot-Product Attention. The Recurrent Linear Transformer outputs only the context vector that corresponds to the last timestep $u(t_n, X)$ of the sequence $(u(t_0, X), \ldots, u(t_n, X))$, in contrast to the Vanilla Transformer that outputs context vectors for each timestep $u(t_i, X)$ with $0 \leq i \leq n$. Therefore, we discard the context vectors that correspond to timesteps $u(t_i, X)$ with $0 \leq i < n$ and use only the context vector that corresponds to $u(t_n, X)$. These context vectors could be discarded since they represent past solutions which are already predicted. Figure 4.7 illustrates the architecture of Vanilla Transformers for solving 1D PDEs.

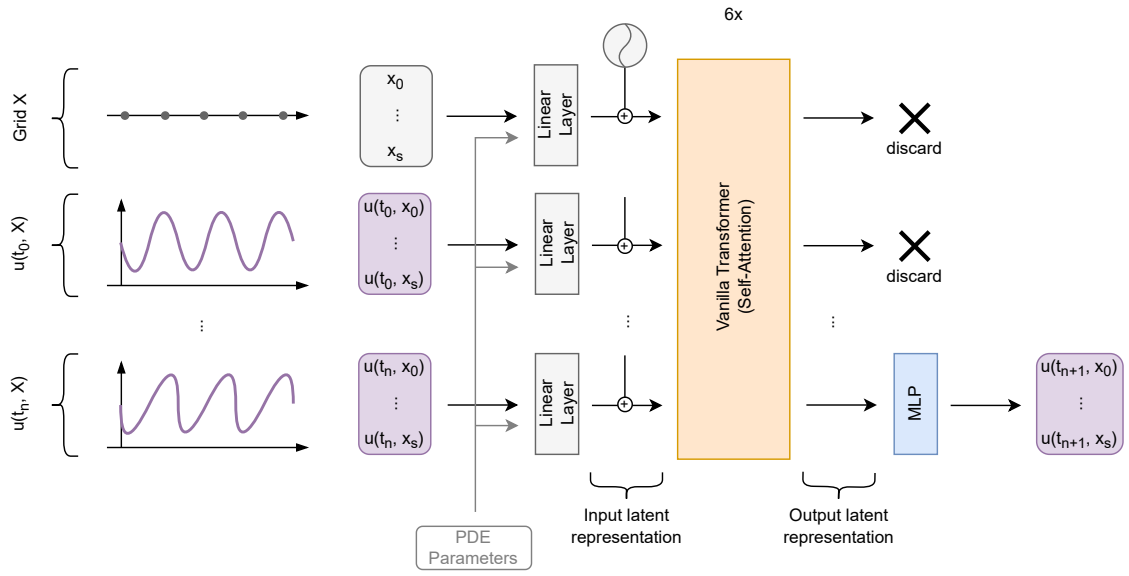


Figure 4.7: Vanilla Transformer inserted in the presented architecture. The grid X and the PDE parameters $\boldsymbol{p}$ are used as an additional input feature. The PDE parameters are fed into the pointwise linear layer (single-layer perceptron) to encode each timestep w.r.t to the PDE parameters, and an MLP is used to map the solution's latent representation back to the physical space. Note that we discard the context vectors for solutions $u(t_i, X)$ with $0 \leq i < n$ since they describe solutions which are already predicted.

4.4 PDE FNet

We also replace the Recurrent Linear Transformer in the previous architecture (Figure 4.3) with an FNet Transformer since solving PDEs in the Fourier space can lead to mathematical simplification. Particularly, the theorem that differentiation is equivalent to multiplication in Fourier space and the convolution theorem (convolution is equivalent to multiplication in Fourier space) can lead to a simplification [LKA+20b]. Unfortunately, the Vanilla FNet Transformer wasn't able to produce good results when using it for solving PDEs (i.e., no convergence during training) in our experiments (see loss plots in Appendix B.1). Therefore, we decided to investigate and modify the architecture of the FNet Transformer to get better results for solving PDEs. Note that we improve the Vanilla FNet Transformer block and not the base architecture for solving PDEs (Figure 4.3) proposed in the previous sections.

First, we introduce our PDE FNet Transformer block and explain the suggested modifications. We also investigate the computational complexity of the proposed model and highlight the omission of additional positional encoding. Furthermore, we insert the PDE FNet Transformer in the architecture for solving PDEs.

4.4.1 PDE FNet Transformer Block

We call our improved FNet Transformer models "PDE FNet small" and "PDE FNet large" and propose to modify the FNet "attention" by

- replacing the discrete fast Fourier transform (DFFT) with a DFFT for real-valued inputs
- keeping the real and imaginary part of the DFFT
- replacing the 2D DFFT with a 1D DFFT inputs along the temporal domain
- introducing learnable parameters in the Fourier space (complex-valued MLP)
- and mapping the output back to physical space with an inverse discrete fast Fourier transform (IDFFT).

The proposed modifications lead to a better performance and are only related to the attention mechanism. Our proposed attention mechanism (PDE FNet attention) can be expressed as

$$\text{Attention}_{\text{PDE FNet small}}(X) = IDRFFT_t(DRFFT_t(X)W + b) \tag{4.5}$$

where $W \in \mathbb{C}^{d \times d}$ and $b \in \mathbb{C}^d$ represent learnable parameters. $DRFFT$ denotes a 1D discrete fast Fourier transform for real-valued inputs and $IDRFFT$ the inverse transformation. The expressivity of the model can be increased by replacing the complex-valued single-layer perceptron with a complex-valued multi-layer perceptron and a non-linearity. We propose PDE FNet large attention which is described as

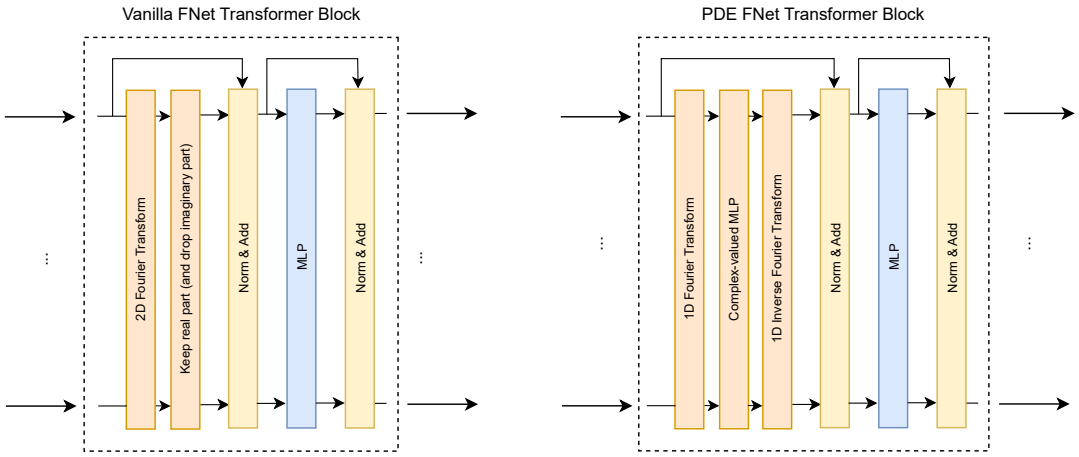
$$\begin{aligned} \text{Attention}_{\text{PDE FNet large}}(X) &= IDRFFT_t((\sigma(DRFFT_t(X)W_1 + b_1))W_2 + b_2) \\ \sigma(Z) &= \mathbb{C}ReLU(Z) = ReLU(Re(Z)) + ReLU(Im(Z))i \text{ with } Z \in \mathbb{C}^{t \times d} \end{aligned} \tag{4.6}$$

where $W_1 \in \mathbb{C}^{d \times (d \cdot 2)}$ and $W_1 \in \mathbb{C}^{(d \cdot 2) \times d}$ are learnable complex-valued weight matrices and $b_1 \in \mathbb{C}^{d \cdot 2}$ and $b_2 \in \mathbb{C}^d$ are the corresponding complex-valued biases. σ represents a non-linearity for complex values. We use $\mathbb{C}ReLU$ proposed by Trabelsi et al. [TBS+17] as a non-linearity in our experiments. $DRFFT$ denotes the discrete fast Fourier transform for real-valued inputs and $IDRRFT$ is the inverse Fourier transform.

Figure 4.8 shows a comparison of the Vanilla FNet Transformer and our proposed PDE FNet Transformer. The components of the PDE FNet Transformer block are similar to the FNet Transformer or Vanilla Transformers proposed by Vaswani et al. [VSP+17]. It consists of the proposed PDE FNet Attention (1D FFT, complex-valued MLP and 1D inverse DFFT), two residual connections and layer normalizations (Add & Norm) as well as a real-valued MLP.

4.4.2 Suggested Modifications to the FNet Transformer

In the following section, we explain our design decisions for the proposed PDE FNet Transformer or more accurately for the proposed PDE FNet Attention.



(a) Vanilla FNet Transformer Block **(b)** Proposed PDE FNet Transformer Block

Figure 4.8: Comparison of the Vanilla FNet Transformer block and the proposed PDE FNet Transformer block. We only modify the token mixing mechanism (cf. attention mechanism).

Discrete Fourier Transform for Real-Valued Inputs

Since the latent vectors (i.e., the input for the attention mechanism) are always real-valued, the output matrix of the DFFT is Hermitian-symmetric (i.e., the entries of the negative frequencies are the complex conjugates of the positive frequencies). This implies that the output vector of each token, after applying FNet attention, is symmetric because the real parts of complex conjugates are the same. Thus, we propose to replace the DFFT with a DFFT for real-valued inputs to avoid redundant information (symmetric output vectors). The following example shows the symmetry of the output vectors (the same values are marked with the same colour) after applying FNet attention which we want to avoid. In the following example, $X \in \mathbb{R}^{2\times 8}$ represents the latent representation of two tokens with dimension $d = 8$.

$$X = \begin{pmatrix} 0.25 & 1 & 0.75 & 1.25 & -0.25 & -1 & 2 & 1.5 \\ 1 & -1.25 & 0.5 & 2 & -0.5 & 1 & 1.25 & 1.5 \end{pmatrix} \in \mathbb{R}^{2\times 8}$$

$$\text{Attention}_{\text{FNet}}(X) = \begin{pmatrix} 11 & 1.6 & -4 & 2.4 & -1 & 2.4 & -4 & 1.6 \\ 0 & 2.5 & -1.5 & -4.5 & 1 & -4.5 & -1.5 & 2.5 \end{pmatrix} \in \mathbb{R}^{2\times 8}$$

Keeping both Real and Imaginary Part

We suggest keeping the real and imaginary part of the DFFT which allows the model to create a more meaningful representation. Keeping real and imaginary parts also ensures that each token's representation can be projected back to the physical space with an inverse DFFT. The Vanilla FNet Transformer keeps only the real-valued part of the 2D DFFT which doesn't allow the model to project the values from Fourier space back to the physical space. Dropping the complex-valued parts of the 2D DFFT looks to us that a lot of information is being lost. Therefore, our proposed PDE FNet attention keeps the real and imaginary parts of the DFFT.

1D DRFFT along Temporal Dimension

The attention mechanism is the only component in our proposed architecture (Recurrent Linear Transformer and Vanilla Transformer for PDEs) that can capture temporal dependencies in the PDE because it considers all input timesteps (i.e., calculates attention between the timesteps and returns the weighted sum) while the other components are applied pointwise to only one timestep. Therefore, we suggest using a 1D DFFT along the temporal dimension, instead of a 2D DFTT along the temporal and spatial dimensions, because we are only interested in capturing temporal dependencies with the DFFT. The combination of 1D DFFT along the temporal dimension, the pointwise complex-valued MLP and real-valued MLP along the spatial dimension allow the model to learn spatio-temporal dependencies for solving the PDE. The $DFFT_s$ along the spatial domain in the Vanilla FNet Transformer allows information to flow from one spatial point (i.e., a component in the latent representation of the solution) to another, but information can also flow from one spatial point to another via the shared real-valued MLP that is applied to each token. Thus, it seems legitimate to us to drop the DFFT along the spatial domain.

Learnable Parameters in Fourier Space

We added learnable parameters to our proposed PDE FNet attention to enable the model to transform the input in the Fourier space. The shared and pointwise complex-valued MLP modifies the output coefficients of the Fourier transform.

Mapping back to the Physical Space

Mapping the output of each PDE FNet Attention block back to the physical space allows us to keep the remaining parts of the Transformer (i.e., residual connection, layer normalization, and MLP) untouched. We apply an inverse DRFFT to the temporal domain to project the output back to physical space.

4.4.3 Computational Complexity of the PDE FNet Transformer

The time complexity of our proposed PDE FNet attention is $O(d \cdot N \log(N) + N \cdot d^2)$ with d dimension of the latent space and N number of timesteps or tokens. The Vanilla FNet Transformer of Lee-Thorp et al. [LAEO22] has a time complexity of $O(d \cdot N \log(N) + N \cdot d \log(d))$ and the Vanilla Transformer proposed by Vaswani et al. [VSP+17] of $O(d \cdot N^2 + N \cdot d^2)$.

4.4.4 Positional Encoding

As proposed by Lee-Thorp et al. [LAEO22], we omit additional positional encoding since the Fourier transforms retain the order of the tokens or timesteps. Omitting additional positional encoding also ensures that PDE FNet small or large does not have to deal with the transformer length generalization problem mentioned in Section 2.3.6. This allows the model to extend the context window size arbitrarily after the training.

4.4.5 Simple Attention Replacement for Time-Series Processing

The introduced PDE FNet Transformer is a simple replacement for Transformers with attention, particularly for Transformers for time-series processing. Attention mechanisms that mix tokens by calculating attention scores can be interpreted as a filter that focuses on important tokens and ignores important ones. So, these models essentially calculate the solution of the PDE by outputting a weighted sum of the timesteps as the prediction. The Vanilla FNet Transformer also tries to mix tokens with a 2D Fourier transform. On the other hand, our proposed PDE FNet works differently by transforming the input sequence along the temporal domain in Fourier space, modifying the sequence in Fourier space, and mapping it back for modelling the dynamics of the PDE or the trend in the time series. Thus, the proposed PDE FNet calculates the solution of the PDE by capturing temporal dependencies of the sequence in Fourier space.

4.4.6 PDE FNet for Solving PDEs

Inserting the proposed PDE FNet Transformer into our architecture yields PDE FNet for solving PDEs. Figure 4.9 shows the proposed model.

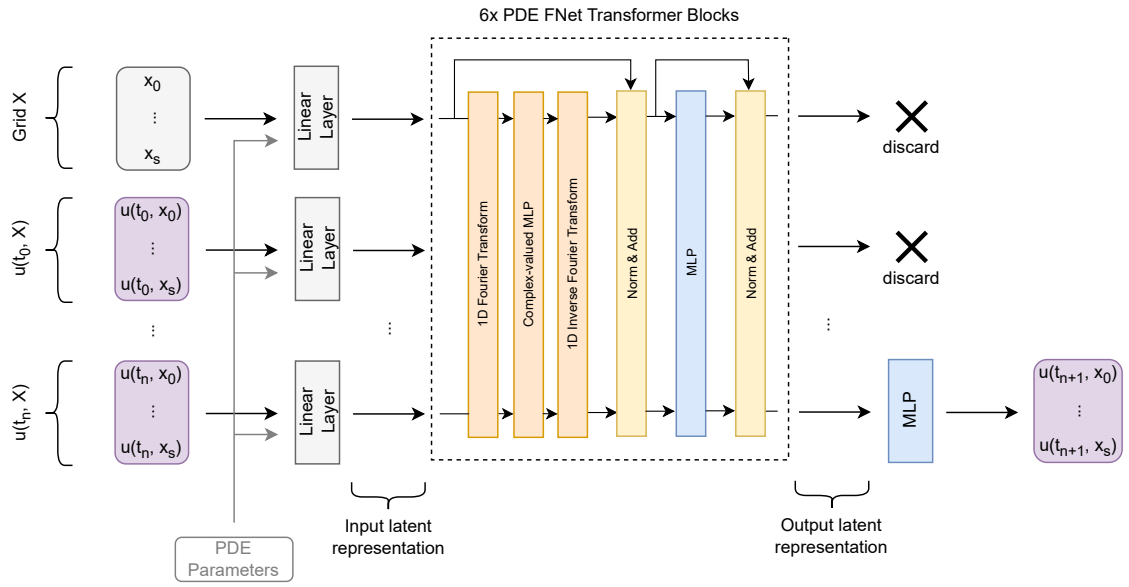


Figure 4.9: Architecture utilizing PDE FNet Transformer blocks for solving 1D PDEs with the grid and PDE parameters as additional input feature. The PDE parameters are fed into the pointwise linear layer (single-layer perceptron). The combination of 1D DRFFT, complex-valued MLP and 1D inverse DRFFT replaces the 2D DFFT of the Vanilla FNet Transformer blocks or the Self-Attention of Vanilla Transformers.

4.5 Time-Transformer (TFormer)

The base architecture introduced above encodes the solutions $u(t, X) = (u(t, x_1), \ldots, u(t, x_s))^T$ of a timestep t as a vector in the latent space. The dimension of the latent space is fixed and cannot be changed after training which yields the problem that the spatial resolution (i.e., size of the vector $u(t, X) \in \mathbb{R}^{s \times c}$) is also fixed and cannot be changed. The only way of processing solutions with a larger spatial resolution than during training is by downsampling the spatial resolution of the input to the resolution seen during training (e.g., subsampling) and upsampling (e.g., interpolating) the prediction to the desired spatial resolution. Applying downsampling results in an information loss as well as contradicts the main task of learning the solution function $u(t, \boldsymbol{x})$ which can be evaluated at some arbitrary spatial point $\boldsymbol{x}$ and temporal point t.

Changing the spatial resolution after the training could occur in real-world applications since training on low resolutions reduces the training time, and inference at higher resolutions lowers the risk of missing crucial dynamics. Thus, we decide to view solving PDEs as learning the underlying function u instead of emulating a numerical PDE solver, and propose Time-Transformer (TFormer). The idea of TFormer unites Neural Operators [LKA+20b] and Physics Informed Neural Networks (PINNs) [RPK19] into a single model.

TFormer is a transformer-based model that applies attention on the spatial domain instead of the temporal domain (similar to Galerkin-Transfomer [Cao21] and OFormer [LMF23]) which endows the model with spatial zero-shot super-resolution capabilities by design. Furthermore, the model uses the target prediction time t as an additional input to achieve temporal zero-shot super-resolution. Our proposed model learns the spatial and temporal dependencies of the solution u and takes as input the initial condition $u(0, X) = (u(0, x_1), \ldots, u(0, x_s))^T$, the discretization grid $X = ((x_i)_{i=1}^s)^T$, PDE Parameters $\boldsymbol{p} = (p_1, \ldots, p_j)^T \in \mathbb{R}^j$ with j number of scalar PDE parameters, and the prediction time t and directly regresses the solution $u(t, X) = (u(t, x_1), \ldots, u(t, x_s))^T$ at time t. As a consequence of this design, there is no need to unroll the entire PDE trajectory autoregressively. If the entire trajectory is needed, the TFormer can be queried with the desired times along the trajectory to get the entire trajectory. TFormer for 1D PDEs can be expressed as a function

$$f_\theta : (\mathbb{R}^{s \times c} \times \mathbb{R}^s \times \mathbb{R} \times \mathbb{R}^j) \to \mathbb{R}^{s \times c} \text{ with } (u(0, X), X, t, \boldsymbol{p}) \mapsto u(t, X) \tag{4.7}$$

where θ represents the weights and biases of the neural network, $u(0, X)$ the initial condition, $X \in \mathbb{R}^s$ the grid with spatial coordinates, and $\boldsymbol{p}$ the vector of PDE parameters. Figure 4.2 illustrates the principle of the TFormer.

4.5.1 Model Architecture

TFormer consists of a mechanism for generating the latent representation of the initial condition and the queried time t, Linear Transformer blocks as well as blocks for spatio-temporal conditioning and a mechanism to map the solution's latent representation back to the physical space.

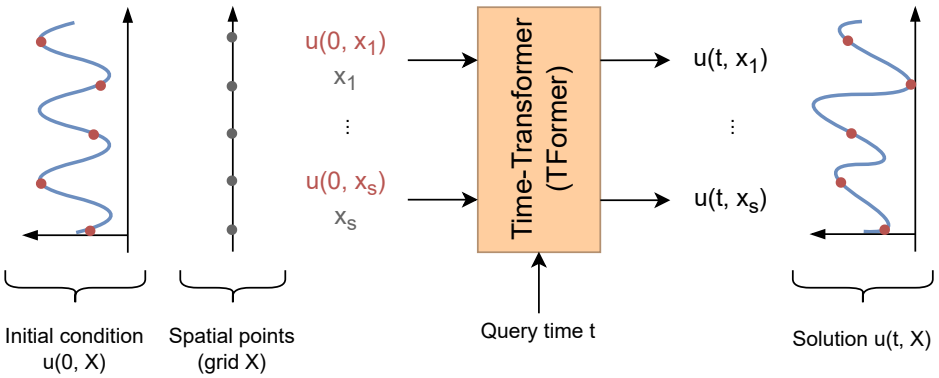


Figure 4.10: Abstract architecture of TFormer for solving 1D PDEs. The model takes the initial condition $u(0, X)$, grid X, PDE Parameters $\boldsymbol{p}$ (omitted for clarity in the illustration) and the query time t as input and directly outputs $u(t, X)$ without unrolling the entire trajectory.

First of all, the physical representation of the initial condition is mapped to a latent representation, by applying a linear layer. Each vector in the latent space represents a spatial solution point $u(0, \boldsymbol{x_i}) \in \mathbb{R}^c$. Linear Transformer blocks with Self-Attention are used to generate an attention-refined latent representation for the input solution points. We call the output of the Transformer blocks *initial condition latent representation*. On the other hand, second embeddings representing the query time t and the grid points $\boldsymbol{x_i}$ are generated. This representation is called *spatio-temporal latent representation*.

The *initial condition latent representation* and the *spatio-temporal latent representation* are fed into spatio-temporal conditioning blocks. We take inspiration from Transformer blocks with Cross-Attention and replace the Cross-Attention in the original Transformer blocks with a non-linearity σ and a simple pointwise multiplication (Hadamard product) since this block only needs to condition the latent representations of the initial condition with the prediction time which doesn't require Cross-Attention. The output latent representation is processed by a shared multi-layer perceptron. We also keep the residual connections and layer normalizations (Add & Norm) of the original Transformer with Cross-Attention. At the end, an MLP is applied to map the solution's latent representation back to physical space. We implement the model in a way, that it can calculate the solution of different times t in parallel to get a speed-up for training and inference. Figure 4.11 shows the detailed architecture of TFormer.

Latent Representation

As a first step, the input initial condition is mapped to a latent representation (embedding). Each solution point $u(0, x_i)$ from $u(t, X) = (u(t, x_1), \ldots, u(t, x_s))^T$ will be interpreted as a token. Thus, a vector in the latent space represents a solution point $u(0, x_i)$. A simple linear layer is applied to each solution point to generate an embedding of dimension $d = 96$. The linear layer is shared across all spatial points x_i. Figure 4.12a shows the mechanisms to generate latent representations for a 1D PDE with one single channel.

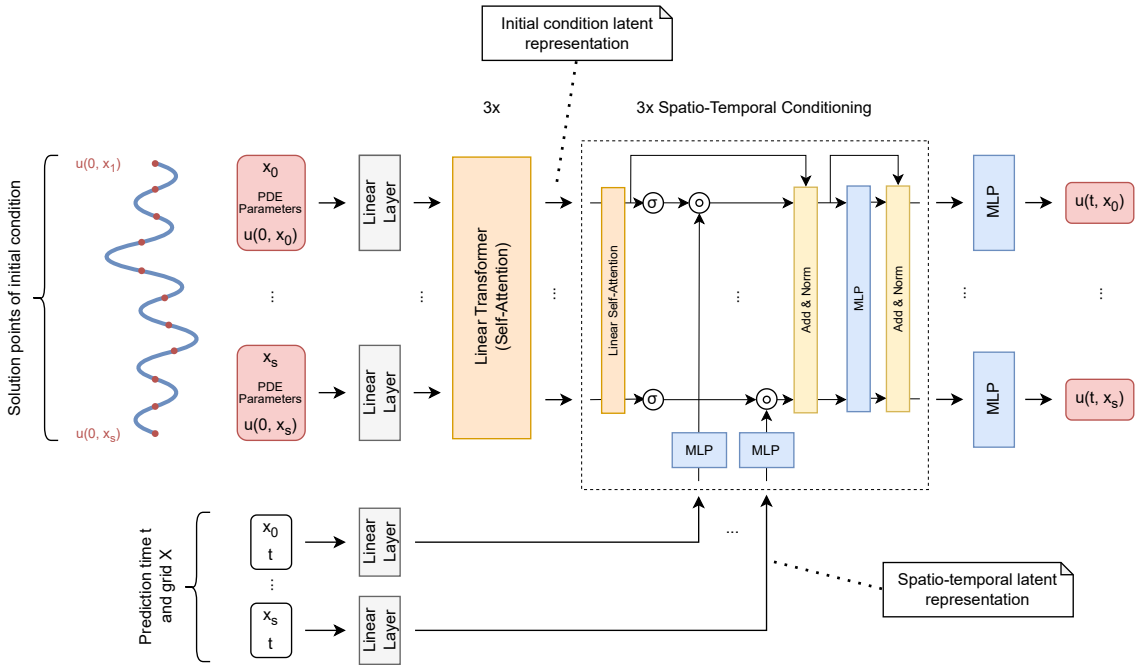
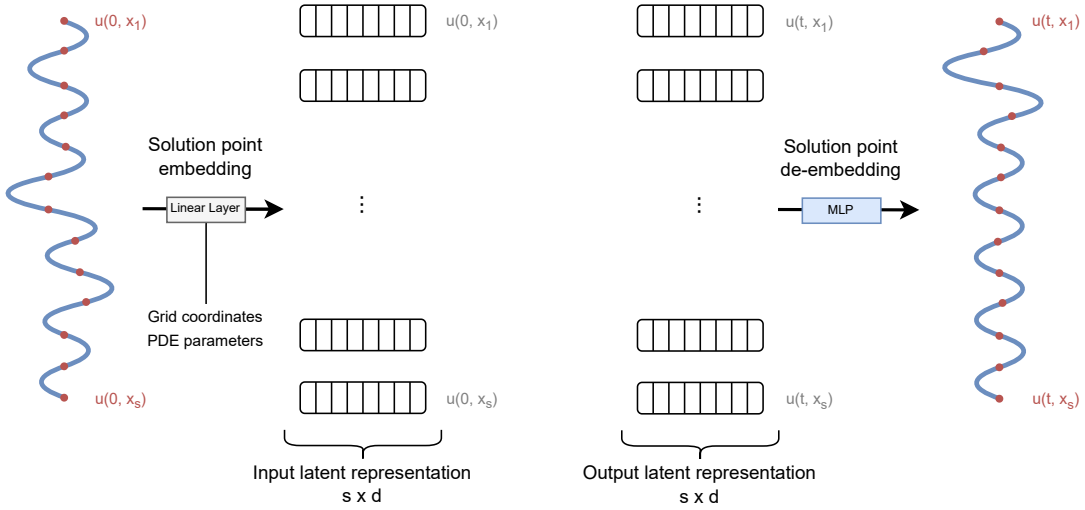


Figure 4.11: Architecture of TFormer for solving 1D PDEs. The spatio-temporal conditioning (dashed rectangle) is inspired by a Transformer block with Cross-Attention. We replace the Cross-Attention with a non-linearity σ and a simple pointwise multiplication $\circ$. The multi-layer-perceptron and Add & Norm are the same as in a Vanilla or Linear Transformer with Cross-Attention.



(a) Generating the latent representation

(b) Mapping back to the physical space

Figure 4.12: Figure 4.12a shows the mechanism to map the initial condition to a latent representation. Each solution point $u(0, x_i)$ is mapped to a latent representation. Figure 4.12b shows the mechanism to map the latent representation of $u(t, X)$ back to the physical space.

PDE Parameter Embedding The PDE parameters $\boldsymbol{p}$ are used as an additional input for the linear layer that generates the latent representation of the initial condition.

Positional Encoding The grid X contains the coordinates where the solutions were sampled in the spatial domain. This information is used when generating the latent representations of the initial condition to ensure that each latent representation has information about the position. We omit additional positional encoding to avoid the model suffering from the Transformer length generalization problem which could hinder spatial zero-shot super-resolution capabilities.

Linear Transformer

We utilize Linear Transformer blocks with Self-Attention in our TFormer architecture to generate an attention-refined latent representation of the input latent representation (embedding). The global receptive field of the Transformer allows the proposed architecture to capture spatial dependencies of the initial condition, although each token contains only partial 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. It can be compared to translation tasks in Natural Language Processing (NLP) where a Transformer with Self-Attention is used as an encoder to generate more meaningful latent representations for each input word by incorporating the entire input text and not only a single word.

Spatio-Temporal Embedding

The query time t determines the time of the model's prediction (i.e., $u(t, X)$). To condition the initial condition latent representations on the time, we need to map $t \in \mathbb{R}_+$ to an embedding vector of dimensionality $d = 96$. In addition to the query time t, we add the spatial coordinates x_i to the embedding to allow conditioning of the initial condition with spatio-temporal information. For 1D PDEs, a simple linear layer is used to generate the embedding.

Spatio-Temporal Conditioning

The components of the spatio-temporal conditioning blocks are similar to a Transformer block with Cross-Attention, but we suggest replacing Cross-Attention with a conditioning mechanism similar to Feature-wise Linear Modulation (FiLM) proposed by Perez et al. [PSV+17]. Our conditioning mechanism uses only the scaling (i.e., pointwise multiplication) of the FiLM and omits the shift (i.e., pointwise addition). A spatio-temporal conditioning block in the proposed TFormer is mathematically expressed by

$$\begin{aligned} \text{Spatio-temporal-conditioning}(I, T) &= \sigma(\text{Self-Attention}(I)) \circ \text{MLP}(T) \\ \sigma(X) &= ELU(X) + 1 \end{aligned} \tag{4.8}$$

where $I \in \mathbb{R}^{s \times d}$ represents the initial condition embedding generated by the Linear Transformer and $T \in \mathbb{R}^{s \times d}$ denotes the spatio-temporal embedding generated by a simple linear layer. $\circ$ represents the Hadamard product or pointwise matrix multiplication and σ represents a non-linearity. The residual connections and layer normalizations (Add & Norm) as well as the MLP are omitted from the equation for the sake of simplicity.

Why Replace Cross-Attention with a Conditioning Mechanism? We propose replacing Cross-Attention in the spatio-temporal conditioning blocks with a non-linearity and a pointwise multiplication (Hadamard product) since each initial condition latent representation (generated by the Linear Transformer) should be individually conditioned with the corresponding spatio-temporal latent embedding (embedding of prediction time and spatial coordinate). Applying Cross-Attention with K and V from the initial condition embedding and Q from the temporal embedding would mix different initial condition latent representations by calculating the weighted sum, but the spatio-temporal conditioning should condition each component of each initial condition embedding separately. Additionally, the Linear Self-Attention before the conditioning allows the model to capture spatial dependencies. Thus, we assume that Cross-Attention with K, V from the initial condition embedding and Q from the spatio-temporal embedding is not necessary. Using Cross-Attention with K and V from the spatio-temporal embedding and Q from the initial condition embedding would yield a weighted sum of the temporal embeddings which is also not desired since the output should be the modified initial condition embeddings and not the spatio-temporal embeddings.

Map Latent Space back to the Physical Space

In the end, the solution's latent representation must be mapped back to physical space. Similar to the generation of the latent representation, a simple multi-layer perceptron is applied to map the solution back to the physical space. The mechanism is shown in Figure 4.12b.

4.5.2 Linear Attention or Vanilla Attention?

The proposed TFormer architecture (Figure 4.11) uses Linear Transformer blocks and Linear Attention in the spatio-temporal conditioning. In theory, the Linear Transformer can be replaced with a Vanilla Transformer and the Linear Attention with Vanilla Attention. Nonetheless, we recommend using Linear Attention instead of Vanilla Attention since it promises to be more memory and time-efficient. Table 4.1 shows the linear memory increase and the quadratic memory increase of Linear Attention and Vanilla Attention, respectively. Double the spatial resolution corresponds to double the number of tokens yielding an increased memory and time consumption. Training TFormer with Vanilla Attention requires more than 640 GiB while TFormer with Linear Attention requires only 99.4 GiB. We are using a non-optimized, pure PyTorch implementation of Linear and Vanilla Attention. The results demonstrate that it is not feasible to train the TFormer with a pure PyTorch implementation of Vanilla Attention, whereas training the TFormer with Linear Attention is easily possible.

Spatial resolution (# tokens)	Attention type	GPU memory	Time per epoch
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 4.1: GPU memory consumption and training time per epoch for TFormer 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. Time per epoch includes the time that is needed to load the data and transfer it to the GPUs. We are using a non-optimized, pure PyTorch implementation of Linear and Vanilla Attention.

4.5.3 Accerlerated Training and Inference

We accelerate the training and inference of the TFormer by processing multiple timesteps in parallel on the GPU. If the solution of multiple timesteps t (e.g., $t \in \{t_1, t_2, \ldots, t_N\}$) is to be predicted, TFormer can calculate the solution of each timestep in parallel due to the fact, that the predictions of $u(t, X)$ are independent of each other. We improve the calculations done in the proposed spatio-temporal conditioning blocks to process multiple timesteps in parallel. To do so we modified the calculation in the Linear Attention mechanism to take an additional dimension of the queried timesteps to calculate the attention of the timesteps in parallel on the GPU. We also modify the conditioning in the spatio-temporal conditioning to process the timesteps in parallel. Sequentially training the model would take too long and the parallel implementation provides some significant speed-up.

In contrast to the previously introduced architecture (Recurrent Linear Transformer and Vanilla Transformer for PDEs as well as PDE FNet), TFormer can directly predict the solution at an arbitrary timestep t without the need to unroll the entire trajectory until timestep t. Figure 4.13 illustrates this process. If the whole trajectory $(u(0, X), u(t_1, X), \ldots, u(t_N, X))$ is required, TFormer can be queried with all timesteps at once and calculates the predictions at once since there is no dependency between the timesteps (i.e., the model only needs the initial condition to calculate the solution at timestep t). The parallel implementation promises a speed-up when predicting a trajectory.

4.5.4 Spatial and Temporal Zero-Shot Super-Resolution

During training, the model is trained on a lower and fixed spatial and temporal resolution (fixed discretization). After the training, the spatial and temporal can be increased to get a higher resolution without retraining the model. Doing high-resolution temporal inference can be achieved by querying the model with corresponding times t (i.e., intermediate times t). Training on low-resolution data requires less computational resources and saves computing time.

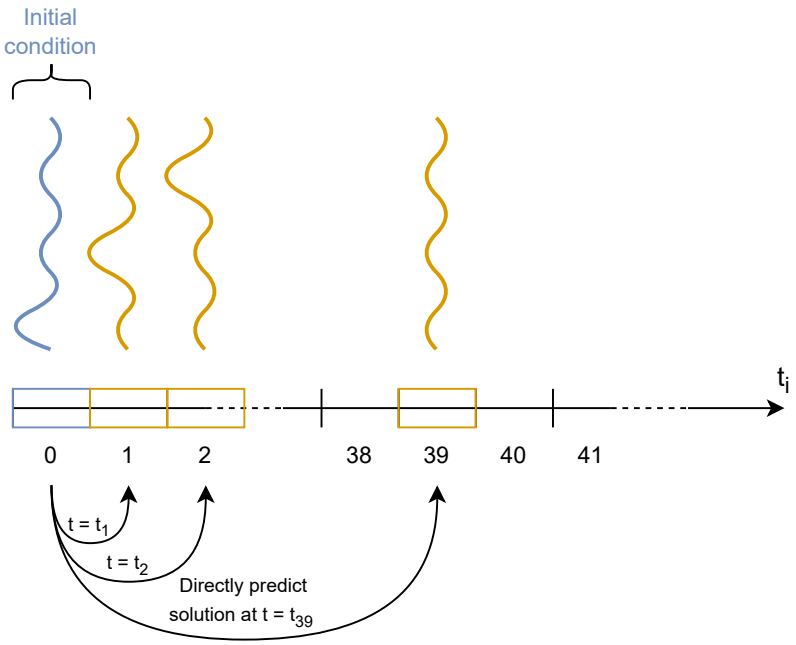


Figure 4.13: TFormer can directly predict the solution $u(t, X)$ without unrolling the entire trajectory until time t. Blue denotes the initial condition and orange the predictions.

4.6 Predicting Long Rollouts

Usually, a model for solving PDEs is trained to predict a fixed number of time steps (e.g., 40 timesteps), but after that, the model might be used to predict more than 40 timesteps (e.g., prediction of timestep t_{41}). In this section, we investigate the model's ability to do longer rollouts. We use the terms longer rollouts and extrapolation interchangeably and refer to that a model predicts more future timesteps than seen during training (i.e., go beyond the times seen during training).

4.6.1 Recurrent Linear Transformer and Vanilla Transformer

The predictions of the Recurrent Linear Transformer and Vanilla Transformer are done in an autoregressive fashion by appending previous predictions to the model's input. Thus, the input sequence length or context size increases with each prediction. Since positional encoding is applied to the sequence, the models suffer from the Transformer sequence length generalization problem. For instance, to predict timestep t_{41} our model would take timesteps $t_0, \ldots, t_{41}$ as input sequence which is longer than the sequences seen during training. Consequently, the model will produce poor predictions. Unfortunately, techniques such as random positional encoding of Ruoss et al. [RDG+23] perform poorly for our model. Therefore, we propose a sliding window approach for Transformers that enables the model to do longer rollouts than done at training.

Sliding Window Approach

Figure 4.14 shows our proposed sliding window approach. In the first step, predictions will be made until the maximum sequence length is reached (i.e., the context window is filled). The predictions are done by appending the previous prediction to the input sequence and increasing the context window step by step. If the maximum input sequence length is reached, the context window will be moved to use only the latest 10 timesteps as a new input context. Moving the context window to the right is equivalent to dropping the latest 30 timesteps to free space for the next 30 predictions. After that, the model predicts the next 30 timesteps and then, the window will be moved again. Mathematically, it can be represented as

$$\begin{aligned}
\text{1st prediction: } & f_\theta((u(t_0, X)) = \widehat{u}(t_1, X) \\
\text{2nd prediction: } & f_\theta((u(t_0, X), \widehat{u}(t_1, X))) = \widehat{u}(t_2, X) \\
& \vdots \\
\text{40th prediction: } & f_\theta((u(t_0, X), \ldots, \widehat{u}(t_{30}, X), \widehat{u}(t_{31}, X), \ldots, \widehat{u}(t_{39}, X))) = \widehat{u}(t_{40}, X) \\
\text{41st prediction: } & \text{Maximum context size reached } \Longrightarrow \text{ Window movement} \\
& f_\theta((\cancel{u(t_0, X), \ldots, \widehat{u}(t_{30}, X),} \widehat{u}(t_{31}, X), \ldots, \widehat{u}(t_{40}, X))) = \widehat{u}(t_{41}, X) \\
& f_\theta(((\widehat{u}(t_{31}, X), \ldots, \widehat{u}(t_{40}, X))) = \widehat{u}(t_{41}, X) \\
\text{42nd prediction: } & f_\theta(((\widehat{u}(t_{31}, X), \ldots, \widehat{u}(t_{40}, X), \widehat{u}(t_{41}, X))) = \widehat{u}(t_{42}, X)
\end{aligned} \tag{4.9}$$

where f_θ denotes the model, $u(t_0, X)$ the initial condition, and $\widehat{u}(\cdot, X)$ the solution predicted by the model. The window is moved for the 41st prediction (extrapolation regime) to free space for the next predictions. The number of movements or slides for arbitrary context sizes and overlaps can be calculated with

$$\text{slides} = \begin{cases} 0 & \text{if (context\_size} - \text{\#ICs)} \geq \text{prediction\_length} \\ \left\lceil \frac{\text{prediction\_length} - \text{\#ICs}}{\text{context\_size} - \text{overlap}} \right\rceil & \text{otherwise} \end{cases} \tag{4.10}$$

where prediction_size denotes the number of timesteps to be predicted, #ICs refers to the number of timesteps used as the initial condition, context_size denotes the length of the model's context window and overlap describes the number of previous predictions that will be used after a window slide.

The ability of the model to handle different initial condition sizes allows using the 10 latest timesteps as the new initial condition after moving the window instead of using only the latest prediction as input. Providing the model more historical context instead of only the latest timestep or prediction should improve the accuracy.

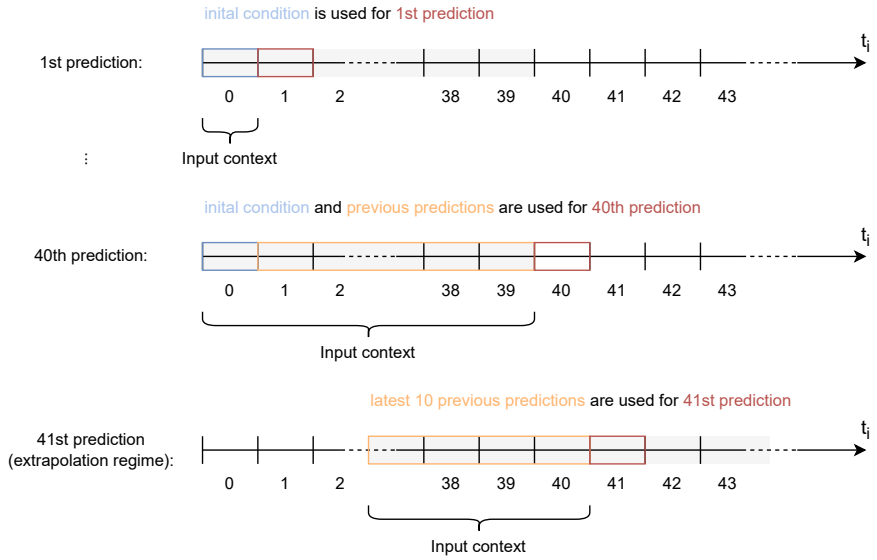


Figure 4.14: Sliding window approach for Transformers. The context window of the model is fixed due to the Transformer's length generalization problem (here: max. 40 time steps). The grey rectangle represents the context window. In the beginning, the model uses one timestep as the initial condition to predict the next timestep. Then it uses the initial condition and the previous prediction as input and so on. For the 41st prediction, the context window length would be 41 if we continue normally. To avoid having a context length greater than seen during training, we move the entire context window (grey rectangle). When doing the window movement, the model uses the latest 10 predictions as new input instead of the initial condition and all previous timesteps or the latest timestep. This should help the model because it has 10 steps as context and not only one, noisy timestep.

4.6.2 PDE FNet

PDE FNet is also used in an autoregressive fashion. Since the Fourier transform retains the order of the timesteps, no additional positional encoding is applied which implies that the model doesn't suffer from the Transformer length generalization problem caused by the positional encoding. This allows the model to make predictions that are longer than seen during training without our proposed sliding window approach or other techniques.

4.6.3 TFormer

TFormer can predict the solution $u(t, X)$ for arbitrary times t by getting queried with the corresponding times t. To do longer rollouts than seen during training, the model could be queried with times t that are greater than seen during training. But we expect that this approach will lead to bad predictions since t gets out-of-distribution. A suitable fix would be using an autoregressive approach by using the latest prediction, for which t is in-distribution, as new input and so on.

5 Baseline Models

This section introduces the chosen state-of-the-art baseline or reference models and briefly explains how they operate. The first model is Fourier Neural Operator (FNO) introduced by Li et al. [LKA+20b], the other two models are transformer-based, namely Galerkin Transformer proposed by Cao [Cao21] and Operator Transformer of Li et al. [LMF23]. We explain the models for 1D PDEs only since we focus on 1D PDEs in this work.

5.1 Fourier Neural Operator (FNO)

Fourier Neural Operator [LKA+20b] is an implementation of a Neural Operator (see Section 2.2.2) that maps from one function $a(x)$ to another function $a'(x)$. In our case, function $a(x) := u(t_n, \boldsymbol{x})$ represents the solution for timestep t_n and $a'(x) := u(t_{n+1}, \boldsymbol{x})$ the solution for a future timestep t_{n+1}. Both functions $a(x)$ and $a'(x)$ are discretized and represented as a finite sequence in $\mathbb{R}^{s \times c}$ with s the spatial resolution and c the number of channels. However, the discretization can be arbitrary which implies that FNO is independent of the spatial discretization $X \in \mathbb{R}^s$. Figure 5.1 shows the architecture of the Fourier Neural Operator.

Lifting First, the input function $a(x)$ is lifted to a higher dimensional space by applying a pointwise linear layer. We are using a hidden representation with dimension $d = 64$ in our experiments. The lifting is identical to the generation of the input latent representation of the proposed TFormer. Mathematically, it can be expressed as

$$v_0(x) = Wa(x) \qquad \forall x \in \mathbb{X} \tag{5.1}$$

with $W : \mathbb{R}^c \to \mathbb{R}^d$ a linear transformation that can be implemented with a pointwise linear layer.

Iterative Updates As a second step, the hidden representation is iteratively updated from $v_0 \mapsto \ldots \mapsto v_{l+1}$ by applying spectral convolution layers which are defined as

$$v_{l+1}(x) = \sigma\big(Wv_l(x) + \big(\mathcal{K}_\theta(a)v_l\big)(x)\big) \qquad \forall x \in \mathbb{X} \tag{5.2}$$

where $l = 0, 1, \ldots$ denotes the layer of the neural network, $\sigma : \mathbb{R} \to \mathbb{R}$ is a pointwise (local) non-linearity, and $W : \mathbb{R}^d \to \mathbb{R}^d$ a linear transformation. $\mathcal{K}_\theta(a)$ denotes a non-local integral transformation. This integral transformation is beneficial for solving PDEs and is not an arbitrary choice. Following Li et al. [LKA+20a] and considering a PDE

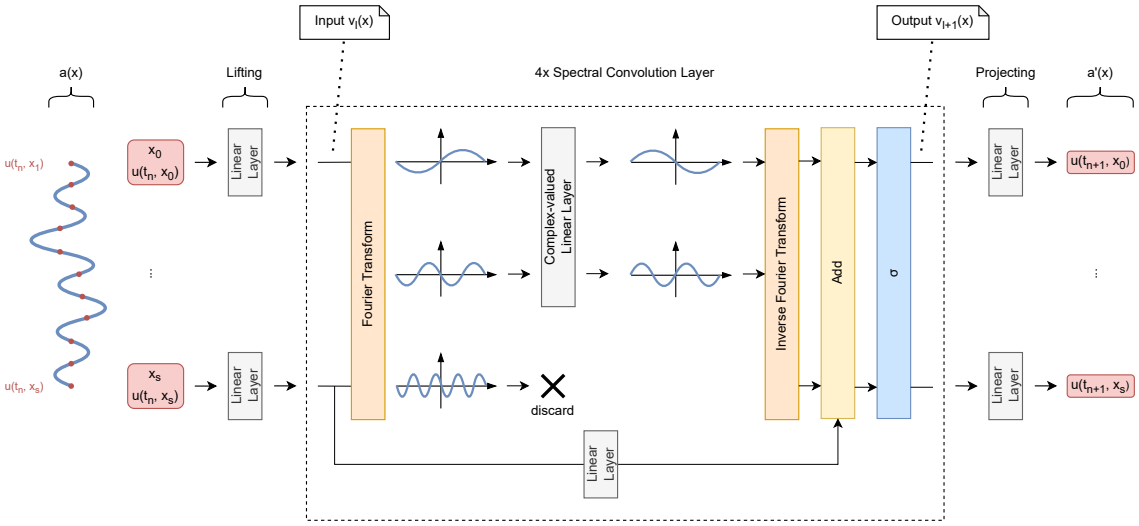


Figure 5.1: Architecture of Fourier Neural Operator. The input of the model is a function $a(x) = u(t_n, \boldsymbol{x})$ which represents the solution for timestep t_n. First, the function is *lifted* to a higher dimension by applying a pointwise linear layer. Four spectral convolution layers are iteratively applied to update the hidden representation. The layers map the input function $v_l(x)$ into Fourier space and transform the first k_{max} Fourier modes (complex-valued linear layer). An inverse Fourier transform is applied to map the function from Fourier space back to the function $v_{l+1}(x)$ in the physical space. In the end, $v_4(x)$ is mapped back to physical space with a pointwise linear layer yielding $a'(x) = u(t_{n+1}, \boldsymbol{x})$ depicting the solution for timestep t_{n+1}.

$$\begin{aligned} (\mathcal{L}_a)u(x) &= f(x) \text{ with } x \in \mathbb{X} \\ B[u](t,x) &= 0 \text{ with } x \in \partial\mathbb{X} \end{aligned} \tag{5.3}$$

where $\mathcal{L}_a$ represents a differential operator (e.g., the partial differential operator that maps a function to its partial derivation) that depends on the initial condition a. Then, the solution of the PDE can be represented as

$$u(x) = \int_{\mathbb{X}} G_a(x,y) f(y) dy \tag{5.4}$$

where G denotes the Green's function that depends on the initial condition. Therefore, they define the integral transformation in their model as

$$\big(\mathcal{K}_\theta(a)v_l\big)(x) = \int_{\mathbb{X}} \kappa_\theta\big(x, y, a(x), a(y)\big) v_l(y) dy \qquad \forall x \in \mathbb{X} \tag{5.5}$$

where κ_θ denotes a neural network with parameters θ (i.e., weights and biases). They propose removing the dependency of a on $\kappa_\theta\big(x, y, a(x), a(y)\big)$ and imposing $\kappa_\theta(x,y) = \kappa_\theta(x-y)$. This leads to

$$\big(\mathcal{K}_\theta(a)v_l\big)(x) = \int_{\mathbb{X}} \kappa_\theta(x-y)v_l(y)dy \qquad \forall x \in \mathbb{X} \tag{5.6}$$

which is simply a convolution of the functions v_t and κ_θ. Since convolution is equivalent to multiplication in Fourier space, the equation can be reformulated using the Fourier transform $\mathcal{F}$ and its inverse $\mathcal{F}^{-1}$:

$$\big(\mathcal{K}_\theta(a)v_l\big)(x) = \int_{\mathbb{X}} \kappa_\theta(x-y)v_l(y)dy = \mathcal{F}^{-1}\big(\mathcal{F}(\kappa_\theta)\cdot\mathcal{F}(v_t)\big)(x) \qquad \forall x \in \mathbb{X} \tag{5.7}$$

As mentioned earlier, the functions are discretized. Thus, the continuous Fourier transform can be replaced with a discrete Fourier transform yielding to

$$\mathcal{K}_\theta(a)v_l = IDFT\big(DFT(\kappa_\theta)\cdot DFT(v_t)\big) \tag{5.8}$$

They further propose to directly parametrize κ_θ in Fourier space to get

$$\mathcal{K}_\theta(a)v_l = IDFT\big(R_\theta \cdot DFT(v_t)\big) \tag{5.9}$$

where $R_\theta \in \mathbb{C}^{s\times d\times d}$ denotes the transformations of the frequencies components or modes. R_θ is a learnable weight matrix that depends on the spatial discretization. Therefore, they choose a finite-dimensional parametrization by keeping k_{max} modes and truncating higher modes. Thus, the integral transformation $\mathcal{K}_\theta(a)v_l$ is parametrized by the finite-dimensional complex-valued tensor $R_\theta \in \mathbb{C}^{k_{max}\times d\times d}$ (e.g., a complex-valued linear layer). We are using $k_{max} = 16$ modes for our experiments.

Projection In the end, the final hidden representation v_{l+1} is projected back to the physical space of the output function $a'(x) = u(t_{n+1}, \boldsymbol{x})$ that represents the solution for timestep t_{n+1}.

Residual Connections (Add) They add residual connections, with a pointwise linear layer in the residual path, to the spectral convolution layers.

5.2 Fourier and Galerkin Transformer

Galerkin Transformer proposed by Cao [Cao21] is an attention-based Neural Operator. Consequently, the model takes as input a discretized function $u(t_n, \boldsymbol{x})$ and maps it to an output function $u(t_{n+1}, \boldsymbol{x})$. Fourier and Galerkin Transformer essentially applies spatial Self-Attention and consists of a pointwise feature extractor, Self-Attention encoder, and a decoder. The entire architecture of Fourier and Galerkin Transformer is shown in Figure 5.2.

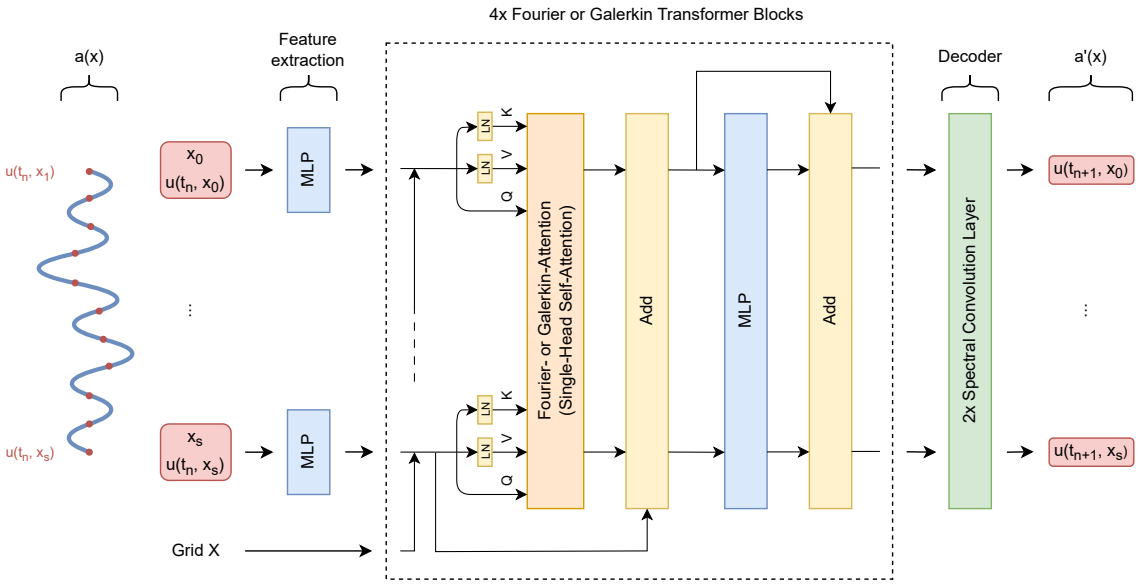


Figure 5.2: Architecture of Galerkin (or Fourier) Transformer. The input of the model is a function $a(x) = u(t_n, \boldsymbol{x})$ which represents the solution for timestep t_n. First, a pointwise MLP is applied to generate a latent representation (feature extraction). Four Galerkin (or Fourier) Transformer blocks are applied to update the hidden representation. In the end, the solution's latent representation is mapped back to physical space with spectral convolution layers.

Pointwise Feature Extractor An MLP is applied to each solution point $u(t_n, x_i)$ of $u(t_n, X)$ to extract features. Thus, it is similar to the lifting done by the FNO or the generation of the latent representation of the TFormer. They concatenate the coordinates of the grid to each solution point's latent representation to add positional information to the tokens and omit additional positional encoding such as absolute positional encoding of Vaswani et al. [VSP+17]. Furthermore, the grid coordinates are added to each latent representation before applying Self-Attention.

Self-Attention Encoder An encoder that consists of multiple Transformer blocks with Self-Attention is applied to the input latent representation to update it iteratively. Usually, the matrices Q, K, V are interpreted row-wise, meaning that q_i, k_i, v_i represent the query, key, and value of the i-th token in the sequence, respectively. Cao [Cao21] interprets the matrices column-wise as the evaluation of learned basis-functions q_j, k_j, v_j for the j-th column or basis function. Thus, the columns j represent the learned basis functions. For example, an entry $q_{i,j}$ is the evaluation of the basis function j on the spatial point i. Similar for the matrices K and V.

Row-wise interpretation of the Query matrix Q: Each row i represents a token in the latent space with $d = 4$.

$$Q = \begin{array}{c} \\ i{=}1 \\ i{=}2 \\ i{=}3 \end{array} \begin{array}{c} \begin{array}{cccc} j{=}1 & j{=}2 & j{=}3 & j{=}4 \end{array} \\ \begin{pmatrix} 0.2 & 0.6 & 0.6 & 0.1 \\ 0.7 & 0.8 & 0.3 & 0.4 \\ 0.2 & 0.3 & 0.2 & 0.3 \end{pmatrix} \end{array} \in \mathbb{R}^{N \times d}$$

Column-wise interpretation of the Query matrix Q: Each column j represents a learnable basis function.

$$Q = \begin{array}{c} \\ i{=}1 \\ i{=}2 \\ i{=}3 \end{array} \begin{array}{c} \begin{array}{cccc} j{=}1 & j{=}2 & j{=}3 & j{=}4 \end{array} \\ \begin{pmatrix} 0.2 & 0.6 & 0.6 & 0.1 \\ 0.7 & 0.8 & 0.3 & 0.4 \\ 0.2 & 0.3 & 0.2 & 0.3 \end{pmatrix} \end{array} \in \mathbb{R}^{N \times d}$$

Figure 5.3: Row-wise and column-wise interpretations of the Query matrix Q. The matrices K and V are similarly interpreted row-wise or column-wise.

The column-wise interpretation allows the formulation of two softmax-free attention mechanisms

$$\text{Fourier-Attn}(Q, K, V) = \frac{(QK^T)V}{N} \qquad \text{Galerkin-Attn}(Q, K, V) = \frac{Q(K^TV)}{N} \tag{5.10}$$

where N denotes the number of tokens which is equivalent to the spatial resolution s. Fourier-Attention has a quadratic complexity and Galerkin-Attention has a linear complexity. Both attention mechanisms have an interpretation as numerical quadrature by applying the Monte Carlo integral approximation. Analogous to FNO, the integral approximation is beneficial since the solution of a PDE can be represented as an integral transformation. An integral could be approximated by randomly sampling points $\{\xi_1, \ldots, \xi_m\}$, evaluating the function on the sampled points ξ_i and calculating the mean value of the function values:

$$\int_a^b f(\xi) d\xi \approx \frac{b-a}{m} \sum_{j=0}^{m} f(\xi_j) \tag{5.11}$$

Reformulating the attention calculations from Equation (5.10) and applying the Monte Carlo (MC) integral approximation and assuming $(b - a) = 1$ according to Kovachki et al. [KLL+21] yields

$$\begin{aligned} \text{Fourier:} \quad & \frac{(QK^T)V}{N} = Z \text{ and } z_{i,j} = \frac{1}{N} \sum_{l=1}^{N} (\mathbf{q}_i \cdot \mathbf{k}_l) v_{l,j} \overset{MC}{\approx} \int_{\mathbb{X}} \kappa(x_i, \xi) v_j(\xi) d\xi \\ \text{Galerkin:} \quad & \frac{Q(K^TV)}{N} = Z \text{ and } z_{i,j} = \sum_{l=1}^{d} \frac{(\mathbf{k}_l^T \cdot \mathbf{v}_j^T)}{N} q_{i,l} \overset{MC}{\approx} \sum_{l=1}^{d} \left(\int_{\mathbb{X}} k_l(\xi) v_j(\xi) d\xi \right) q_l(x_i) \end{aligned} \tag{5.12}$$

where $(\cdot)_j$ denotes the j-th row of a matrix. $\kappa(x_i, \xi)$ denotes a learnable kernel function that is approximated by $\mathbf{q}_i \cdot \mathbf{k}_l$ and $v_j(\cdot)$ is the j-th basis function. The similarity between Fourier-Attention and the spectral convolution layer proposed by Li et al. [LKA+20b] is responsible for the name "Fourier-Attention". Cao [Cao21] keeps the residual connections and the MLP of the Vanilla Transformer of Vaswani et al. [VSP+17] and proposes to replace the layer normalization (post-layer normalization) with a pre-layer normalization for the key K and value V matrix to allow the scaling of the values to propagate through the encoder layers.

Decoder The decoder maps the latent representation back to the physical space. Depending on the problem, he proposes using spectral convolution layers of FNO [LKA+20b] for regular and smooth target solutions or a pointwise MLP for non-smooth target solutions.

5.3 Operator Transformer (OFormer)

Operator Transformer, short OFormer, proposed by Li et al. [LMF23] is a Neural Operator based on the Galerkin and Fourier Transformer. FNO as well as Fourier and Galerkin Transformer have the issue that the output spatial resolution depends on the input spatial resolution and that the spatial coordinates of the output are the same as for the input. However, having different spatial resolutions for the input and output as well as querying arbitrary spatial points would be desirable. OFormer solves this issue by decoupling the input spatial domain from the output spatial domain by adding Cross-Attention based on Fourier and Galerkin Attention to the architecture. Thus, the input grid X and output grid X' are independent which allows querying for arbitrary spatial points x_i. They also propose adding additional positional encoding (Rotary Positional Embedding of Su et al. [SLP+23]) to the latent representations and dropping all layer normalizations to enable the scaling to propagate through the model. In addition, they suggest using an MLP to propagate the dynamics in the latent space. This MLP can applied iteratively to evolve the solution for multiple timesteps. Figure 5.4 illustrates the architecture of OFormer.

Latent Representation The latent representation of the input solution $u(t_n, X)$ is generated by utilizing a pointwise MLP. The spatial coordinates of the grid X are added to each solution point to add positional information to the tokens. Consequently, the generation of the latent representation is similar to the Fourier and Galerkin Transformer as well as the proposed TFormer. On the other hand, OFormer creates a latent representation for the query spatial points X' by also applying a pointwise MLP.

Encoder The transformer-based encoder takes the input solution latent representation and the query point latent representation as input. The solution latent representation is updated by several Transformer blocks with Galerkin or Fourier Self-Attention which is identical to the Fourier and Galerkin Transformer. However, OFormer drops the pre-layer normalization of V, Q and OFormer applies additional positional encoding (Rotary Positional Embedding from Su et al. [SLP+23]). The query point latent representation is used to "query" the input solution latent representation with the desired spatial locations (i.e., the query points are used as query Q and the solution latent representation as key K and value V for the Galerkin- or Fourier-based Cross-Attention). The final output of the encoder is the latent representation of the solutions for the queried spatial points.

Latent Propagator A shared MLP processes the output of the transformer-based encoder to propagate the dynamics of the PDE. It can be applied multiple times to propagate the dynamics in the latent space.

Decoder The solution's latent representation is mapped back to the physical space by applying a pointwise MLP.

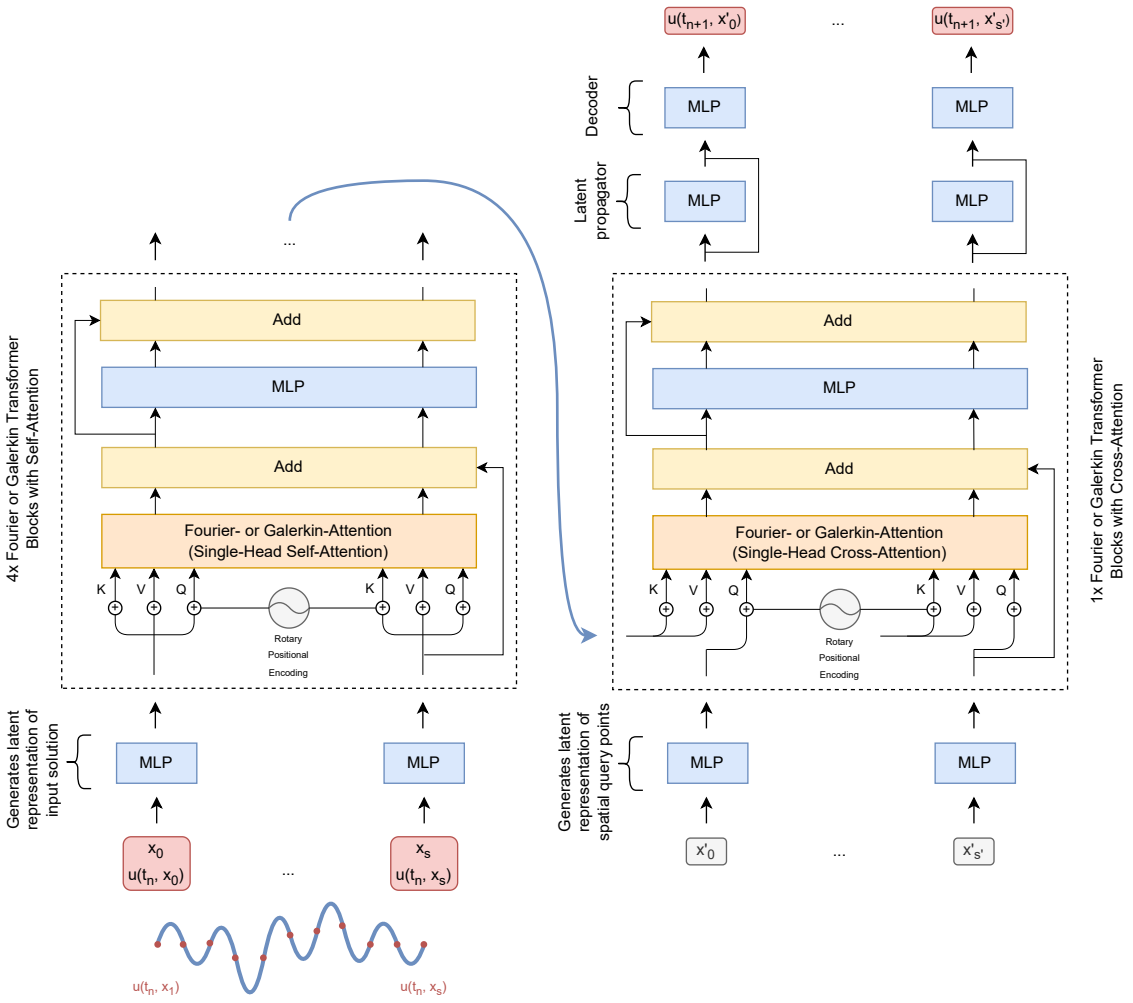


Figure 5.4: Architecture of OFormer. The input of the model is the solution $u(t_n, X)$ at timestep t_n and the query spatial points X'. The solution's latent representation is updated iteratively by Transformer blocks with Self-Attention, similar to the Fourier and Galerkin Transformer. Cross-attention is utilized to merge the input solution's latent representation with the query point's latent representation. The output of the model is the solution $u(t_{n+1}, X')$ at timestep t_{n+1} at the queried spatial points X'.

5.4 cFNO and cOFormer

The models introduced above do not consider the PDE parameters. Takamoto et al. [TAN23] propose a parameter-conditioned Fourier Neural Operator (cFNO) and add the PDE parameter $\boldsymbol{p} \in \mathbb{R}^j$ to the model by creating additional input channels for the PDE parameters. We utilize the same mechanism to add the PDE parameter to OFormer, yielding cOFormer.

6 Architecture Overview

The goal of this section is to give a short overview of the introduced architectures and their contributions. Furthermore, we compare our models with the chosen state-of-the-art baseline models.

We introduced two base architectures (Transformers for solving PDEs by emulating numerical PDE solver and Transformers for solving PDEs by learning the underlying function) and three models that implement the first base architecture (Reccurent Linear Transformer and Vanilla Transformer for PDEs as well as PDE FNet) and a model that implements the last base architecture (TFormer). The four models are:

- Recurrent Linear Transformer and Vanilla Transformer for PDEs: Uses a Recurrent Linear Transformer or Vanilla Transformer to capture temporal dependencies in the trajectory by applying temporal Self-Attention. The model uses the entire history as context to allow the model to use as much information as possible.
- PDE FNet: A model that is similar to the Vanilla Transformer, but it uses a modified version of the Vanilla FNet Transformer, with an improved token mixing mechanism based on 1D Fourier transforms and a complex-valued MLP.
- TFormer: Spatio-temporal resolution invariant model based on Linear Attention which unites the idea of Neural Operators and Physics Informed Neural Networks (PINNs).

6.0.1 Contributions

We introduce a method to utilize the PDE parameters in transformer-based models that emulate numerical PDE solvers. We also apply the PDE parameter conditioning of cFNO [TAN23] to transformer-based models (OFormer and our TFormer). Furthermore, we propose four different transformer-based models to solve PDEs.

Recurrent Linear Transformer and Vanilla Transformer

The contributions of these models include:

- Usage of transformer-based Self-Attention to model the evolution of PDE dynamics by applying temporal Self-Attention
- A model that incorporates the entire history, the initial condition to the latest prediction, to predict future timesteps

- A single model to handle variable temporal context size (different numbers of timesteps used as the initial condition)
- Extension to extrapolate to unseen lengths (sliding window approach)
- A model that generalizes to unseen PDE Parameters by utilizing our PDE parameter embedding mechanism
- Investigating the importance of attention weights to understand the model's behaviours

PDE FNet

Several important contributions are made by the PDE FNet, such as:

- Improved token mixing for Vanilla FNet Transformer by using a 1D Fourier transform and a complex-valued MLP. The changes are not specific for solving PDEs and might be beneficial for time-series tasks or NLP
- Using a Transformer with Fourier transforms to propagate the dynamics of PDE

TFormer

Noteworthy contributions of the TFormer encompass:

- TFormer unites the advantages of Neural Operators and PINNs
- A model that is both spatially and temporally resolution invariant
- A model that represents the solution function u and doesn't emulate numerical PDE solvers
- Spatio-Temporal Conditioning: An easy way to condition the initial condition latent representation with the spatio-temporal latent representation to propagate the dynamics of the PDE. Transformers with Cross-Attention inspire spatio-temporal conditioning, but we propose to replace the Cross-Attention component with a non-linearity and a pointwise multiplication as in Feature-wise Linear Modulation (FiLM) [PSV+17]

6.0.2 Comparison against other Models

OFormer from Li et al. [LMF23] uses the Transformer to mix up $u(t, x_i)$ values from different grid positions x_i (spatial Self-Attention) and an MLP to calculate the PDE dynamics whereas the Vanilla Transformer, Recurrent Linear Transformer (RLT), and PDE FNet use Transformers to calculate dynamics of the PDE (temporal Self-Attention). Both models use a type of Linear Attention, but OFormer uses a different linearized attention (Galerkin and Fourier Attention [Cao21]) than our models (Linear Attention with kernel function [KVPF20]). The TFormer also applies spatial Self-Attention but uses the prediction time as additional input to directly predict the solution at an *arbitrary* time t. In contrast, FNO, OFormer, and Galerkin Transformer predict the solution by unrolling the entire trajectory until timestep t. Consequently, these models are fixed to a temporal step size. OFormer uses Cross-Attention to query for arbitrary grid points while the proposed TFormer uses the pointwise matrix product to query for arbitrary prediction times.

FNO from Li et al. [LKA+20b] *lifts* each solution point $u(t, x_i)$ to a higher dimensional space while our approaches (only RLT, Vanilla Transformer, and PDE FNet) encode $u(t, X) = (u(t, x_1), \ldots, u(t, x_s))^T \in \mathbb{R}^{s \times c}$ in a latent space that has a smaller dimension. The conditioned FNO (cFNO) adds the PDE parameter explicitly to every solution point whereas the embedding approach for Vanilla Transformer, RLT, and PDE FNet does it implicitly. However, our PDE parameter embedding mechanism also guarantees that each component of latent representation could be influenced by the PDE parameter.

We define the following properties and compare the baseline models with our proposed models regarding the defined properties in Table 6.1.

PDE parameter-conditioned: A model is parameter-conditioned if it takes the PDE parameter into account as an additional input.

Temporal generalization: We say that a model can do temporal generalization if it can be used to predict rollouts with a low error which are longer than the rollouts done during training (i.e., go beyond the time seen during training).

Spatial zero-shot super-resolution: Spatial zero-shot super-resolution means that a model is not fixed to a spatial resolution that is seen during training. The model can be trained on a spatial resolution of s and evaluated on a spatial resolution of s' without adapting the model's architecture and without retraining.

Temporal zero-shot super-resolution: Similar to spatial zero-shot super-resolution but for temporal dimension.

Model	PDE parameter-conditioned	Temporal Generalization	Zero-Shot Super-Resolution	
			Spatial	Temporal
FNO	✗	✓	✓	✗
OFormer	✗	✓	✓	✗
Galerkin Transformer	✗	✓	✓	✗
cFNO	✓	✓	✓	✗
cOFormer	✓	✓	✓	✗
Vanilla Transformer (*)	✓	✓	✗	✗
Recurrent LT (*)	✓	✓	✗	✗
PDE FNet (*)	✓	✓	✗	✗
TFormer (*)	✓	✓	✓	✓

Table 6.1: Different properties of the baseline and proposed models. Our models are marked with (*).

7 Benchmark PDEs

We train and evaluate our models on the following 1D PDEs from PDEBench of Takamoto et al. [TPL+22]. The dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ for each PDE consists of $n = 10k$ samples where $9k$ samples are used for training and $1k$ samples for testing. $x_j = (u(t_0, \cdot), \ldots, u(t_{N_I}, \cdot))$ denotes the solutions given as the initial condition and $y_j = ((u(t_0, \cdot), \ldots, u(t_N, \cdot))$ denotes the target sequence of timesteps which represents the trajectory of the PDE. N_I is the number of initial timesteps and N is the number of all timesteps which depends on the temporal discretization or temporal resolution. $u(t, \cdot) \in \mathbb{R}^{s \times c}$, with s spatial resolution, contains the solutions on different spatial coordinates.

7.1 1D Burger's Equation

$$\partial_t u(t,x) + u(t,x)\partial_x u(t,x) = \frac{\nu}{\pi}\partial_{xx} u(t,x) \tag{7.1}$$

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

7.2 1D Advection Equation

$$\partial_t u(t,x) + \beta\partial_x u(t,x) = 0 \tag{7.2}$$

where the PDE parameter β denotes the advection velocity. The dataset contains solutions for $x \in (0, 1)$ with a maximum spatial resolution of $s = 1024$ and $t \in (0, 2]$ with a maximum temporal resolution of $N = 201$. Similar to 1D Burgers, we subsample the data to get a trajectory of $N = 41$ timesteps and a spatial resolution of $s = 256$.

7.3 1D Computational Fluid Dynamics (CFD)

The compressible Navier-Stokes equations (Equations 7.3a to 7.3c) 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). Equation 7.3a refers to the continuity equation which is also called the transport equation or equation of conservation of mass, and Equation 7.3b describes the conservation of momentum.

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{7.3a}$$

$$\rho(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \eta \triangle \mathbf{v} + (\zeta + \frac{\eta}{3}) \nabla(\nabla \cdot \mathbf{v}) \tag{7.3b}$$

$$\partial_t (\epsilon + \frac{\rho v^2}{2}) + \nabla \cdot [(p + \epsilon + \frac{\rho v^2}{2})\mathbf{v} - \mathbf{v} \cdot \sigma'] = 0 \tag{7.3c}$$

where ρ represents the density of the mass or fluid, $\mathbf{v}$ denotes the fluid velocity (as a 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 along both temporal and spatial dimensions yielding a trajectory of $N = 51$ timesteps and a spatial resolution of $s = 256$.

7.4 Used PDE Parameters

We use the PDE parameters in Table 7.1 for our experiments with one single PDE parameter. This means that we train on a single PDE parameter and that we test the models on the same single parameter. Table 7.2 shows the combinations of PDE parameters used in our experiments with multiple parameters. In this case, we train the models on a set of PDE parameters and test it on a set of PDE parameters that include the seen and unseen PDE parameters.

PDE	Trajectory length N	Spatial resolution s	Training and testing parameters
1D Burgers	41	256	$\nu = 0.001$
1D Advection	41	256	$\beta = 0.1$
1D CFD	$41^{(1)}$	256	$\eta = \zeta = 0.007$

Table 7.1: PDE parameters used in the experiments with a single PDE parameter value. (1) The original trajectory has 51 timesteps, but the models are trained on the first 41 timesteps only to be consistent with the other experiments.

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 CFD	$\eta = \zeta = (10^{-8}, 0.001, 0.004, 0.01, 0.04, 0.1)$	$\eta = \zeta = (0.007, 0.07)$

Table 7.2: Exemplary set of PDE parameters used in the experiments with multiple PDE parameters.

8 Empirical Results

We focus on training and testing the considered models with one timestep as the initial condition ($N_I = 1$) and predicting multiple future steps (40 timesteps in future, trajectory length $N = 41$) as this setting is better suited for real-world applications. The initial condition could be the data from measurements, and multiple future timesteps are usually required to do a simulation of the physical system under consideration. We choose FNO [LKA+20b], OFormer [LMF23], and Galerkin Transformer [Cao21] as the state-of-the-art baseline or reference models. We also test the PDE parameter conditioned FNO [TAN23] and a PDE parameter conditioned OFormer. The predictions of FNO, OFormer, Galerkin Transformer, Vanilla Transformer, Recurrent Linear Transformer, and PDE FNet are achieved in an autoregressive fashion while TFormer predicts the entire trajectory directly in one shot.

The models mentioned in Chapter 6 are evaluated on the PDEs introduced in Chapter 7. Table 8.1 lists the most important specifications of the evaluated models such as the number of parameters and the training time. The reported benchmark scores are the mean values obtained by training and testing the models using five different initializations. We also provide the percentage deviation of the errors between FNO and the other models, as FNO seems to be a very strong model for us that has high accuracy, is lightweight and is fast at doing inference. The entire configuration of hyperparameters as well as the benchmark results can be found in Appendix C.

In the first Section 8.1, we train and test the models on a fixed PDE with a fixed PDE parameter to test the model's ability to generalize to different initial conditions. Furthermore, we test whether FNO, OFormer, and TFormer can do spatial zero-shot super-resolution. We also investigate the temporal and spatio-temporal zero-shot super-resolution of our proposed TFormer. As we are interested in doing longer rollouts, we also evaluate this property and test the proposed sliding window approach to overcome the Transformer length generalization problem.

The second Section 8.2 deals with the generalization of different PDE parameters. We train the models on a fixed PDE with a set of PDE parameters (i.e., a set of different PDE parameters) and evaluate the models on the same PDE, but with another set of PDE parameters that includes the seen and unseen PDE parameters.

Finally, in Section 8.3 we compare the inference times and memory consumption of the baseline and proposed models.

8.1 Single PDE Parameter

We train and test the models on one single PDE Parameter to demonstrate that the models also perform well when used on one specific PDE Parameter. The full benchmark results are available in Appendix C.1.

Model name	# Parameters	GPU memory	# Epochs	Training time	Time per epoch
FNO	550 k	4.3 GiB	500	3.6 h	26 s
OFormer	661 k	45 GiB	500	16.7 h	120 s
Galerkin Transformer	530 k	14.2 GiB	500	10.3 h	74 s
PyTorch Vanilla Transformer (*)	796 k	4.2 GiB	1000	16.1 h	58 s
Naive Vanilla Transformer (*)	796 k	4.7 GiB	1000	20 h	72 s
Recurrent LT (*)	796 k	1.6 GiB	1000	19.7 h	71 s
PDE FNet small (*)	462 k	2.8 GiB	1000	12.5 h	45 s
PDE FNet large (*)	796 k	3.3 GiB	1000	16.9 h	61 s
TFormer (*)	794 k	24 GiB	500	15.3 h	55 s

Table 8.1: Overview of the specifications of the compared models used in the 1D Burgers experiments. GPU memory consumption, training time, and time per epoch refer to a training with a batch size of 64 on a single NVIDIA A100-SXM4 80GB GPU. Training time and time per epoch include the time that is needed to load the data and transfer it to the GPU. Our proposed models are marked with (*). The values of TFormer differ from Table 4.1 as the models in Table 4.1 were trained on 4 GPUs using data parallelism, yielding a smaller time per epoch but a higher memory consumption since each GPU has a copy of the model, and the models in this table were trained on a single GPU.

8.1.1 Generalization to Different Initial Conditions

First of all, we test whether our proposed models can generalize to different initial conditions by testing the models on the corresponding test sets of PDEBench. The models are trained and tested on a reduced spatial resolution (256 solution points per timestep) and reduced temporal resolution (41 timesteps in total). Table 8.2 shows the errors of the models when trained and evaluated on 1D Burgers' equation with $\nu = 0.001$, 1D Advection with $\beta = 0.1$, and 1D CFD with $\eta = \zeta = 0.007$. For 1D Burgers, all of our proposed models have a lower RMSE and normalized RMSE than FNO. The RMSE on the boundary of TFormer is higher than the error of the other models. For 1D Advection and 1D CFD, all proposed models also achieve state-of-the-art performance. The Galerkin Transformer has the highest errors in all three experiments.

Example Predictions

Figure 8.1 shows an example prediction of the Vanilla Transformer with Scaled Dot-Product Attention and Figure 8.2 shows an example prediction of the Recurrent Linear Transformer (RLT) on 1D Advection. The similarity between the prediction and ground truth is clearly visible, nevertheless, there is space for improvement.

PDE	Model	RMSE	nRMSE	bRMSE
1D Burgers ($\nu = 0.001$)	FNO	0.03511	0.09821	0.02177
	OFormer	0.03402 (-3.09%)	0.10326 (+5.14%)	0.02144 (-1.55%)
	Galerkin T	0.06170 (+75.75%)	0.16545 (+68.47%)	0.03658 (+67.97%)
	Vanilla T (*)	**0.03019** (-14.00%)	0.08308 (-15.41%)	0.01704 (-21.72%)
	Recurrent LT (*)	0.03110 (-11.41%)	0.08583 (-12.61%)	**0.01495** (-31.32%)
	PDE FNet S (*)	0.03287 (-6.37%)	0.09199 (-6.34%)	0.01760 (-19.17%)
	PDE FNet L (*)	0.03127 (-10.94%)	0.08947 (-8.90%)	0.01567 (-28.02%)
	TFormer (*)	0.03080 (-12.27%)	**0.07977** (-18.77%)	0.02258 (+3.69%)
1D Advection ($\beta = 0.1$)	FNO	0.01114	0.01901	0.02366
	OFormer	0.00726 (-34.83%)	0.01252 (-34.12%)	0.00805 (-65.98%)
	Galerkin T	0.03238 (+190.58%)	0.05576 (+193.36%)	0.03575 (+51.10%)
	Vanilla T (*)	0.00596 (-46.55%)	0.01072 (-43.60%)	0.00453 (-80.88%)
	Recurrent LT (*)	**0.00377** (-66.18%)	**0.00695** (-63.46%)	**0.00250** (-89.42%)
	PDE FNet S (*)	0.01186 (+6.47%)	0.02005 (+5.50%)	0.01060 (-55.20%)
	PDE FNet L (*)	0.00791 (-28.98%)	0.01391 (-26.79%)	0.00734 (-68.99%)
	TFormer (*)	0.01030 (-7.56%)	0.01722 (-9.40%)	0.00868 (-63.32%)
1D CFD ($\eta = \zeta = 0.007$)	FNO	2.47824	0.52616	2.02319
	OFormer	2.63904 (+6.49%)	0.51778 (-1.59%)	2.14893 (+6.21%)
	Galerkin T	3.57612 (+44.30%)	0.70376 (+33.76%)	3.02366 (+49.45%)
	Vanilla T (*)	1.46042 (-41.07%)	0.40786 (-22.48%)	1.09774 (-45.74%)
	Recurrent LT (*)	**1.33055** (-46.31%)	0.39167 (-25.56%)	**0.97581** (-51.77%)
	PDE FNet S (*)	2.07708 (-16.19%)	0.58116 (+10.45%)	1.53763 (-24.00%)
	PDE FNet (*) L	1.64760 (-33.52%)	0.45492 (-13.54%)	1.21210 (-40.09%)
	TFormer (*)	1.55307 (-37.33%)	**0.29644** (-43.66%)	1.32100 (-34.71%)

Table 8.2: Errors of different models trained and tested on the spatial resolution of 256 and temporal resolution of 41 timesteps. nRMSE denotes the normalized RMSE and bRMSE the RMSE at the boundaries. The value in parentheses denotes the percentage deviation to FNO. Our proposed models are marked with (*) and lower values are better.

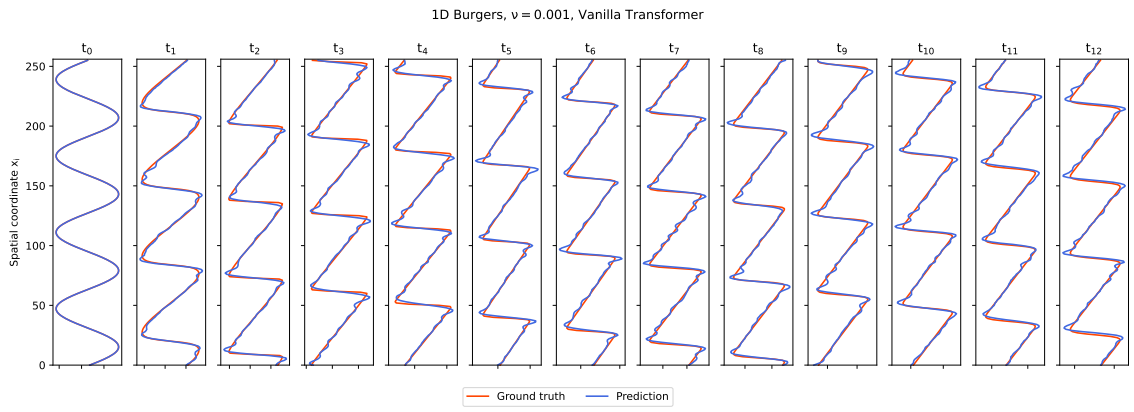


Figure 8.1: Example predictions of the Vanilla Transformer for a random initial condition of the 1D Burgers test set with PDE parameter $\nu = 0.001$. The plot shows the first 12 predictions of the model and the ground truth.

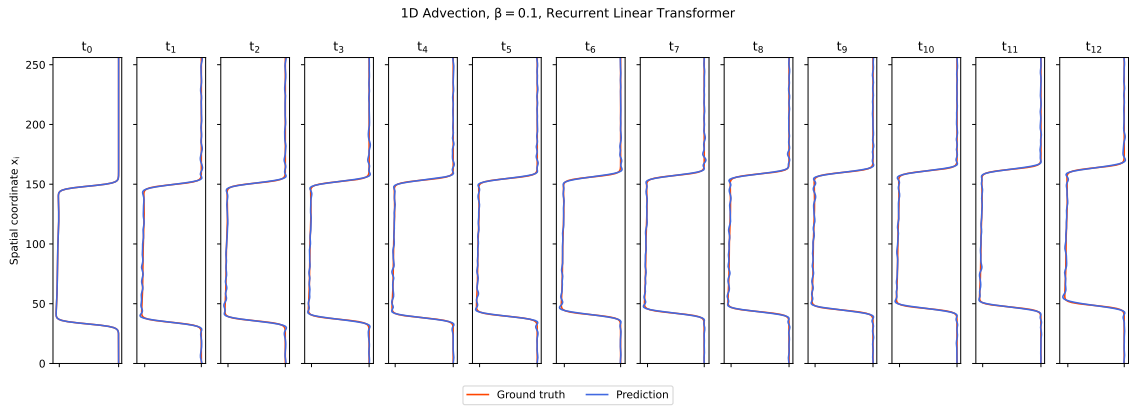


Figure 8.2: Example predictions of the Recurrent Linear Transformers for a random initial condition of the 1D Advecion test set with PDE parameter $\beta = 0.1$. The plot shows the first 12 predictions of the model and the ground truth.

2D Spatio-Temporal Error Visualization

The 2D spatio-temporal error visualization or error heatmap helps to visualize the error along the spatial and temporal domain. Let $y_t \in \mathbb{R}^s$ be the ground truth and $\widehat{y_t} \in \mathbb{R}^s$ model's prediction for a given timestep t. Then we calculate the pointwise relative error for the 2D error visualization as $e(t) = \sqrt{\frac{(y_t - \widehat{y_t})^2}{y_t^2}} = \frac{|y_t - \widehat{y_t}|}{|y_t|} \in \mathbb{R}^s$. Each operation is applied pointwise to the elements in y_t and $\widehat{y_t}$. The error heatmap for 1D Burgers' equation in Figure 8.3 shows that all models, except the TFormer, have a very high error at the first and second timestep (white area at the beginning of the trajectory). Figure 8.4 shows that the FNO and Galerkin Transformer have a very high error on a few spatial coordinates for 1D Advection PDE (red straight lines). The error heatmap for 1D CFD in Figure 8.11 reveals that the PDE FNet struggle for some specific spatial points at the beginning of the trajectory.

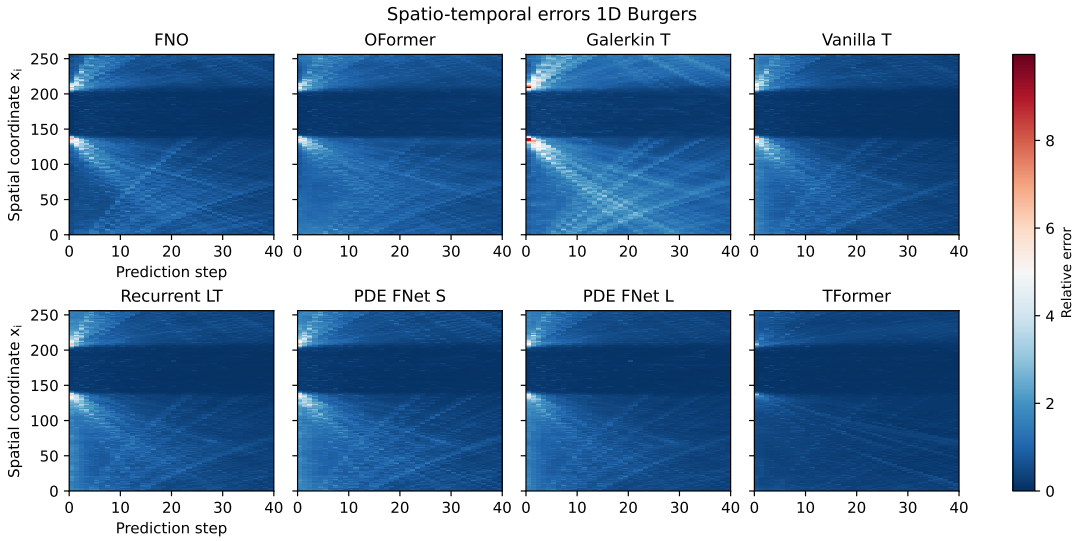


Figure 8.3: Error heatmap for 1D Burger's equation with $\nu = 0.001$. The models are trained and tested on a spatial resolution $s = 256$ and temporal resolution $N = 41$.

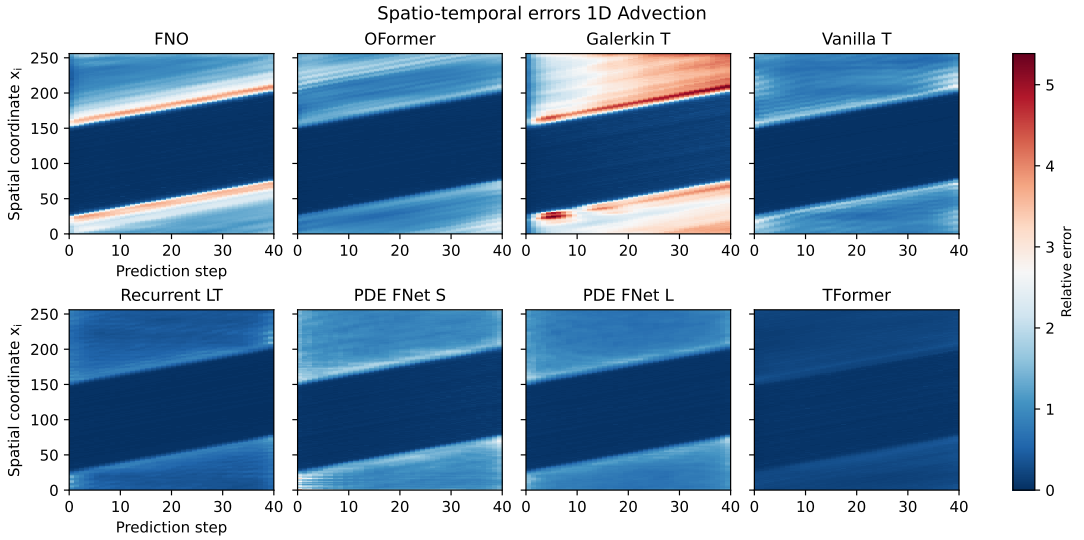


Figure 8.4: Error heatmap for 1D Advection with $\beta = 0.1$. The models are trained and tested on a spatial resolution $s = 256$ and temporal resolution $N = 41$.

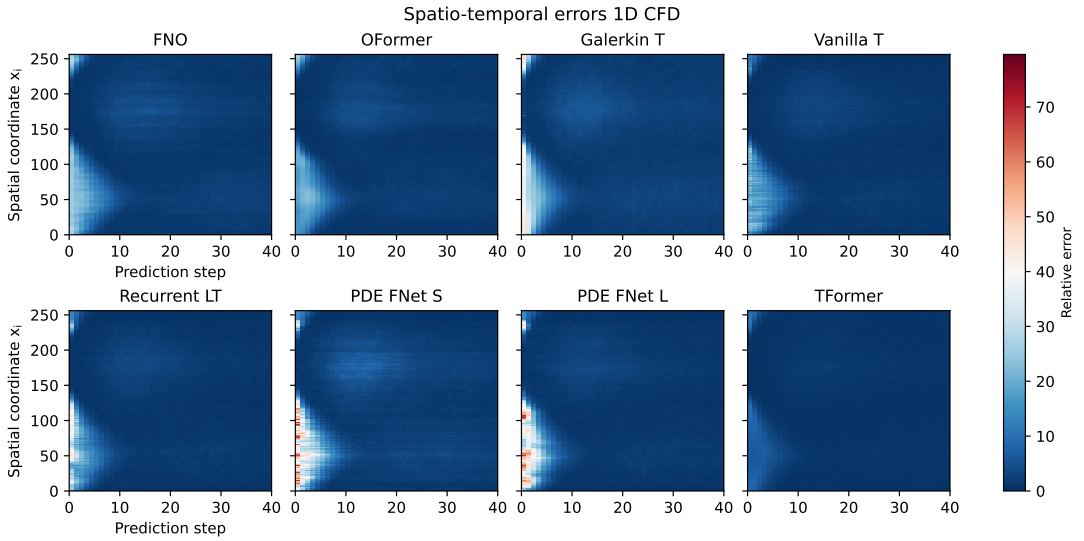


Figure 8.5: Error heatmap for 1D CFD with $\eta = \zeta = 0.007$. The models are trained and tested on a spatial resolution $s = 256$ and temporal resolution $N = 41$.

Violin Plot

We use a violin plot to visualize the error distribution on the test set. We consider each sample in the test set (i.e., prediction of complete trajectory) as a sample and calculate the RMSE for each sample individually. The error distribution across all samples, in our case 1k samples, is shown in the violin plot. It is noticeable that our proposed models perform well. The error of the Galerkin Transformer is in all experiments higher than compared to the remaining reference models and our models.

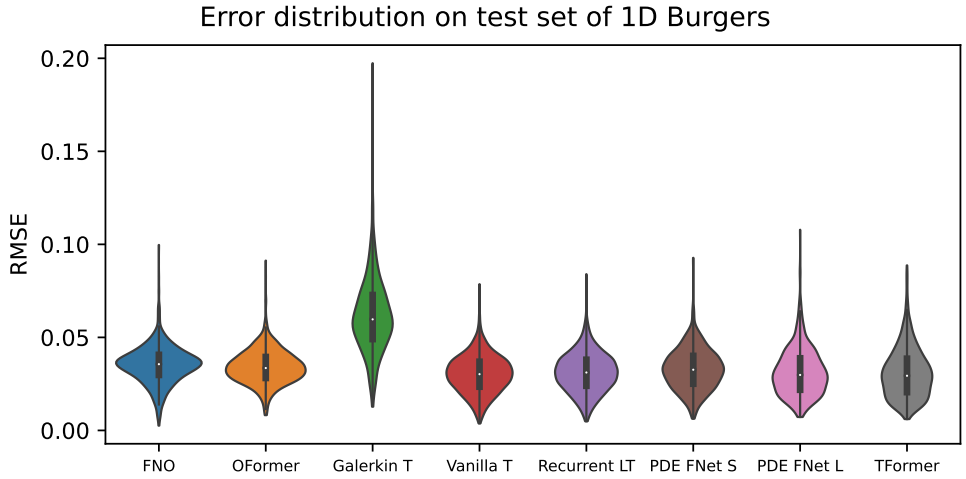


Figure 8.6: Error distribution of all samples in the test set of 1D Burgers.

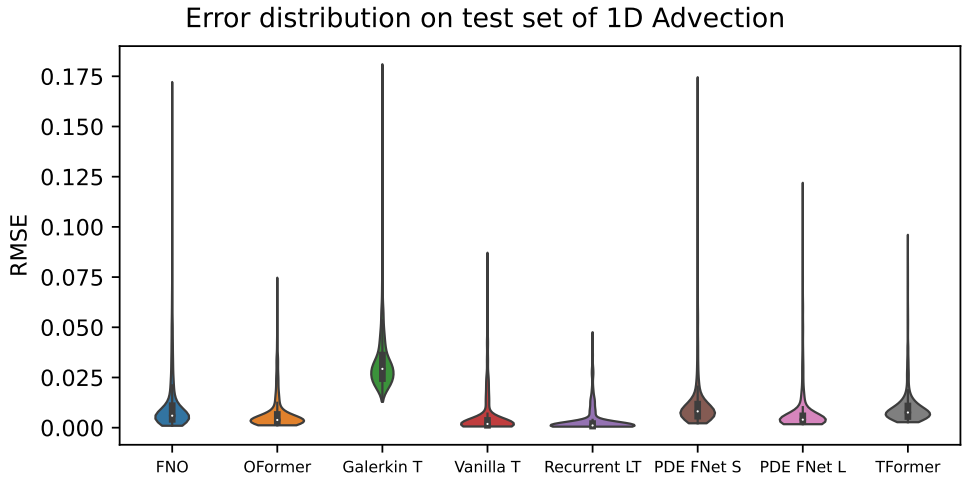


Figure 8.7: Error distribution of all samples in the test set of 1D Advection.

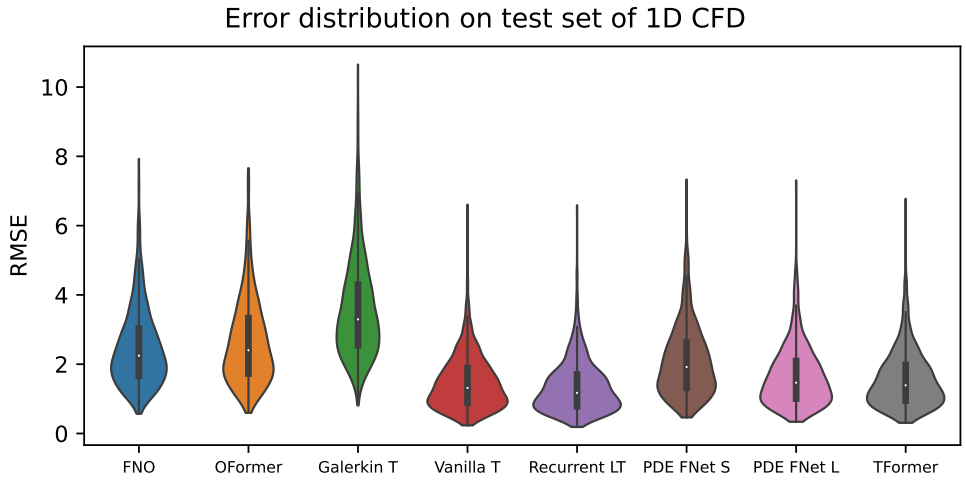


Figure 8.8: Error distribution of all samples in the test set of 1D CFD.

Temporal Error

The following section shows the temporal errors of the baselines and proposed models. Figure 8.9 shows the temporal error of the models for the 1D Burger's equation. It is noticeable that the errors of all models, except TFormer, decrease with time. This is probably correlated to the fact that TFormer directly predicts the solution while the other models do the predictions in an autoregressive fashion. The same effect can be observed for 1D CFD (Figure 8.11). For 1D Advection (Figure 8.10), the errors of all models increase with time.

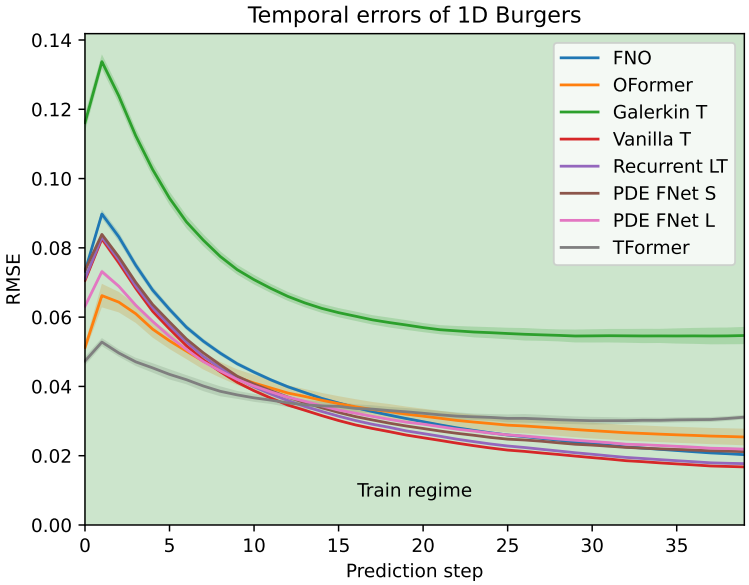


Figure 8.9: Temporal error of the models for 1D Burgers PDE. The confidence band shows the standard deviation.

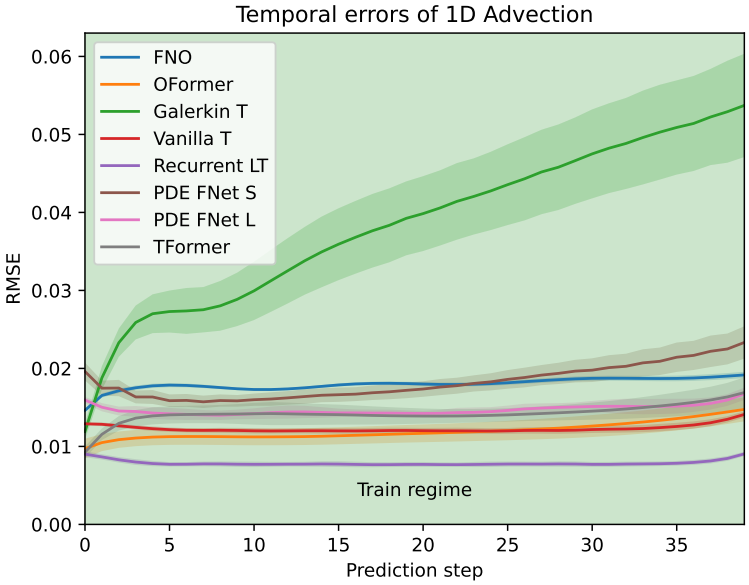


Figure 8.10: Temporal error of the models for 1D Advection PDE. The confidence band shows the standard deviation.

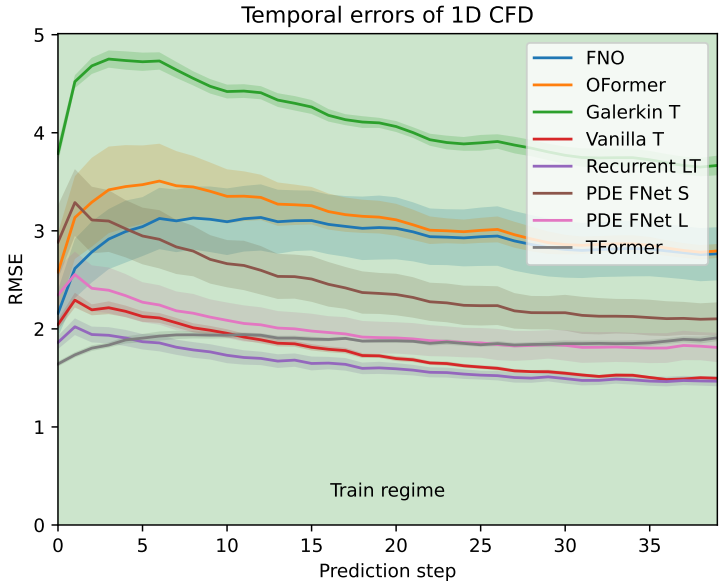


Figure 8.11: Temporal error of the models for 1D CFD PDE. The confidence band shows the standard deviation.

8.1.2 Spatial and Temporal Zero-Shot Super-Resolution

We also test the ability of the models to do spatial and temporal Zero-Shot Super-Resolution (ZSSR) which means that the model is trained on a reduced spatial and temporal resolution, and is used to do predictions with higher resolutions at inference time. We completely exclude the Vanilla Transformer, Recurrent Linear Transformer, and PDE FNet models since these models encode each timestep in a latent space with a fixed dimension that prevents doing a real spatial ZSSR. We also exclude the Galerkin Transformer, which can do spatial zero-shot super-resolution, since the model performed worse in the first experiment, where we tested the models on the same spatial resolution as seen during training.

Spatial Zero-Shot Super-Resolution

Table 8.3 shows the errors when using a model that was trained on a low spatial resolution to do inference with a higher resolution. The FNO was not able to do spatial zero-shot super-resolution on 1D Burgers and 1D Advection PDEs in our experiments. OFormer and our proposed TFormer seem to have spatial ZSSR since there is no significant increase in the error.

PDE	Spatial resolution	Model	RMSE	nRMSE	bRMSE
1D Burgers	256	FNO	0.03511	0.09821	0.02177
		OFormer	0.03402 (-3.09%)	0.10326 (+5.14%)	**0.02144** (-1.55%)
		TFormer (*)	**0.03080** (-12.27%)	**0.07977** (-18.78%)	0.02258 (+3.69%)
	512	FNO	0.08932	0.23637	0.04726
		OFormer	0.03583 (-59.89%)	0.10793 (-54.34%)	**0.02240** (-52.60%)
		TFormer (*)	**0.03107** (-65.22%)	**0.08053** (-65.93%)	0.02267 (-52.03%)
	1024	FNO	0.11775	0.31598	0.06377
		OFormer	0.03639 (-69.10%)	0.10962 (-65.31%)	0.02285 (-64.16%)
		TFormer (*)	**0.03125** (-73.46%)	**0.08105** (-74.35%)	**0.02275** (-64.33%)
1D Advection	256	FNO	0.01114	0.01901	0.02366
		OFormer	**0.00726** (-34.83%)	**0.01252** (-34.12%)	**0.00805** (-65.98%)
		TFormer (*)	0.01030 (-7.56%)	0.01722 (-9.40%)	0.00868 (-63.32%)
	512	FNO	0.15456	0.22270	0.18218
		OFormer	**0.00860** (-94.43%)	**0.01443** (-93.52%)	0.01331 (-92.69%)
		TFormer (*)	0.01052 (-93.20%)	0.01753 (-92.13%)	**0.00927** (-94.91%)
	1024	FNO	0.25812	0.37613	0.30754
		OFormer	**0.00972** (-96.23%)	**0.01604** (-95.74%)	0.01645 (-94.65%)
		TFormer (*)	0.01079 (-95.82%)	0.01793 (-95.23%)	**0.00972** (-96.84%)
1D CFD	256	FNO	2.47824	0.52616	2.02319
		OFormer	2.63904 (+6.49%)	0.51778 (-1.59%)	2.14893 (+6.21%)
		TFormer (*)	**1.55307** (-37.33%)	**0.29644** (-43.66%)	**1.32100** (-34.71%)
	512	FNO	3.17138	0.57985	2.74778
		OFormer	3.22349 (+1.64%)	0.55294 (-4.64%)	2.65562 (-3.35%)
		TFormer (*)	**1.55228** (-51.05%)	**0.29642** (-48.88%)	**1.32197** (-51.89%)
	1024	FNO	3.78287	0.62778	3.44694
		OFormer	3.23065 (-14.60%)	0.55220 (-12.04%)	2.66293 (-22.75%)
		TFormer (*)	**1.55194** (-58.97%)	**0.29642** (-52.78%)	**1.32298** (-61.62%)

Table 8.3: Errors of different models trained on a spatial resolution of 256 and evaluated on higher spatial resolutions. We use the same setup as introduced in Section 8.1 (i.e., one timestep as the initial condition and predicting the next 40 timesteps). The value in parentheses denotes the percentage deviation to FNO that was evaluated at the same spatial resolution. Our proposed model is marked with (*).

Temporal Zero-Shot Super-Resolution

We also test the ZSSR along the temporal domain of the TFormer. This means that the model is evaluated on a higher temporal solution than seen during training by adapting the query or prediction times t accordingly (i.e., querying with intermediate times t). The temporal resolution describes how many timesteps the trajectory is divided into. A small time discretization step size $\Delta t = t_{i+1} - t_i$ yields a high temporal resolution and a large time discretization step size Δt generates a low temporal resolution. We exclude OFormer and FNO as these models do not know the temporal resolution implicitly or explicitly. Table 8.4 shows that there is no increase in error for 1D Burgers and 1D Advection which means that the model can do temporal ZSSR in the train regime and learns the dependency between solution and time. For 1D CFD, the error doubles which indicates that the model struggles with the 1D CFD PDE.

PDE	Temporal resolution	Model	RMSE	nRMSE	bRMSE
1D Burgers	41	TFormer	0.03080	0.07977	0.02258
	101	TFormer	0.03134	0.08020	0.02305
	201	TFormer	0.03156	0.08039	0.02327
1D Advection	41	TFormer	0.01030	0.01722	0.00868
	101	TFormer	0.01028	0.01718	0.00867
	201	TFormer	0.01028	0.01718	0.00867
1D CFD	41	TFormer	1.55307	0.29644	1.32100
	101	TFormer	3.19055	0.41232	2.69203

Table 8.4: Errors of different models trained on a temporal resolution of 41 timesteps and evaluated on higher temporal resolutions. The spatial resolution is 256 (same as during training).

Spatial and Temporal Zero-Shot Super-Resolution

In this section, we test the ability of TFormer to do spatial and temporal zero-shot super-resolution in parallel by using a model trained on lower spatial and temporal resolution doing inference on a higher spatial and temporal resolution. We exclude all other models since these models aren't able to do spatio-temporal zero-shot super-resolution. Table 8.5 shows the metrics of this experiment.

PDE	Spatial resolution	Temporal resolution	Model	RMSE	nRMSE	bRMSE
1D Burgers	256	41	TFormer	0.03080	0.07977	0.02258
		101	TFormer	0.03134	0.08020	0.02305
		201	TFormer	0.03156	0.08039	0.02327
	512	41	TFormer	0.03107	0.08053	0.02267
		101	TFormer	0.03160	0.08094	0.02312
		201	TFormer	0.03182	0.08113	0.02335
	1024	41	TFormer	0.03125	0.08105	0.02275
		101	TFormer	0.03178	0.08145	0.02322
		201	TFormer	0.03201	0.08164	0.02345
1D Advection	256	41	TFormer	0.01030	0.01722	0.00868
		101	TFormer	0.01028	0.01718	0.00867
		201	TFormer	0.01028	0.01718	0.00867
	512	41	TFormer	0.01052	0.01753	0.00927
		101	TFormer	0.01050	0.01748	0.00925
		201	TFormer	0.01050	0.01748	0.00925
	1024	41	TFormer	0.01079	0.01793	0.00972
		101	TFormer	0.01077	0.01788	0.00970
		201	TFormer	0.01077	0.01788	0.00970
1D CFD	256	41	TFormer	1.55307	0.29644	1.32100
		101	TFormer	3.19055	0.41232	2.69203
	512	41	TFormer	1.55228	0.29642	1.32197
		101	TFormer	3.18979	0.41229	2.68824
	1024	41	TFormer	1.55194	0.29642	1.32298
		101	TFormer	3.18943	0.41228	2.68677

Table 8.5: Errors of TFormer model trained on a spatial resolution of 256 and temporal resolution of 41, evaluated on higher spatial and temporal resolutions.

8.1.3 Temporal Generalization (Predicting Long Rollouts)

In this section, we investigate the error of the models when used to predict more timesteps than seen during training (i.e., go beyond the times seen during training). We run the autoregressive loop for FNO, OFormer, Galerkin Transformer, and PDE FNet longer to predict more timesteps than seen during training. TFormer is queried with prediction times t that are larger than seen during training to predict a longer trajectory. We also demonstrate that the Recurrent Linear Transformer for PDEs is affected by the Transformer length generalization problem by also running the autoregressive loop longer. We compare random positional encoding proposed by [RDG+23] against our proposed sliding window approach for solving the issue.

Long autoregressive rollouts are challenging since the previous prediction is used as a new input which causes an error accumulation. Each prediction has an error and using the prediction that already has an error as input yields a prediction with a higher error than previously. This process repeats iteratively and at some point in time, the model will produce poor results because of a too large error accumulation.

We explore the temporal generalization on the 1D Burgers PDE with a spatial resolution $s = 256$ (same as during training) and a trajectory for $t \in (0, 4]$ (twice as long as seen during training). In contrast to temporal zero-shot super-resolution, the temporal discretizations stay the same, but the trajectory is twice as long.

FNO and OFormer

The FNO and OFormer take the previous solution as input and predict the next solution which can be expressed as $f_\theta : \mathbb{R}^s \to \mathbb{R}^s$ with $u(t_n, \boldsymbol{x}) \mapsto u(t_{n+1}, \boldsymbol{x})$. The iterative application of the model f_θ allows predictions of arbitrary lengths. Figure 8.12a shows the temporal errors of both models. There is no significant increase in the error in the extrapolation regime since these models only suffer from the error accumulation problem.

PDE FNet

The proposed PDE FNet utilizes a Fourier transform to replace the Self-Attention mechanism in the Transformer. The Fourier transform preserves the order of the tokens (i.e., timesteps) and allows omitting the additional positional encoding needed for the Linear or Vanilla Attention mechanism. Consequently, the PDE FNet doesn't suffer from the Transformer length generalization problem. Figure 8.12a shows that the PDE FNet can handle long rollouts without a significantly increasing error.

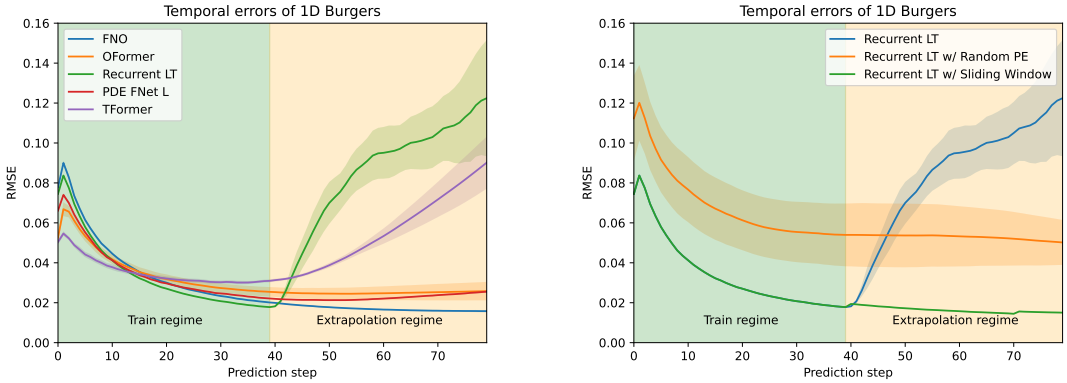
TFormer

Figure 8.12a reveals that the TFormer also suffers from a length generalization problem like regular Transformers. The error in the extrapolation regime increases since the queried time t gets out-of-distribution. A suitable fix would be applying the model autoregressively by using the latest prediction as new input to avoid the time t getting out-of-distribution.

Recurrent Linear Transformer

Our proposed Recurrent Linear Transformer and Vanilla Transformer for solving PDEs suffer from the Transformer length generalization problem mentioned in Section 2.3.6. The applied positional encoding gets out-of-distribution for sequences longer than seen during training. Figure 8.12a illustrates the problem that the error increases if the previous predictions are appended to the input even if the maximum context size is reached.

Therefore, we have to apply some additional techniques to reduce the errors produced by the out-of-distribution positional encoding. We compare random positional encoding (Random PE) proposed by Ruoss et al. [RDG+23] against our proposed sliding window approach. Figure 8.12b shows that the Recurrent LT with random positional encoding doesn't suffer from the length generalization problem as the Recurrent LT, but Random positional encoding leads to a higher variance and generally to a larger error. The higher variance could be explained by an increased stochasticity of the training process which is added through Random PE. Our proposed sliding window approach also solves the problem of the out-of-distribution PE and yields lower errors than Random PE.

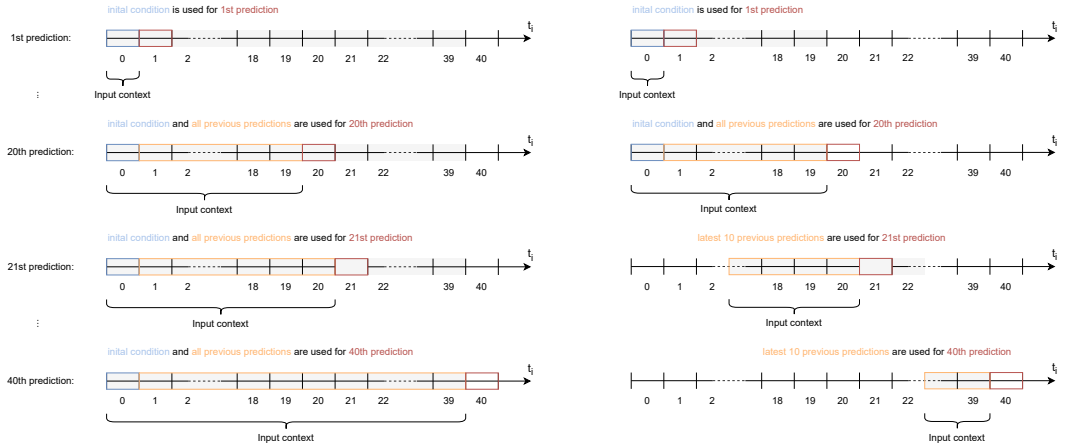


(a) Without any tricks to reduce the errors

(b) With Random PE and Sliding Window

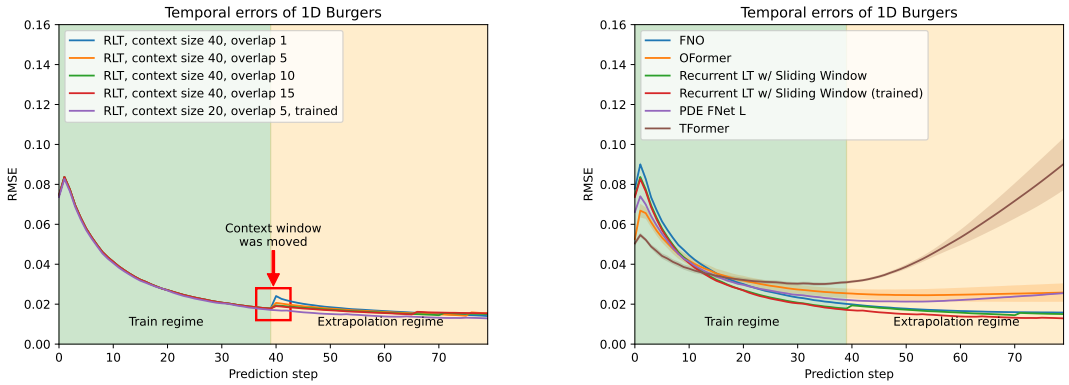
Figure 8.12: Temporal errors of different models trained to predict 40 timesteps given 1 timestep as the initial condition. Figure 8.12a shows the error of the models without any additional tricks (i.e., no random positional encoding and no sliding window approach) to reduce the error of Recurrent LT. Figure 8.12b shows the errors of Recurrent LT with random positional encoding and our proposed sliding window approach. Both techniques reduce the errors, but our proposed sliding window approach leads to a lower error.

The size of the context window and the overlap of the sliding window approach are hyperparameters that must be chosen appropriately. Figure 8.14a shows the errors of different configurations (overlaps and window sizes). A small overlap results in a "bump" in the error. This effect is reduced if the overlap is increased or if the model is trained with the sliding window approach. This means that the model is not trained to predict all 40 timesteps in one sweep (i.e., without any context window movement), but it is trained to predict 20 timesteps, move the context window, and then predict the next 10 timesteps and so on. Technically, the training with the sliding window approach is achieved by reducing the context window size from 40 to 20 timesteps, forcing the model to move the context window at training time to predict all 40 future timesteps. Consequently, the model is exposed to the sliding window approach during training which reduces the error of the sliding window approach during inference. Figure 8.13 illustrates the training with and without the sliding window approach. Figure 8.14b and Table 8.6 show the final values of the baseline and proposed models.



(a) Trained without sliding window **(b)** Trained with sliding window

Figure 8.13: Training without the sliding window approach (Figure 8.13a) and with the sliding window approach (Figure 8.13b). The context window, which is moved by the sliding window approach, is depicted by the grey rectangle. Reducing the context window size from 40 (Figure 8.13a) to 20 (Figure 8.13b) forces the model to shift the window at training time. Thus, the model is trained with the sliding window approach.



(a) Errors of different configurations **(b)** Final values

Figure 8.14: Temporal errors of different sliding window configurations and final values. Figure 8.14a shows the errors of different configurations (overlaps and context sizes) of the sliding window approach. The red box marks a "bump" that occurs if the context window is moved. This effect can be reduced by choosing a larger window overlap or training the model with the sliding window approach (purple line). Figure 8.14b displays the final values for models trained to predict 40 timesteps given 1 timestep as the initial condition.

Regime	Model	RMSE	nRMSE	bRMSE
Train	FNO	0.03515	0.09505	0.02134
	OFormer	0.03426 (-2.54%)	0.10023 (+5.45%)	0.02132 (-0.11%)
	Recurrent LT SW (*)	0.03157 (-10.21%)	0.08508 (-10.49%)	**0.01500** (-29.73%)
	Recurrent LT SW (trained) (*)	**0.03113** (-11.45%)	0.08373 (-11.91%)	0.01673 (-21.61%)
	PDE FNet L (*)	0.03185 (-9.41%)	0.08926 (-6.09%)	0.01581 (-25.93%)
	TFormer (*)	0.03126 (-11.09%)	**0.07794** (-18.01%)	0.02307 (+8.06%)
Extrapolation	FNO	0.01628	0.06888	0.01209
	OFormer	0.02296 (+41.04%)	0.11797 (+71.28%)	0.01487 (+22.98%)
	Recurrent LT SW (*)	0.01530 (-6.06%)	0.06700 (-2.73%)	0.00888 (-26.54%)
	Recurrent LT SW (trained) (*)	**0.01339** (-17.76%)	**0.05850** (-15.07%)	**0.00744** (-38.49%)
	PDE FNet L (*)	0.02020 (+24.06%)	0.08883 (+28.97%)	0.01159 (-4.13%)
	TFormer (*)	0.04410 (+170.87%)	0.12240 (+77.71%)	0.04572 (+278.15%)

Table 8.6: Errors of different models that were trained to predict 40 timesteps and used to predict 80 timesteps on the 1D Burgers dataset. The spatial resolution is 256 (same as during training) and the value in parentheses denotes the percentage deviation to FNO. Our proposed models are marked with (*) and lower values are better.

8.2 Multiple PDE Parameters

This section contains the results of our experiments where the models are trained on a set of different PDE parameters and evaluated on another set of PDE parameters that contain unseen PDE parameters. Table 7.2 shows the PDE parameter configurations.

1D Burgers

First, we train and test the models on the 1D Burgers' PDE. All models seem to perform well on seen and unseen PDE parameters, except for the parameter $\nu = 4.0$. This parameter is special since it is in the extrapolation regime of a large PDE parameter value. It is noticeable that the Recurrent Linear Transformer and PDE FNet Large have much lower errors than the other models.

PDE parameter	Model	RMSE	nRMSE	bRMSE
$\nu = 0.001$	cFNO	0.03725	0.10518	0.02272
	cOFormer	0.03236 (-13.13%)	0.09582 (-8.90%)	0.02059 (-9.39%)
	Vanilla T	0.03212 (-13.79%)	0.08913 (-15.25%)	0.01715 (-24.53%)
	RLT	0.03113 (-16.44%)	0.08673 (-17.54%)	**0.01499** (-34.04%)
	PDE FNet L	0.03184 (-14.54%)	0.09044 (-14.01%)	0.01549 (-31.83%)
	TFormer	**0.02633** (-29.34%)	**0.06837** (-34.99%)	0.01658 (-27.04%)

$\nu = 0.002$	cFNO	0.03564	0.09972	0.02200
	cOFormer	0.03065 (-14.01%)	0.09009 (-9.66%)	0.01984 (-9.81%)
	Vanilla T	0.03039 (-14.74%)	0.08318 (-16.59%)	0.01654 (-24.80%)
	RLT	0.02938 (-17.58%)	0.08067 (-19.11%)	**0.01437** (-34.65%)
	PDE FNet L	0.03014 (-15.45%)	0.08473 (-15.03%)	0.01493 (-32.11%)
	TFormer	**0.02469** (-30.73%)	**0.06250** (-37.32%)	0.01609 (-26.85%)
$\nu = 0.004$	cFNO	0.03221	0.08854	0.02047
	cOFormer	0.02717 (-15.66%)	0.07903 (-10.75%)	0.01830 (-10.57%)
	Vanilla T	0.02692 (-16.43%)	0.07206 (-18.62%)	0.01528 (-25.34%)
	RLT	0.02589 (-19.63%)	0.06943 (-21.58%)	**0.01313** (-35.87%)
	PDE FNet L	0.02685 (-16.62%)	0.07452 (-15.84%)	0.01384 (-32.37%)
	TFormer	**0.02179** (-32.35%)	**0.05352** (-39.55%)	0.01515 (-25.97%)
$\nu = 0.01$	cFNO	0.02323	0.06075	0.01655
	cOFormer	0.01915 (-17.57%)	0.05550 (-8.64%)	0.01443 (-12.79%)
	Vanilla T	0.01913 (-17.66%)	0.04986 (-17.93%)	0.01222 (-26.19%)
	RLT	0.01821 (-21.62%)	0.04752 (-21.78%)	**0.01019** (-38.42%)
	PDE FNet L	0.02005 (-13.69%)	0.05575 (-8.22%)	0.01148 (-30.64%)
	TFormer	**0.01679** (-27.74%)	**0.04100** (-32.50%)	0.01316 (-20.47%)
$\nu = 0.02$	cFNO	0.01429	0.03653	0.01279
	cOFormer	0.01284 (-10.15%)	0.03922 (+7.35%)	0.01074 (-16.06%)
	Vanilla T	0.01254 (-12.27%)	0.03345 (-8.43%)	0.00922 (-27.93%)
	RLT	**0.01195** (-16.38%)	**0.03210** (-12.13%)	**0.00752** (-41.21%)
	PDE FNet L	0.01471 (+2.92%)	0.04280 (+17.14%)	0.00945 (-26.12%)
	TFormer	0.01292 (-9.62%)	0.03251 (-11.03%)	0.01129 (-11.74%)
$\nu = 0.04$	cFNO	0.01023	0.03151	0.01058
	cOFormer	0.00996 (-2.65%)	0.03333 (+5.79%)	0.00838 (-20.75%)
	Vanilla T	0.00813 (-20.51%)	0.02361 (-25.09%)	0.00674 (-36.26%)
	RLT	**0.00796** (-22.20%)	**0.02335** (-25.91%)	**0.00566** (-46.49%)
	PDE FNet L	0.01102 (+7.73%)	0.03428 (+8.77%)	0.00785 (-25.81%)
	TFormer	0.01022 (-0.11%)	0.02805 (-10.98%)	0.00976 (-7.76%)
$\nu = 0.1$	cFNO	0.00980	0.03267	0.00925
	cOFormer	0.00818 (-16.48%)	0.03394 (+3.90%)	0.00716 (-22.56%)
	Vanilla T	**0.00528** (-46.14%)	**0.01765** (-45.96%)	0.00451 (-51.20%)
	RLT	0.00531 (-45.79%)	0.01777 (-45.60%)	**0.00416** (-55.02%)
	PDE FNet L	0.00781 (-20.30%)	0.02841 (-13.03%)	0.00606 (-34.46%)
	TFormer	0.00981 (+0.18%)	0.03279 (+0.36%)	0.00921 (-0.43%)
$\nu = 0.2$	cFNO	0.00638	0.02449	0.00710
	cOFormer	0.00553 (-13.36%)	0.02422 (-1.11%)	0.00549 (-22.65%)
	Vanilla T	**0.00320** (-49.90%)	**0.01205** (-50.80%)	0.00291 (-59.06%)
	RLT	0.00325 (-49.11%)	0.01264 (-48.39%)	**0.00260** (-63.41%)
	PDE FNet L	0.00496 (-22.36%)	0.02032 (-17.01%)	0.00409 (-42.43%)
	TFormer	0.00574 (-10.03%)	0.02202 (-10.08%)	0.00633 (-10.78%)

$\nu = 0.4$	cFNO	0.00500	0.02260	0.00726
	cOFormer	0.00457 (-8.45%)	0.02241 (-0.83%)	0.00472 (-34.96%)
	Vanilla T	**0.00217** (-56.65%)	**0.00976** (-56.83%)	0.00200 (-72.45%)
	RLT	0.00224 (-55.09%)	0.01031 (-54.37%)	**0.00179** (-75.26%)
	PDE FNet L	0.00348 (-30.28%)	0.01769 (-21.70%)	0.00301 (-58.54%)
	TFormer	0.00440 (-11.84%)	0.02091 (-7.46%)	0.00505 (-30.41%)
$\nu = 1.0$	cFNO	0.02253	0.19532	0.02387
	cOFormer	0.02344 (+4.02%)	0.15105 (-22.67%)	0.02512 (+5.23%)
	Vanilla T	0.00566 (-74.89%)	0.03273 (-83.24%)	0.00486 (-79.64%)
	RLT	**0.00460** (-79.57%)	**0.02788** (-85.73%)	**0.00362** (-84.84%)
	PDE FNet L	0.00503 (-77.66%)	0.03570 (-81.72%)	0.00444 (-81.39%)
	TFormer	0.00884 (-60.78%)	0.06013 (-69.21%)	0.00869 (-63.58%)
$\nu = 2.0$	cFNO	0.00357	0.02330	0.00620
	cOFormer	0.00345 (-3.30%)	0.02246 (-3.58%)	0.00404 (-34.88%)
	Vanilla T	0.00178 (-50.22%)	0.00916 (-60.66%)	0.00188 (-69.60%)
	RLT	**0.00172** (-51.73%)	**0.00859** (-63.11%)	**0.00153** (-75.37%)
	PDE FNet L	0.00218 (-39.07%)	0.01452 (-37.68%)	0.00197 (-68.13%)
	TFormer	0.00372 (+4.06%)	0.02989 (+28.30%)	0.00419 (-32.45%)
$\nu = 4.0$	cFNO	0.52389	5.99772	0.53006
	cOFormer	0.15337 (-70.72%)	1.42134 (-76.30%)	0.14382 (-72.87%)
	Vanilla T	0.09770 (-81.35%)	0.28902 (-95.18%)	0.09467 (-82.14%)
	RLT	0.03169 (-93.95%)	**0.15804** (-97.36%)	0.02801 (-94.72%)
	PDE FNet L	**0.02700** (-94.85%)	0.27361 (-95.44%)	**0.02561** (-95.17%)
	TFormer	0.48880 (-6.70%)	5.62934 (-6.14%)	0.47401 (-10.57%)

Table 8.7: Errors of different models on seen and unseen PDE parameters for 1D Burger's equation. The unseen PDE parameters are highlighted in light grey.

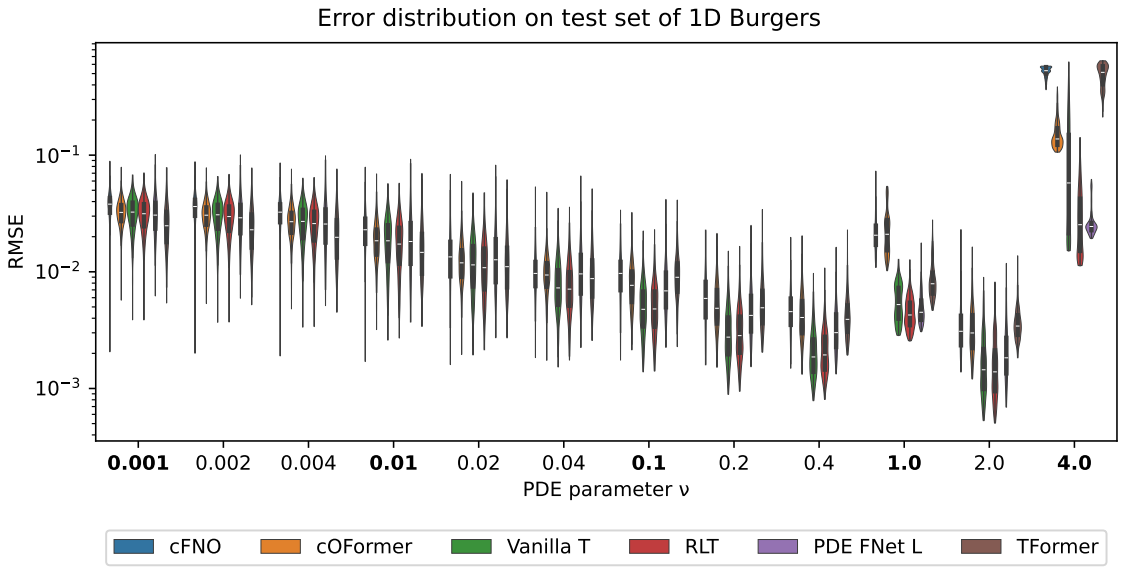


Figure 8.15: Error distribution of all samples in the test set of 1D Burgers. The PDE parameters printed in bold denote the unseen PDE parameters.

1D Advection

Secondly, we train and test the models on 1D Advection. It is clearly visible that all models struggle to generalize to the unseen PDE parameters. For the parameter $\beta = 0.1$ cFNO and OFormer perform much better than our proposed models, and for the PDE parameter $\beta = 7.0$, cOFormer produced invalid predictions (i.e., NaN). This experiment indicates that PDE parameter generalization could be different from PDE to PDE depending on how the PDE parameter influences the evolution.

PDE Parameter	Model	RMSE	nRMSE	bRMSE
$\beta = 0.1$	cFNO	**0.13815**	**0.22822**	**0.13538**
	cOFormer	0.21770 (+57.58%)	0.35954 (+57.54%)	0.19703 (+45.54%)
	Vanilla T	0.56658 (+310.12%)	0.85041 (+272.63%)	0.44764 (+230.66%)
	RLT	0.56903 (+311.89%)	0.85107 (+272.92%)	0.44764 (+230.66%)
	PDE FNet L	0.54161 (+292.04%)	0.81292 (+256.20%)	0.43243 (+219.43%)
	TFormer	0.56060 (+305.79%)	0.84121 (+268.60%)	0.44122 (+225.92%)
$\beta = 0.2$	cFNO	0.01803	0.03116	0.02886
	cOFormer	0.01464 (-18.77%)	0.02415 (-22.50%)	0.01497 (-48.13%)
	Vanilla T	0.02131 (+18.21%)	0.03773 (+21.08%)	0.01623 (-43.75%)
	RLT	**0.01353** (-24.95%)	**0.02388** (-23.37%)	**0.00928** (-67.86%)
	PDE FNet L	0.01710 (-5.11%)	0.03061 (-1.77%)	0.01587 (-45.00%)
	TFormer	0.03306 (+83.39%)	0.05410 (+73.62%)	0.03205 (+11.05%)

$\beta = 0.4$	cFNO	0.01747	0.03049	0.02997
	cOFormer	**0.01266** (-27.56%)	**0.02125** (-30.30%)	0.01157 (-61.41%)
	Vanilla T	0.01871 (+7.06%)	0.03437 (+12.71%)	0.01553 (-48.18%)
	RLT	0.01306 (-25.25%)	0.02338 (-23.32%)	**0.00961** (-67.94%)
	PDE FNet L	0.01742 (-0.33%)	0.03129 (+2.64%)	0.01638 (-45.34%)
	TFormer	0.03379 (+93.40%)	0.05575 (+82.86%)	0.03533 (+17.89%)
$\beta = 0.7$	cFNO	0.01982	0.03427	0.03148
	cOFormer	**0.01172** (-40.86%)	**0.01975** (-42.37%)	0.01332 (-57.69%)
	Vanilla T	0.01890 (-4.61%)	0.03475 (+1.38%)	0.01592 (-49.41%)
	RLT	0.01329 (-32.94%)	0.02382 (-30.49%)	**0.01004** (-68.09%)
	PDE FNet L	0.01803 (-9.01%)	0.03239 (-5.51%)	0.01692 (-46.23%)
	TFormer	0.03752 (+89.31%)	0.06206 (+81.06%)	0.03779 (+20.06%)
$\beta = 1.0$	cFNO	**0.45158**	**0.71939**	**0.39199**
	cOFormer	1.68027 (+272.09%)	3.02688 (+320.75%)	1.43922 (+267.16%)
	Vanilla T	0.60066 (+33.01%)	0.89767 (+24.78%)	0.51701 (+31.89%)
	RLT	0.62245 (+37.84%)	0.92993 (+29.27%)	0.53317 (+36.02%)
	PDE FNet L	0.53243 (+17.90%)	0.79934 (+11.11%)	0.45746 (+16.70%)
	TFormer	0.58862 (+30.35%)	0.87474 (+21.59%)	0.51016 (+30.15%)
$\beta = 2.0$	cFNO	0.02787	0.04855	0.03794
	cOFormer	**0.01167** (-58.13%)	**0.01962** (-59.60%)	0.01219 (-67.87%)
	Vanilla T	0.01851 (-33.58%)	0.03441 (-29.14%)	0.01605 (-57.69%)
	RLT	0.01312 (-52.92%)	0.02342 (-51.77%)	**0.01029** (-72.87%)
	PDE FNet L	0.01903 (-31.69%)	0.03387 (-30.24%)	0.01748 (-53.92%)
	TFormer	0.04998 (+79.36%)	0.08228 (+69.47%)	0.05091 (+34.20%)
$\beta = 4.0$	cFNO	0.02437	0.03955	0.03653
	cOFormer	**0.01174** (-51.84%)	**0.01931** (-51.17%)	0.01168 (-68.01%)
	Vanilla T	0.01814 (-25.57%)	0.03347 (-15.38%)	0.01583 (-56.65%)
	RLT	0.01297 (-46.77%)	0.02275 (-42.48%)	**0.01023** (-72.01%)
	PDE FNet L	0.01798 (-26.21%)	0.03188 (-19.40%)	0.01508 (-58.71%)
	TFormer	0.05964 (+144.73%)	0.09731 (+146.02%)	0.06221 (+70.31%)
$\beta = 7.0$	cFNO	4.02187	7.28715	3.71412
	cOFormer	N/A	N/A	N/A (1)
	Vanilla T	0.61439 (-84.72%)	0.90613 (-87.57%)	0.52418 (-85.89%)
	RLT	**0.52842** (-86.86%)	**0.81613** (-88.80%)	**0.43363** (-88.32%)
	PDE FNet L	0.69885 (-82.62%)	1.10154 (-84.88%)	0.59522 (-83.97%)
	TFormer	0.59857 (-85.12%)	0.92697 (-87.28%)	0.52201 (-85.95%)

Table 8.8: Errors of different models on seen and unseen PDE parameters for 1D Advection. The unseen PDE parameters are highlighted in light grey. (1) The values for cOFormer and $\beta = 7.0$ are not available since the model produced an invalid error (NaN).

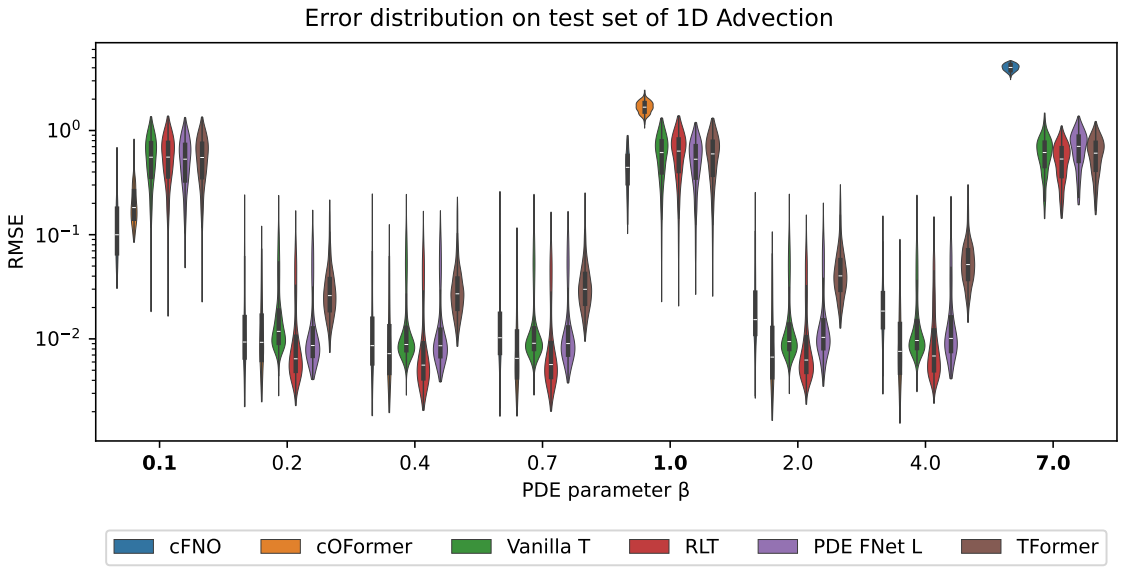


Figure 8.16: Error distribution of all samples in the test set of 1D Advection. The PDE parameters printed in bold denote the unseen PDE parameters.

1D CFD

We also train and test the models on 1D CFD. The baseline and proposed models perform well and generalize to the unseen PDE parameters. In general, our models perform slightly better than cFNO and cOFormer for seen as well as unseen PDE parameters.

PDE Parameter	Model	RMSE	nRMSE	bRMSE
$\eta = \zeta = 1.e-8$	cFNO	2.58155	0.44192	2.04999
	cOFormer	3.02712 (+17.26%)	0.49915 (+12.95%)	2.49086 (+21.51%)
	Vanilla T	1.51246 (-41.41%)	0.35558 (-19.54%)	1.10144 (-46.27%)
	RLT	**1.39417** (-45.99%)	0.35618 (-19.40%)	**0.99421** (-51.50%)
	PDE FNet L	1.61054 (-37.61%)	0.37169 (-15.89%)	1.17303 (-42.78%)
	TFormer	1.60045 (-38.00%)	**0.27225** (-38.39%)	1.29771 (-36.70%)
$\eta = \zeta = 0.001$	cFNO	2.51804	0.44073	1.98166
	cOFormer	2.98272 (+18.45%)	0.49961 (+13.36%)	2.44187 (+23.22%)
	Vanilla T	1.47247 (-41.52%)	0.35629 (-19.16%)	1.05734 (-46.64%)
	RLT	**1.35471** (-46.20%)	0.35773 (-18.83%)	**0.95569** (-51.77%)
	PDE FNet L	1.56243 (-37.95%)	0.37346 (-15.26%)	1.12648 (-43.15%)
	TFormer	1.56350 (-37.91%)	**0.27466** (-37.68%)	1.26045 (-36.39%)

$\eta = \zeta = 0.004$	cFNO	2.35751	0.43956	1.86015
	cOFormer	2.83530 (+20.27%)	0.50092 (+13.96%)	2.32945 (+25.23%)
	Vanilla T	1.26284 (-46.43%)	0.34836 (-20.75%)	0.90945 (-51.11%)
	RLT	**1.14453** (-51.45%)	0.35055 (-20.25%)	**0.80990** (-56.46%)
	PDE FNet L	1.36383 (-42.15%)	0.36813 (-16.25%)	0.98626 (-46.98%)
	TFormer	1.37717 (-41.58%)	**0.26320** (-40.12%)	1.12718 (-39.40%)
$\eta = \zeta = 0.007$	cFNO	2.22366	0.44008	1.76068
	cOFormer	2.71108 (+21.92%)	0.50343 (+14.40%)	2.23511 (+26.95%)
	Vanilla T	1.10451 (-50.33%)	0.34216 (-22.25%)	0.79643 (-54.77%)
	RLT	**0.99061** (-55.45%)	0.34387 (-21.86%)	**0.70170** (-60.15%)
	PDE FNet L	1.21495 (-45.36%)	0.36449 (-17.18%)	0.88096 (-49.96%)
	TFormer	1.24004 (-44.23%)	**0.25506** (-42.04%)	1.02836 (-41.59%)
$\eta = \zeta = 0.01$	cFNO	2.16902	0.43798	1.73917
	cOFormer	2.64474 (+21.93%)	0.50261 (+14.76%)	2.19530 (+26.23%)
	Vanilla T	1.01409 (-53.25%)	0.33234 (-24.12%)	0.74767 (-57.01%)
	RLT	**0.90573** (-58.24%)	0.33160 (-24.29%)	**0.65217** (-62.50%)
	PDE FNet L	1.13759 (-47.55%)	0.35488 (-18.97%)	0.83687 (-51.88%)
	TFormer	1.17383 (-45.88%)	**0.24642** (-43.74%)	0.98981 (-43.09%)
$\eta = \zeta = 0.04$	cFNO	1.55144	0.47518	1.26527
	cOFormer	2.02555 (+30.56%)	0.54523 (+14.74%)	1.69482 (+33.95%)
	Vanilla T	0.60103 (-61.26%)	0.32090 (-32.47%)	0.42921 (-66.08%)
	RLT	**0.53251** (-65.68%)	0.30997 (-34.77%)	**0.37033** (-70.73%)
	PDE FNet L	0.71688 (-53.79%)	0.36654 (-22.86%)	0.52043 (-58.87%)
	TFormer	0.75831 (-51.12%)	**0.23247** (-51.08%)	0.65429 (-48.29%)
$\eta = \zeta = 0.07$	cFNO	1.39346	0.51188	1.15472
	cOFormer	1.80251 (+29.36%)	0.58325 (+13.94%)	1.50776 (+30.57%)
	Vanilla T	0.52379 (-62.41%)	0.32858 (-35.81%)	0.37487 (-67.54%)
	RLT	**0.46030** (-66.97%)	0.30654 (-40.11%)	**0.31894** (-72.38%)
	PDE FNet L	0.62423 (-55.20%)	0.38370 (-25.04%)	0.45092 (-60.95%)
	TFormer	0.64916 (-53.41%)	**0.23221** (-54.64%)	0.56360 (-51.19%)
$\eta = \zeta = 1.0$	cFNO	1.37875	0.53848	1.15472
	cOFormer	1.72455 (+25.08%)	0.62663 (+16.37%)	1.44018 (+24.72%)
	Vanilla T	0.50313 (-63.51%)	0.34896 (-35.19%)	0.36573 (-68.33%)
	RLT	**0.43985** (-68.10%)	0.31424 (-41.64%)	**0.30727** (-73.39%)
	PDE FNet L	0.59589 (-56.78%)	0.40086 (-25.56%)	0.43143 (-62.64%)
	TFormer	0.61170 (-55.63%)	**0.23482** (-56.39%)	0.53171 (-53.95%)

Table 8.9: Errors of different models on seen and unseen PDE parameters for 1D CFD. The unseen PDE parameters are highlighted in light grey.

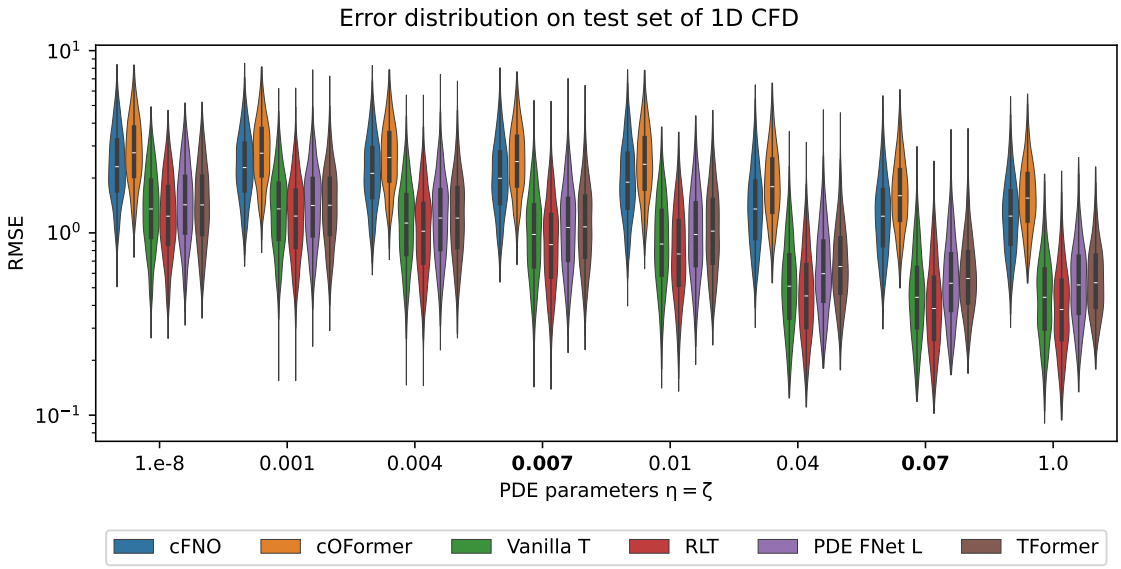


Figure 8.17: Error distribution of all samples in the test set of 1D CFD. The PDE parameters printed in bold denote the unseen PDE parameters.

8.3 Comparison of Inference Times

We also measure the inference times of the selected baselines and compare them against our 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 initial conditions, predicting 40, 80 and 160 timesteps in the future) with a batch size of 64. The spatial resolution is $s = 256$. The GPU is warmed up, and each measurement is repeated 20 times. Table 8.10 shows the mean values and standard deviations of the inference times.

Linear Attention vs. Scaled Dot-Product Attention

First, we compare the inference times and memory consumption of the Recurrent Linear Transformer (RLT) against the Vanilla Transformer with Scaled Dot-Product Attention. If we compare the RLT against the PyTorch implementation of Vanilla Transformers, the RLT is much slower than Vanilla Transformer. Increasing the number of prediction steps (i.e., the number of tokens) closes the gap between RLT and Vanilla Transformer, but for 160 prediction steps, Vanilla Transformer is still faster than RLT. We can observe this behaviour as the PyTorch implementation of Vanilla Transformers is heavily optimized to reduce memory consumption and computational time, while the implementation of the Recurrent Linear Transformer isn't optimized. It's a pure PyTorch implementation without custom CUDA kernels. Therefore, RLT is slower than Vanilla Transformer in our experiments. The GPU memory consumptions are almost identical. Only for 160 predicted timesteps, the memory consumption of the Vanilla Transformer is much higher than compared to RLT. We assume that the gap between the memory consumption of RLT and Vanilla Transformer will become larger if we increase the number of prediction steps.

To make a fair comparison, we also benchmark a pure PyTorch implementation of a Vanilla Transformer (named naive Vanilla Transformer in the table). We use the Vanilla Transformer implementation of Katharopoulos et al. [KVPF20]. Now, the results look different and the Recurrent Linear Transformer is faster than the Vanilla Transformer. Predicting 160 timesteps also shows that the memory consumption of non-optimized Vanilla Attention is much higher than for the Recurrent Linear Transformer.

Baseline Models vs. Proposed Models

We also look at the inference times and memory consumption of the baseline models for solving PDEs. Our models are slower than FNO, but faster than Galerkin Transformer and OFormer when doing inference. FNO is a non-transformer-based model which is very lightweight and fast. Therefore, it is hard to beat its inference times. It is also noticeable that the parallel predictions of TFormer provide some speed-up, but require much more GPU memory.

Prediction steps	Model	Inference time	GPU Memory consumption
40	FNO	$917.77^{\pm 2.51}$ ms	1665 MiB
	Galerkin Transformer	$2415.99^{\pm 54.56}$ ms	1535 MiB
	OFormer	$6025.75^{\pm 12.75}$ ms	1897 MiB
	PyTorch Vanilla Transformer (*)	$1220.23^{\pm 39.27}$ ms	1463 MiB
	Naive Vanilla Transformer (*)	$2274.33^{\pm 56.73}$ ms	1463 MiB
	Recurrent LT (*)	$2293.71^{\pm 53.17}$ ms	1461 MiB
	PDE FNet S (*)	$1488.16^{\pm 40.29}$ ms	1463 MiB
	PDE FNet L (*)	$1912.39^{\pm 58.39}$ ms	1469 MiB
	TFormer (*)	$2244.04^{\pm 6.65}$ ms	5625 MiB
80	FNO	$1912.19^{\pm 56.03}$ ms	1665 MiB
	Galerkin Transformer	$4940.80^{\pm 89.44}$ ms	1545 MiB
	OFormer	$12081.98^{\pm 19.39}$ ms	1897 MiB
	PyTorch Vanilla Transformer (*)	$2529.17^{\pm 40.95}$ ms	1495 MiB
	Naive Vanilla Transformer (*)	$4583.54^{\pm 93.67}$ ms	1517 MiB
	Recurrent LT (*)	$4527.53^{\pm 62.84}$ ms	1461 MiB
	PDE FNet S (*)	$2985.02^{\pm 60.23}$ ms	1489 MiB
	PDE FNet L (*)	$3849.17^{\pm 75.27}$ ms	1493 MiB
	TFormer (*)	$4422.65^{\pm 4.11}$ ms	10185 MiB
160	FNO	$3733.10^{\pm 62.94}$ ms	1665 MiB
	Galerkin Transformer	$9888.65^{\pm 125.72}$ ms	1547 MiB
	OFormer	$24108.24^{\pm 6.45}$ ms	1897 MiB
	PyTorch Vanilla Transformer (*)	$7396.29^{\pm 52.39}$ ms	2103 MiB
	Naive Vanilla Transformer (*)	$10717.40^{\pm 116.45}$ ms	2721 MiB
	Recurrent LT (*)	$8962.91^{\pm 112.41}$ ms	1501 MiB
	PDE FNet S (*)	$6042.62^{\pm 98.89}$ ms	1519 MiB
	PDE FNet L (*)	$7741.35^{\pm 104.28}$ ms	1533 MiB
	TFormer (*)	$6084.04^{\pm 9.37}$ ms	18871 MiB

Table 8.10: Inference time of different models trained and evaluated on a spatial resolution of 256, predicting 40, 80, and 160 timesteps in future. Our proposed models are marked with (*).

9 Model Analysis

We analyze the attention matrix of the Vanilla Transformer to get better insights into how the model works. A motivation for the first class of models or base architecture (emulating numerical PDE solver and using the entire historical context) was that using the entire history could help the model to improve the error. In the next section, we confirm this motivation by utilizing the attention matrix.

9.1 Attention Matrix

The attention scores of the attention mechanism provide insights into how the Transformer model works by indicating which timesteps are considered important. As shown in Section 2.3.1, the attention scores are the weights for the weighted sum of the tokens. The higher the attention score or weight, the higher the influence of the token on the context vector. This implies that if a token (i.e., timestep) has a high attention score, the timestep will likely help the model to predict a future timestep. Otherwise, the model wouldn't assign a high attention score to the timestep if it doesn't help to reduce the prediction error.

Figures 9.1, 9.2 and 9.3 show the attention matrices of the Vanilla Transformer on the test set of 1D Burgers's equation, 1D Advection and 1D CFD, respectively. The x-axis represents the context (keys), the y-axis represents the prediction (query), and the colour coding indicates the attention score between the key and the query. Context 0 (first token) is the grid, context 1 (second token) is the initial condition, context 2 (third token) is the first prediction and so on. Since the model works in an autoregressive fashion, the attention matrix has the shape of a triangular matrix. The values are computed by averaging the attention scores of all attention heads (8 heads in total) and by averaging the scores of 64 random samples from the test set.

1D Burgers and 1D Advection

The attention matrices for 1D Burgers and 1D Advection show that the earlier layers (layers 1 to 3) assign a high attention score to the first token (grid) and second token (initial condition). They also assign a high attention score to the latest ≈ 10 timesteps. It follows that these layers capture information from the initial condition and the latest historical predictions. The later layers (layers 4 to 6) assign a high attention score to the first input token and the remaining input tokens have approximately the same scores. It is difficult to interpret the meaning of the input tokens of these later layers, especially to distinguish from which tokens they already contain information. The previous transformer blocks or layers allow information flow from one token to another and the later the layer, the more likely that a token already contains information of other tokens. Figure 9.4 illustrates that information from one token can flow to different tokens and that token $x_0^{(6)}$, used as

input for the last layer, could also contain information from other tokens x_i with $0 \leq i \leq N$. Thus, it might be possible that the first input token doesn't represent the grid anymore. For instance, the model could use the first input token to store useful information in it and, therefore, the model assigns a high attention weight to it. The attention scores of the other tokens are presumably almost the same because the input tokens of the latest layer already contain information about the other tokens.

1D CFD

For 1D CFD the attention matrix reveals that the first layers (layers 1 to 3) pay attention to the initial condition and barely to previous predictions. The later layers (layers 4 to 6) seem to pay slightly more attention to the latest ≈ 10 timesteps. This could be an indicator that it depends on the PDE whether the historical context is useful or not.

To summarize, the attention matrix shows that the model pays attention to the provided historical context for 1D Burgers and 1D Advection which implies that historical context can help the model to reduce the error. It also shows that the initial condition could also provide some useful information for the model.

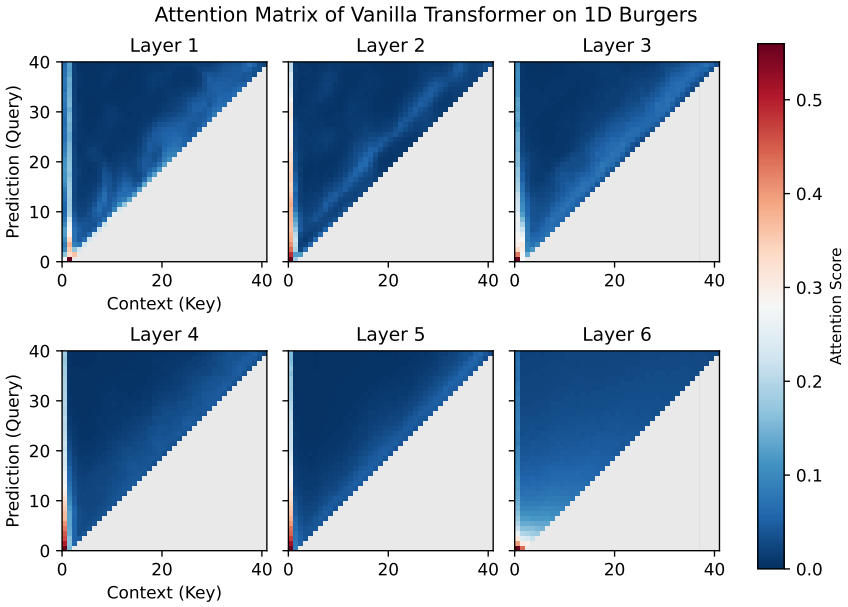


Figure 9.1: Attention matrix of the Vanilla Transformer on the 1D Burgers's equation. The attention scores are the mean values of all attention heads and 64 random samples of our 1D Burgers test set.

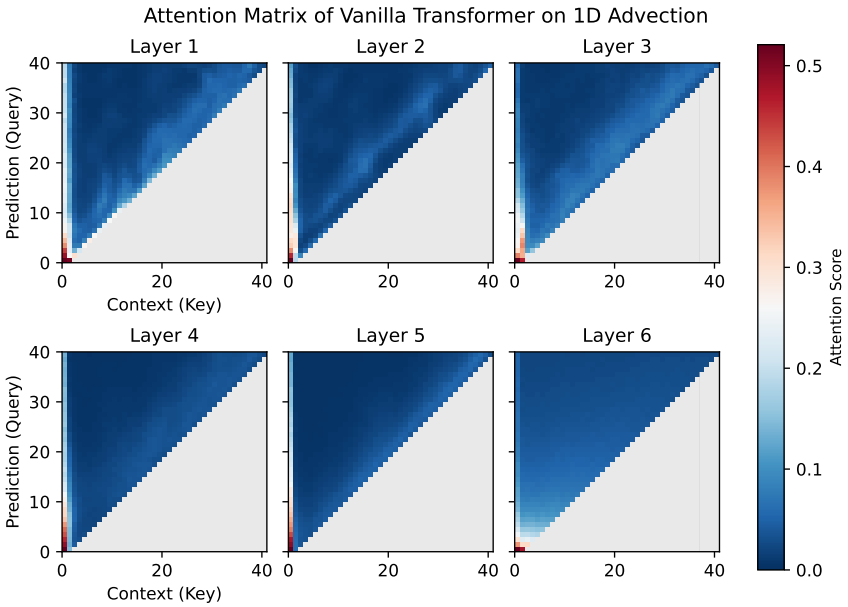


Figure 9.2: Attention matrix of the Vanilla Transformer on the 1D Advection equation. The attention scores are the mean values of all attention heads and 64 random samples of our 1D Advection test set.

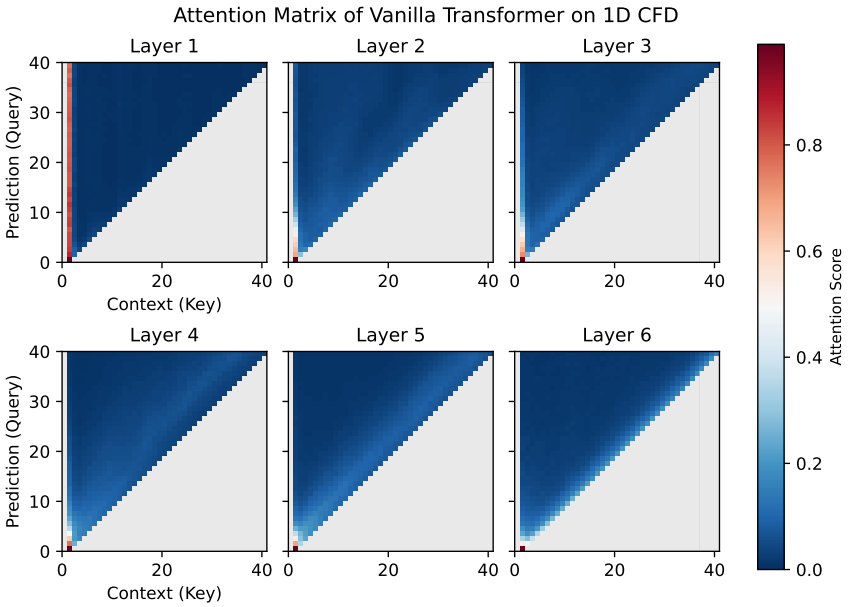


Figure 9.3: Attention matrix of the Vanilla Transformer on the 1D CFD equation. The attention scores are the mean values of all attention heads and 64 random samples of our 1D CFD test set.

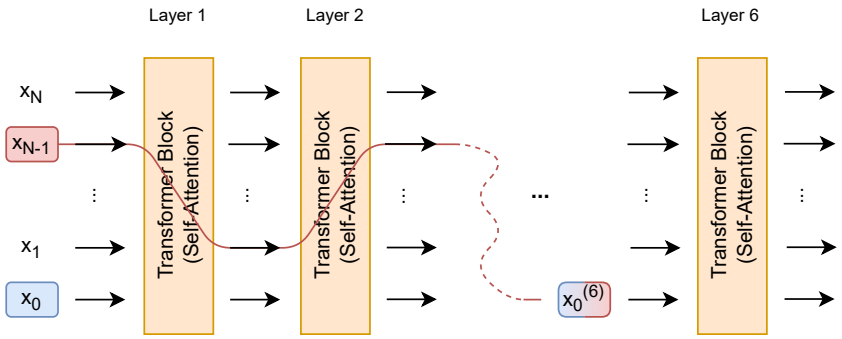


Figure 9.4: Information from one token or timestep (here x_{N-1}) can attend to different tokens (here x_0) because of the Self-Attention mechanism in the Transformer blocks or layers. Thus, token $x_0^{(6)}$, which is the output of the previous five layers and the input of the latest layer, does not necessarily only contain information from token x_0, but also from different tokens such as x_{N-1}. Therefore, it is difficult to interpret the meaning of $x_0^{(6)}$, especially to distinguish from which tokens it already contains information.

10 Discussion

In this section, we give a brief overview of the results and bring the empirical results from the previous Chapter 8 and the model analysis from Chapter 9 together. In general, the developed models provide a state-of-the-art method for solving 1D parametric PDEs. The models work well for a single PDE parameter as well as for multiple PDE parameters.

Single PDE Parameter All models, except for the Galerkin Transformer, are in the same order of magnitude of the errors. Our models are often slightly better than FNO, Galerkin Transformer, and OFormer. In our experiments, the Galerkin Transformer has the highest error for all three PDEs. It is also noteworthy that the errors of PDE FNet small and FNO are in the same order of magnitude, even though PDE FNet has fewer parameters than FNO. So, our proposed models offer an alternative to FNO, Galerkin Transformer, and OFormer for solving a PDE with a fixed PDE parameter.

Multiple PDE Parameters In general, all models generalize to the unseen PDE parameters $\nu \in \{0.001, 0.01, 0.1, 1.0\}$ for 1D Burgers. The parameter $\nu = 4.0$ is in the extrapolation regime for large parameter values and seems to be problematic since all models have a higher error. The errors of the Recurrent Linear Transformer and PDE FNet are lower than compared against the remaining models for the PDE parameter $\nu = 4.0$. For 1D Advection, all models seem to struggle to generalize to unseen PDE parameters. The errors for the unseen PDE parameters are noticeably higher than compared against the seen PDE parameters. For 1D CFD, the errors of all models are in the same order of magnitude for seen and unseen PDE parameters. Takamoto et al. [TAN23] observed the same effect that the models can generalize to unseen PDE parameters for 1D Burgers and CFD, but not for 1D Advection. To summarize, the models can generalize to unseen PDE parameters, but it seems to depend on the PDE.

Zero-Shot Super-Resolution In our experiments, FNO wasn't able to do spatial zero-shot super-resolution for 1D Burgers and 1D Advection, while OFormer and TFormer could do it. Thus, applying spatial Self-Attention seems to be a good idea to enable this capability. TFormer was also able to do temporal zero-shot super-resolution for 1D Burgers and 1D Advection. The errors of TFormer for 1D CFD increase if the temporal resolution is increased which indicates that the model struggles for 1D CFD.

Long Rollouts The proposed sliding window approach for the Recurrent Linear Transformer and Vanilla Transformer as well as omitting positional encoding for PDE FNet allows using the models for predicting long rollouts. The errors of the baseline and our models are in the same order

of magnitude and our models have a slightly lower error than FNO and OFormer. Only TFormer suffers when used for long rollouts since the query time t gets out-of-distribution. Therefore, the model should be used autoregressively to avoid an error explosion in the extrapolation regime.

Using Entire History as Context It seems that the models that emulate a numerical PDE solver (i.e., Recurrent Linear Transformer, Vanilla Transformer, and PDE FNet) benefit from using the entire history as input. The attention matrix shows that Vanilla Transformer assigns high attention weights to the initial condition (1D Burgers, 1D Advection, and 1D CFD) and to the latest ≈ 10 timesteps (1D Burgers and 1D Advection). Thus, the model uses the historical context and gains knowledge from it which helps to improve the accuracy.

Fourier Transform vs. Self-Attention The PDE FNet, the model that uses a 1D Fourier transform instead of Self-Attention, works surprisingly well for solving 1D PDEs. This might be an indicator that Self-Attention is not always necessary and that it could be replaced with simpler mechanisms for mixing tokens.

Inference Times FNO is a very lightweight and fast model. Therefore, it is very hard to achieve the inference times of it. Our proposed models are slower than FNO when doing inference, but faster than the two transformer-based baselines. OFormer and TFormer are technically speaking similar since both models encode a solution point in a latent space of dimensionality $d = 96$ and apply spatial Self-Attention. The parallel predictions of TFormer allow the model to be ≈ 3x faster than OFormer. However, bear in mind that this speed-up costs a lot of GPU memory. Our experiments also show that Linear Transformers are faster and more memory efficient than Vanilla Transformers, but much slower than a heavily optimized Vanilla Transformer implementation with custom CUDA kernels (e.g., PyTorch implementation).

11 Future Work

The results demonstrate that our models are competitive with others. Therefore, we believe that the proposed models are a serious alternative to FNO, Galerkin Transformer, and OFormer for solving PDEs. Nonetheless, the work raised several research questions which should be followed up to further enhance the models.

Our work only considered 1D PDEs and completely neglected 2D and 3D PDEs. However, many real-world scenarios are modelled in 2D or 3D space and require solving 2D or 3D PDEs, respectively. For instance, numerical weather forecasting or simulating the aerodynamics of a vehicle relies on solving PDEs in 2D or 3D space. Therefore, future work includes adapting the proposed models for solving 2D and 3D PDEs.

We think that the TFormer model has a lot of potential since it unites the idea of Neural Operators and Physics Informed Neural Networks (PINNs). A PDE-specific loss could be added to the TFormer to incorporate physical knowledge into the model. Since TFormer is similar to PINNs a PDE-specific loss could be easily added to the model. This would make TFormer more physics-aware and could help to improve the accuracy. Currently, the temporal embedding, which represents the prediction time t in the spatio-temporal latent representation, is generated with a simple linear layer. Using a more sophisticated method based on sine and cosine functions such as absolute positional encoding [VSP+17] could help to further improve the performance. Future work may also include adapting TFormer to use historical context (i.e., previous predictions) and to work in an autoregressive fashion.

Future work could also investigate the effect of different feature functions for the Recurrent Linear Transformer and the Linear Attention in the TFormer since we haven't experimented with different feature functions in this work. Inventing a feature function that is related to the task of PDE solving could also be part of the research.

Transformers show great success when pre-training them on a large dataset and then fine-tuning it on a smaller dataset [DCLT18]. Similar to the work of McCabe et al. [MBP+23] our models could be pre-trained on a large dataset and afterwards fine-tuned on a task-specific PDE. This direction of research could also include the comparison of different pre-training strategies such as masked reconstruction [DCLT18].

Takamoto et al. [TAN23] outlined that different training strategies could help to improve the performance of the models. Training the introduced models with different strategies and comparing different strategies would also be interesting for future work.

In future, it should also be tested whether the Recurrent Linear Transformer needs additional positional encoding or if the model works without it. Since the model is essentially a Recurrent Neural Network, it shouldn't be necessary to add positional encoding to the input tokens. This

would solve the length generalization problem of the model without any additional techniques such as the proposed sliding window approach. In addition, other techniques to solve the length generalization problems of Transformers for solving PDEs could also investigated.

The proposed PDE FNet Transformer showed promising performance for solving 1D PDEs and worked much better than the Vanilla FNet Transformer. Thus, it would be interesting to compare the accuracy of this model used in other domains. For example, in Natural Language Processing and other time-series tasks including forecasting the electricity consumption in a power grid.

We neglected the influence of the spatial and temporal resolutions and trained the model on a fixed spatial resolution of $s = 256$ and temporal resolution of $N = 41$ timesteps in total. Examining the effect of different spatial and temporal resolutions should be done in future work.

12 Conclusion

In this thesis, we investigated how Transformers could be utilized for solving 1D parametric PDEs and proposed four transformer-based models (from two different classes of models) for solving parametric PDEs. Our results demonstrate that our models perform competitively with state-of-the-art models for solving parametric PDEs. The results also show that models, trained on a chosen set of PDE parameters, can generalize to unseen PDE parameters. Thus, multiple models trained on specific parameters could be replaced with one single model. However, it seems that the generalization to unseen PDE parameters depends on the PDE, particularly on the influence of the parameter on the evolution of the trajectory.

The first class of models (base architecture) solves PDEs by emulating a numerical PDE solver. Recurrent Linear Transformer, Vanilla Transformer and PDE FNet are implementations of this class of models. These models apply temporal Self-Attention or a 1D Fourier transform (across different timesteps) to capture temporal dependencies which could help the model to predict the next timestep. The models receive the entire history as input for getting as much input as possible. Analyzing the attention matrix of the Transformer shows that the model uses the historical context. Consequently, providing more context to the model could improve the model's performance.

The TFormer is an implementation of the second class of models which models the underlying solution function u. This idea unites Neural Operators and Physics Informed Neural Networks and allows changing the spatial and temporal resolution (i.e., discretization) after the training. Each solution for time t is predicted independently which allows parallel predictions of multiple times t on the GPU. This makes the TFormer much faster than other transformer-based models for solving PDEs but requires much more GPU memory.

We also demonstrated that Linear Attention is faster than Vanilla Attention but slower than a heavily optimized Vanilla Transformer implementation (e.g., PyTorch implementation).

Bibliography

[ADH+21] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lucic, C. Schmid. "ViViT: A Video Vision Transformer". In: *CoRR* abs/2103.15691 (2021). arXiv: 2103.15691. URL: https://arxiv.org/abs/2103.15691 (cit. on p. 47).

[Aga18] A. F. Agarap. "Deep Learning using Rectified Linear Units (ReLU)". In: *CoRR* abs/1803.08375 (2018). arXiv: 1803.08375. URL: http://arxiv.org/abs/1803.08375 (cit. on p. 34).

[BKH16] J. L. Ba, J. R. Kiros, G. E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML] (cit. on p. 31).

[BWW22] J. Brandstetter, D. E. Worrall, M. Welling. "Message Passing Neural PDE Solvers". In: *CoRR* abs/2202.03376 (2022). arXiv: 2202.03376. URL: https://arxiv.org/abs/2202.03376 (cit. on pp. 21, 23).

[Cao21] S. Cao. "Choose a Transformer: Fourier or Galerkin". In: *CoRR* abs/2105.14995 (2021). arXiv: 2105.14995. URL: https://arxiv.org/abs/2105.14995 (cit. on pp. 21, 40, 43, 55, 65, 67–69, 74, 79).

[CLR+21] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, I. Mordatch. "Decision Transformer: Reinforcement Learning via Sequence Modeling". In: *CoRR* abs/2106.01345 (2021). arXiv: 2106.01345. URL: https://arxiv.org/abs/2106.01345 (cit. on p. 21).

[CT65] J. W. Cooley, J. W. Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of Computation* 19 (1965), pp. 297–301. URL: https://api.semanticscholar.org/CorpusID:121744946 (cit. on p. 36).

[CUH15] D.-A. Clevert, T. Unterthiner, S. Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015) (cit. on p. 34).

[DBK+20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929 (cit. on pp. 21, 26).

[DCLT18] J. Devlin, M. Chang, K. Lee, K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805 (cit. on pp. 21, 26, 109).

[GQC+20] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, R. Pang. *Conformer: Convolution-augmented Transformer for Speech Recognition*. 2020. arXiv: 2005.08100 [eess.AS] (cit. on pp. 21, 26).

[GZ22] N. Geneva, N. Zabaras. "Transformers for modeling physical systems". In: *Neural Networks* 146 (2022), pp. 272–289. DOI: 10.1016/j.neunet.2021.11.022. URL: https://doi.org/10.1016/j.neunet.2021.11.022 (cit. on pp. 43–45).

[HGP+22] X. Han, H. Gao, T. Pfaff, J. Wang, L. Liu. "Predicting Physics in Mesh-reduced Space with Temporal Attention". In: *CoRR* abs/2201.09113 (2022). arXiv: 2201.09113. URL: https://arxiv.org/abs/2201.09113 (cit. on p. 43).

[HS97] S. Hochreiter, J. Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (cit. on p. 25).

[HZRS15] K. He, X. Zhang, S. Ren, J. Sun. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (cit. on pp. 31, 39).

[JEP+21] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589 (cit. on p. 21).

[KLL+21] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, A. Anandkumar. "Neural Operator: Learning Maps Between Function Spaces". In: *CoRR* abs/2108.08481 (2021). arXiv: 2108.08481. URL: https://arxiv.org/abs/2108.08481 (cit. on p. 69).

[KVPF20] A. Katharopoulos, A. Vyas, N. Pappas, F. Fleuret. "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*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 5156–5165. URL: http://proceedings.mlr.press/v119/katharopoulos20a.html (cit. on pp. 21, 32, 34, 35, 74, 102).

[LAEO22] J. Lee-Thorp, J. Ainslie, I. Eckstein, S. Ontañón. "FNet: Mixing Tokens with Fourier Transforms". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*. Ed. by M. Carpuat, M. de Marneffe, I. V. M. Ruíz. Association for Computational Linguistics, 2022, pp. 4296–4313. DOI: 10.18653/v1/2022.naacl-main.319. URL: https://doi.org/10.18653/v1/2022.naacl-main.319 (cit. on pp. 21, 35, 53).

[LKA+20a] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar. *Neural Operator: Graph Kernel Network for Partial Differential Equations*. 2020. arXiv: 2003.03485 [cs.LG] (cit. on pp. 25, 65).

[LKA+20b] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. M. Stuart, A. Anandkumar. "Fourier Neural Operator for Parametric Partial Differential Equations". In: *CoRR* abs/2010.08895 (2020). arXiv: 2010.08895. URL: https://arxiv.org/abs/2010.08895 (cit. on pp. 21, 24, 39, 41, 50, 55, 65, 69, 70, 75, 79).

[LLC+18] S. Li, W. Li, C. Cook, C. Zhu, Y. Gao. "Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5457–5466. DOI: 10.1109/CVPR.2018.00572 (cit. on p. 25).

[LMF23] Z. Li, K. Meidani, A. B. Farimani. "Transformer for Partial Differential Equations' Operator Learning". In: *Trans. Mach. Learn. Res.* 2023 (2023). URL: https://openreview.net/forum?id=EPPqt3uERT (cit. on pp. 40, 43, 55, 65, 70, 74, 79).

[Lyn08] P. Lynch. "The origins of computer weather prediction and climate modeling". In: *Journal of computational physics* 227.7 (2008), pp. 3431–3444 (cit. on p. 21).

[LZK+21] Z. Li, H. Zheng, N. B. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, A. Anandkumar. "Physics-Informed Neural Operator for Learning Partial Differential Equations". In: *CoRR* abs/2111.03794 (2021). arXiv: 2111.03794. URL: https://arxiv.org/abs/2111.03794 (cit. on pp. 21, 24).

[MBP+23] M. McCabe, B. R.-S. Blancard, L. H. Parker, R. Ohana, M. Cranmer, A. Bietti, M. Eickenberg, S. Golkar, G. Krawezik, F. Lanusse, M. Pettee, T. Tesileanu, K. Cho, S. Ho. *Multiple Physics Pretraining for Physical Surrogate Models*. 2023. arXiv: 2310.02994 [cs.LG] (cit. on p. 109).

[NIGM18] C. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall. "Activation Functions: Comparison of trends in Practice and Research for Deep Learning". In: *CoRR* abs/1811.03378 (2018). arXiv: 1811.03378. URL: http://arxiv.org/abs/1811.03378 (cit. on p. 28).

[Olv+14] P. J. Olver et al. *Introduction to partial differential equations*. Vol. 1. Springer, 2014 (cit. on p. 21).

[PSV+17] E. Perez, F. Strub, H. de Vries, V. Dumoulin, A. C. Courville. "FiLM: Visual Reasoning with a General Conditioning Layer". In: *CoRR* abs/1709.07871 (2017). arXiv: 1709.07871. URL: http://arxiv.org/abs/1709.07871 (cit. on pp. 58, 74).

[PTA12] R. H. Pletcher, J. C. Tannehill, D. Anderson. *Computational fluid mechanics and heat transfer*. CRC press, 2012 (cit. on p. 21).

[RDG+23] A. Ruoss, G. Delétang, T. Genewein, J. Grau-Moya, R. Csordás, M. Bennani, S. Legg, J. Veness. *Randomized Positional Encodings Boost Length Generalization of Transformers*. 2023. arXiv: 2305.16843 [cs.LG] (cit. on pp. 32, 61, 90, 92).

[RFB15] O. Ronneberger, P. Fischer, T. Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV] (cit. on pp. 39, 41).

[RNSS+18] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. "Improving language understanding by generative pre-training". In: (2018) (cit. on p. 44).

[Roh21] B. Rohrer. *Transformers from Scratch*. Oct. 2021. URL: https://e2eml.school/transformers.html (cit. on p. 31).

[RPK19] M. Raissi, P. Perdikaris, G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational physics* 378 (2019), pp. 686–707 (cit. on pp. 25, 41, 55).

[SLP+23] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, Y. Liu. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. 2023. arXiv: 2104.09864 [cs.CL] (cit. on p. 70).

[TAN23] M. Takamoto, F. Alesiani, M. Niepert. "Learning Neural PDE Solvers with Parameter-Guided Channel Attention". In: *ICML* abs/2304.14118 (2023). DOI: 10.48550/arXiv.2304.14118. arXiv: 2304.14118. URL: https://doi.org/10.48550/arXiv.2304.14118 (cit. on pp. 21, 41, 71, 73, 79, 107, 109, 121).

[TBS+17] C. Trabelsi, O. Bilaniuk, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, C. J. Pal. “Deep Complex Networks”. In: *CoRR* abs/1705.09792 (2017). arXiv: 1705.09792. URL: http://arxiv.org/abs/1705.09792 (cit. on p. 51).

[TPL+22] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, M. Niepert. “PDEBench: An extensive benchmark for scientific machine learning”. In: *NeurIPS* (2022) (cit. on pp. 21, 22, 24, 41, 77).

[VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (cit. on pp. 21, 26, 29, 30, 48, 51, 53, 68, 69, 109).

All links were last followed on December 22, 2023.

A Architectures

A.1 Recurrent Linear Transformer for PDEs with Multiple Channels

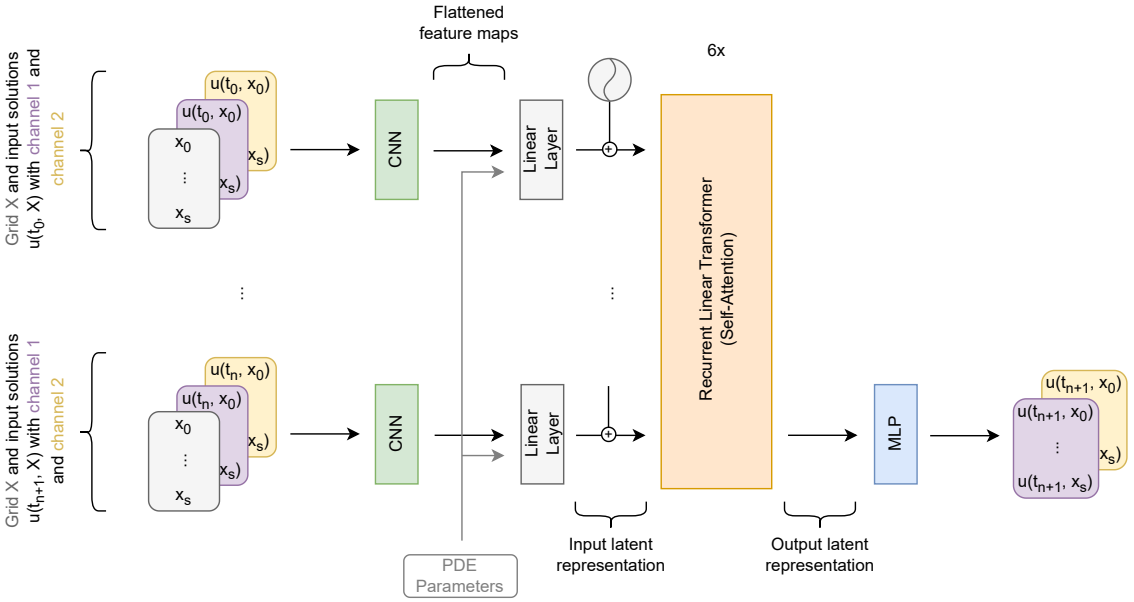


Figure A.1: Architecture of the Recurrent Linear Transformer for solving 1D PDEs with multiple input channels. A 1D Convolutional Neural Network (CNN) takes the grid X and $u(t_i, X) \in \mathbb{R}^{s \times c}$ as input and generates feature maps which are flattened and fed into a linear layer. The output of the linear layer is used as input latent representation for the Recurrent Linear Transformer that calculates the dynamics of the PDE. An MLP maps the output latent representation that represents the solution $u(t_{n+1}, X)$ back to the physical space.

B Loss Plots

B.1 Vanilla FNet Transformer vs. PDE FNet Transformer

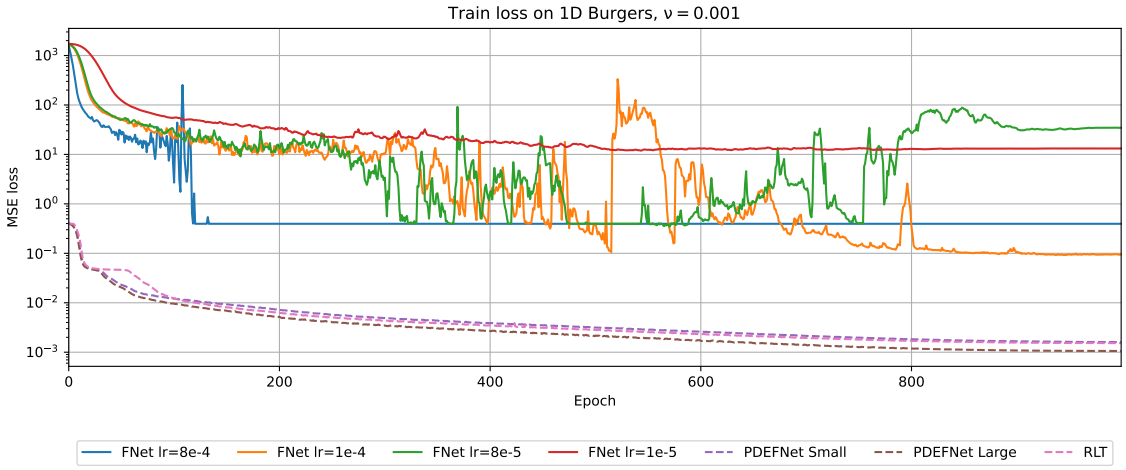


Figure B.1: Loss plot of the Vanilla FNet Transformer as well as the proposed PDE FNet and Recurrent Linear Transformer on the train set of 1D Burgers with $\nu = 0.001$.

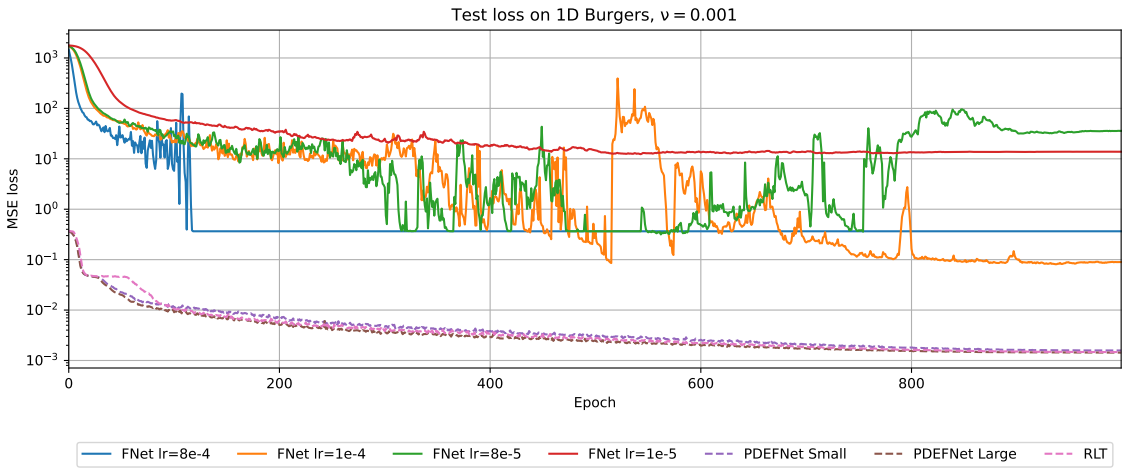


Figure B.2: Loss plot of the Vanilla FNet Transformer as well as the proposed PDE FNet and Recurrent Linear Transformer on the test set of 1D Burgers with $\nu = 0.001$.

C Benchmark

C.1 Experiments with Single PDE Parameter

C.1.1 Model's Hyperparameters

Table C.1 and Table C.2 show the hyperparameters used for the single PDE parameter experiments. OFormer was trained with a curriculum learning strategy because the model diverged without any additional tricks. We decided to use the curriculum learning strategy proposed by Takamoto et al. [TAN23]. The Vanilla Transformer uses Scaled Dot-Product Attention and has no final layer normalization compared to the Recurrent Linear Transformer with Linear Attention and a final layer normalization. Thus, the Vanilla Transformer has $96 \cdot 2$ parameters less. The proposed TFormer has 3 + 3 layers in total with 3 Linear Transformer layers and 3 spatio-temporal conditioning layers.

C.1.2 Benchmark Results

This section contains all metrics of PDEBench of the single PDE parameter experiments. The models were trained and evaluated five times on the random seeds 23, 42, 1729, 3407, and 1993. The shown values are the average over all five runs and the standard deviation. One single timestep was used as the initial condition and the remaining timesteps were either predicted in an autoregressive fashion or in one shot (TFormer). Table C.3, Table C.4, Table C.5 shows the full benchmark values for 1D Burgers, 1D Advection, and 1D CFD, respectively.

C.2 Experiments with Multiple PDE Parameters

C.2.1 Model's Hyperparameters

Table C.6 and Table C.7 show the hyperparameters used for the multiple PDE parameter experiments. We reduced the number of epochs to 250 for OFormer and 500 for the other models to reduce the training time to an acceptable level.

C.2.2 Benchmark Results

This section contains all metrics of PDEBench of the multiple PDE parameter experiments. The models were trained on a set of PDE Parameters and tested on a set that contains the seen and unseen PDE parameters. The shown values are the average over all five runs and the standard deviation.

One single timestep was used as the initial condition and the remaining timesteps were either predicted in an autoregressive fashion or in one shot (TFormer). Table C.8, Table C.9, Table C.8 shows the full benchmark values for 1D Burgers, 1D Advection, and 1D CFD, respectively.

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	549569	✗
1D Advection	FNO	500	64	64	16	4	1.e-4	Step Scheduler	549569	✗
1D CFD	FNO	500	64	64	16	4	6.e-5	Step Scheduler	549955	✗

Table C.1: 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	# Heads	# Layers	Learning rate	LR Scheduler	# Parameters	Curriculum learning
1D Burgers	OFormer	500	64	96	1	4+3	6.e-5	One Cycle Scheduler	660814	✓
	Galerkin Transformer	500	64	96	1	4+2	1.e-5	One Cycle Scheduler	530305	✗
	Vanilla Transformer	1000	64	96	8	6	8.e-4	One Cycle Scheduler	795744 (1)	✗
	RLT	1000	64	96	8	6	8.e-4	One Cycle Scheduler	795936	✗
	PDE FNet small	1000	64	96	-	6	8.e-4	One Cycle Scheduler	461664	✗
	PDE FNet large	1000	64	96	-	6	8.e-4	One Cycle Scheduler	795744	✗
	TFormer	500	32	96	8	3+3	3.e-4	One Cycle Scheduler	793825	✗
1D Advection	OFormer	500	64	96	1	4+3	6.e-5	One Cycle Scheduler	660814	✓
	Galerkin Transformer	500	64	96	1	4+2	1.e-5	One Cycle Scheduler	530305	✗
	Vanilla Transformer	1000	64	96	8	6	6.e-4 (2)	One Cycle Scheduler	795744	✗
	RLT	1000	64	96	8	6	8.e-4	One Cycle Scheduler	795936	✗
	PDE FNet small	1000	64	96	-	6	8.e-4	One Cycle Scheduler	461664	✗
	PDE FNet large	1000	64	96	-	6	8.e-4	One Cycle Scheduler	795744	✗
	TFormer	500	64	96	8	3+3	6.e-4	One Cycle Scheduler	793825	✗
1D CFD	OFormer	500	64	96	1	4+3	6.e-5	One Cycle Scheduler	662733	✓
	Galerkin Transformer	500	64	96	1	4+2	1.e-5	One Cycle Scheduler	530595	✗
	Vanilla Transformer	500	64	96	8	6	4.e-4 (2)	One Cycle Scheduler	765710 (1)	✗
	RLT	500	64	96	8	6	8.e-4	One Cycle Scheduler	765902	✗
	PDE FNet small	1000	64	96	-	6	8.e-4	One Cycle Scheduler	431630	✗
	PDE FNet large	1000	64	96	-	6	8.e-4	One Cycle Scheduler	765710	✗
	TFormer	500	64	96	8	3+3	4.e-4	One Cycle Scheduler	794307	✗

Table C.2: 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. (1) Vanilla Transformer has $96 \cdot 2$ parameters less because it has no final layer normalization. (2) The learning rate was reduced since a learning rate of 8.e-4 leads to instabilities and divergence.

Metrics of 1D Burgers' Equation with $\nu = 0.001$, spatial resolution of 256, and trajectory of 41 timesteps

	Baseline & Proposed Models							
Metric	FNO	OFormer	Galerkin T	Vanilla T	Recurrent LT	PDE FNet S	PDE FNet L	TFormer
RMSE	$0.03511^{\pm 0.00042}$	$0.03402^{\pm 0.00209}$	$0.06170^{\pm 0.00119}$	$\mathbf{0.03019}^{\pm 0.00025}$	$0.03110^{\pm 0.00018}$	$0.03287^{\pm 0.00009}$	$0.03127^{\pm 0.00020}$	$0.03080^{\pm 0.00093}$
nRMSE	$0.09821^{\pm 0.00110}$	$0.10326^{\pm 0.00841}$	$0.16545^{\pm 0.00361}$	$0.08308^{\pm 0.00061}$	$0.08583^{\pm 0.00052}$	$0.09199^{\pm 0.00026}$	$0.08947^{\pm 0.00078}$	$\mathbf{0.07977}^{\pm 0.00234}$
cRMSE	$0.00235^{\pm 0.00018}$	$0.00407^{\pm 0.00068}$	$0.02142^{\pm 0.00165}$	$\mathbf{0.00145}^{\pm 0.00002}$	$0.00173^{\pm 0.00007}$	$0.00209^{\pm 0.00004}$	$0.00254^{\pm 0.00003}$	$0.00585^{\pm 0.00034}$
max error	$0.59746^{\pm 0.00381}$	$0.71420^{\pm 0.01389}$	$0.84150^{\pm 0.00598}$	$\mathbf{0.57749}^{\pm 0.00927}$	$0.60536^{\pm 0.00932}$	$0.64465^{\pm 0.00469}$	$0.76039^{\pm 0.00983}$	$0.72029^{\pm 0.01894}$
bRMSE	$0.02177^{\pm 0.00083}$	$0.02144^{\pm 0.00217}$	$0.03658^{\pm 0.00102}$	$0.01704^{\pm 0.00033}$	$\mathbf{0.01495}^{\pm 0.00063}$	$0.01760^{\pm 0.00030}$	$0.01567^{\pm 0.00052}$	$0.02258^{\pm 0.00029}$
fRMSE low	$\mathbf{0.00213}^{\pm 0.00007}$	$0.00385^{\pm 0.00057}$	$0.01216^{\pm 0.00062}$	$0.00228^{\pm 0.00002}$	$0.00251^{\pm 0.00006}$	$0.00316^{\pm 0.00005}$	$0.00366^{\pm 0.00004}$	$0.00512^{\pm 0.00018}$
fRMSE mid	$0.00289^{\pm 0.00007}$	$0.00391^{\pm 0.00038}$	$0.00877^{\pm 0.00022}$	$\mathbf{0.00250}^{\pm 0.00006}$	$0.00272^{\pm 0.00004}$	$0.00331^{\pm 0.00004}$	$0.00356^{\pm 0.00004}$	$0.00441^{\pm 0.00017}$
fRMSE high	$0.00202^{\pm 0.00002}$	$0.00181^{\pm 0.00005}$	$0.00235^{\pm 0.00001}$	$0.00183^{\pm 0.00001}$	$0.00186^{\pm 0.00001}$	$0.00189^{\pm 0.00000}$	$0.00180^{\pm 0.00001}$	$\mathbf{0.00152}^{\pm 0.00003}$

Table C.3: Errors of the baseline and proposed models for different evaluation metrics, including RMSE, normalized RMSE (nRMSE), RMSE from conserved value (cRMSE), maximum error, RMSE at the boundaries (bRMSE), RMSE in Fourier space at low (fRMSE low), medium (fRMSE mid), and high frequency (fRMSE high) ranges on the 1D Burger's equation with a fixed PDE parameter.

Metrics of 1D Advecion' Equation with $\beta = 0.1$, spatial resolution of 256, and trajectory of 41 timesteps

	Baseline & Proposed Models							
Metric	FNO	OFormer	Galerkin T	Vanilla T	RLT	PDE FNet S	PDE FNet L	TFormer
RMSE	$0.01114^{\pm 0.00033}$	$0.00726^{\pm 0.00099}$	$0.03238^{\pm 0.00303}$	$0.00596^{\pm 0.00028}$	$\mathbf{0.00377}^{\pm 0.00016}$	$0.01186^{\pm 0.00133}$	$0.00791^{\pm 0.00020}$	$0.01030^{\pm 0.00102}$
nRMSE	$0.01901^{\pm 0.00052}$	$0.01252^{\pm 0.00159}$	$0.05576^{\pm 0.00576}$	$0.01072^{\pm 0.00043}$	$\mathbf{0.00695}^{\pm 0.00029}$	$0.02005^{\pm 0.00210}$	$0.01391^{\pm 0.00035}$	$0.01722^{\pm 0.00166}$
cRMSE	$0.00185^{\pm 0.00039}$	$0.00085^{\pm 0.00020}$	$0.02481^{\pm 0.00334}$	$0.00087^{\pm 0.00006}$	$\mathbf{0.00051}^{\pm 0.00001}$	$0.00163^{\pm 0.00058}$	$0.00127^{\pm 0.00005}$	$0.00235^{\pm 0.00025}$
max error	$0.28731^{\pm 0.00416}$	$0.18834^{\pm 0.01669}$	$0.31463^{\pm 0.02017}$	$0.19673^{\pm 0.00486}$	$\mathbf{0.13569}^{\pm 0.00405}$	$0.25657^{\pm 0.01042}$	$0.22555^{\pm 0.01181}$	$0.21518^{\pm 0.01979}$
bRMSE	$0.02366^{\pm 0.00048}$	$0.00805^{\pm 0.00130}$	$0.03575^{\pm 0.00223}$	$0.00453^{\pm 0.00036}$	$\mathbf{0.00250}^{\pm 0.00012}$	$0.01060^{\pm 0.00116}$	$0.00734^{\pm 0.00075}$	$0.00868^{\pm 0.00022}$
fRMSE low	$0.00125^{\pm 0.00018}$	$0.00071^{\pm 0.00015}$	$0.01197^{\pm 0.00160}$	$0.00086^{\pm 0.00007}$	$\mathbf{0.00044}^{\pm 0.00002}$	$0.00203^{\pm 0.00035}$	$0.00144^{\pm 0.00005}$	$0.00220^{\pm 0.00022}$
fRMSE mid	$0.00112^{\pm 0.00005}$	$0.00099^{\pm 0.00029}$	$0.00272^{\pm 0.00035}$	$0.00121^{\pm 0.00009}$	$\mathbf{0.00058}^{\pm 0.00004}$	$0.00257^{\pm 0.00022}$	$0.00205^{\pm 0.00010}$	$0.00225^{\pm 0.00025}$
fRMSE high	$0.00055^{\pm 0.00000}$	$0.00035^{\pm 0.00002}$	$0.00048^{\pm 0.00002}$	$0.00034^{\pm 0.00001}$	$\mathbf{0.00026}^{\pm 0.00001}$	$0.00042^{\pm 0.00002}$	$0.00034^{\pm 0.00001}$	$0.00032^{\pm 0.00001}$

Table C.4: Errors of the baseline and proposed models for different evaluation metrics, including RMSE, normalized RMSE (nRMSE), RMSE from conserved value (cRMSE), maximum error, RMSE at the boundaries (bRMSE), RMSE in Fourier space at low (fRMSE low), medium (fRMSE mid), and high frequency (fRMSE high) ranges on the 1D Advection equation with a fixed PDE parameter.

Metrics of 1D CFD with $\eta = \zeta = 0.007$, spatial resolution of 256, and trajectory of 41 timesteps

	Baseline & Proposed Models							
Metric	FNO	OFormer	Galerkin T	Vanilla T	Recurrent LT	PDE FNet S	PDE FNet L	TFormer
RMSE	$2.47824^{\pm 0.27980}$	$2.63904^{\pm 0.18728}$	$3.57612^{\pm 0.06263}$	$1.46042^{\pm 0.02527}$	$\mathbf{1.33055}^{\pm 0.05930}$	$2.07708^{\pm 0.20027}$	$1.64760^{\pm 0.15177}$	$1.55307^{\pm 0.01595}$
nRMSE	$0.52616^{\pm 0.07197}$	$0.51778^{\pm 0.13010}$	$0.70376^{\pm 0.03648}$	$0.40786^{\pm 0.00642}$	$0.39167^{\pm 0.00749}$	$0.58116^{\pm 0.05020}$	$0.45492^{\pm 0.02190}$	$\mathbf{0.29644}^{\pm 0.00341}$
cRMSE	$0.66707^{\pm 0.29749}$	$0.82767^{\pm 0.03220}$	$1.90882^{\pm 0.14092}$	$0.19383^{\pm 0.00702}$	$\mathbf{0.16482}^{\pm 0.00656}$	$0.29041^{\pm 0.03341}$	$0.26162^{\pm 0.04192}$	$0.45856^{\pm 0.04349}$
max error	$19.87561^{\pm 0.60100}$	$20.95084^{\pm 0.48996}$	$23.39379^{\pm 0.25445}$	$15.34628^{\pm 0.06312}$	$\mathbf{15.17232}^{\pm 0.17461}$	$18.66349^{\pm 0.91054}$	$16.80494^{\pm 0.71396}$	$15.37480^{\pm 0.11989}$
bRMSE	$2.02319^{\pm 0.30314}$	$2.14893^{\pm 0.15257}$	$3.02366^{\pm 0.06016}$	$1.09774^{\pm 0.01490}$	$\mathbf{0.97581}^{\pm 0.05123}$	$1.53763^{\pm 0.16826}$	$1.21210^{\pm 0.12224}$	$1.32100^{\pm 0.03547}$
fRMSE low	$0.83973^{\pm 0.11193}$	$0.95460^{\pm 0.06955}$	$1.35789^{\pm 0.03638}$	$0.35208^{\pm 0.00539}$	$\mathbf{0.33316}^{\pm 0.01130}$	$0.61690^{\pm 0.08817}$	$0.46007^{\pm 0.05827}$	$0.50049^{\pm 0.00777}$
fRMSE mid	$0.30163^{\pm 0.00352}$	$0.31097^{\pm 0.00608}$	$0.31153^{\pm 0.00078}$	$0.29166^{\pm 0.00509}$	$0.26627^{\pm 0.01026}$	$0.32175^{\pm 0.00489}$	$0.29463^{\pm 0.01179}$	$\mathbf{0.26190}^{\pm 0.00209}$
fRMSE high	$0.01287^{\pm 0.00069}$	$\mathbf{0.01214}^{\pm 0.00020}$	$0.01257^{\pm 0.00018}$	$0.01672^{\pm 0.00062}$	$0.01495^{\pm 0.00029}$	$0.02018^{\pm 0.00208}$	$0.01686^{\pm 0.00053}$	$0.01309^{\pm 0.00022}$

Table C.5: Errors of the baseline and proposed models for different evaluation metrics, including RMSE, normalized RMSE (nRMSE), RMSE from conserved value (cRMSE), maximum error, RMSE at the boundaries (bRMSE), RMSE in Fourier space at low (fRMSE low), medium (fRMSE mid), and high frequency (fRMSE high) ranges on the 1D CFD equation with fixed PDE parameters.

PDE	Model	Epochs	Batch size	Fourier width	# Fourier Modes	# Layers	Learning rate	LR Scheduler	# Parameters	Curriculum learning
1D Burgers	cFNO	500	256	64	16	4	1.e-4	Step Scheduler	549762	✗
1D Advection	cFNO	500	256	64	16	4	1.e-4	Step Scheduler	549762	✗
1D CFD	cFNO	500	256	64	16	4	6.e-5	Step Scheduler	550341	✗

Table C.6: Hyperparameters for the cFNO used in the multiple 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	# Heads	# Layers	Learning rate	LR Scheduler	# Parameters	Curriculum learning
1D Burgers	cOFormer	250	200	96	1	4+3	6.e-5	One Cycle Scheduler	660814	✓
	Vanilla Transformer	500	256	96	8	6	8.e-4	One Cycle Scheduler	795744 (1)	✗
	RLT	500	256	96	8	6	8.e-4	One Cycle Scheduler	795936	✗
	PDE FNet large	500	256	96	-	6	8.e-4	One Cycle Scheduler	795744	✗
	TFormer	500	128	96	8	3+3	6.e-4	One Cycle Scheduler	793825	✗
1D Advection	cOFormer	250	200	96	1	4+3	6.e-5	One Cycle Scheduler	660814	✓
	Vanilla Transformer	500	256	96	8	6	6.e-4	One Cycle Scheduler	795744	✗
	RLT	500	256	96	8	6	8.e-4	One Cycle Scheduler	795936	✗
	PDE FNet large	500	256	96	-	6	8.e-4	One Cycle Scheduler	795744	✗
	TFormer	500	128	96	8	3+3	6.e-4	One Cycle Scheduler	793825	✗
1D CFD	cOFormer	250	200	96	1	4+3	6.e-5	One Cycle Scheduler	662733	✓
	Vanilla Transformer	500	256	96	8	6	4.e-4	One Cycle Scheduler	765710	✗
	RLT	500	256	96	8	6	8.e-4	One Cycle Scheduler	765902	✗
	PDE FNet large	500	256	96	-	6	8.e-4	One Cycle Scheduler	765710	✗
	TFormer	500	256	96	8	3+3	4.e-4	One Cycle Scheduler	794307	✗

Table C.7: Hyperparameters for the transformer-based models used in the multiple 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. (1) Vanilla Transformer has $96 \cdot 2$ parameters less because it has no final layer normalization.

PDE Parameter	Model	RMSE	nRMSE	cRMSE	max error	bRMSE	fRMSE low	fRMSE mid	fRMSE high
$\nu = 0.001$	cFNO	0.03725$^{\pm 0.00056}$	0.10518$^{\pm 0.00159}$	0.00206$^{\pm 0.00018}$	0.57781$^{\pm 0.00581}$	0.02272$^{\pm 0.00089}$	0.00210$^{\pm 0.00005}$	0.00323$^{\pm 0.00011}$	0.00208$^{\pm 0.00002}$
	cOFormer	0.03236$^{\pm 0.00335}$	0.09582$^{\pm 0.01096}$	0.00300$^{\pm 0.00065}$	0.64431$^{\pm 0.02665}$	0.02059$^{\pm 0.00217}$	0.00307$^{\pm 0.00057}$	0.00375$^{\pm 0.00048}$	0.00175$^{\pm 0.00012}$
	Vanilla T	0.03212$^{\pm 0.00033}$	0.08913$^{\pm 0.00080}$	0.00142$^{\pm 0.00004}$	**0.55896**$^{\pm 0.00354}$	0.01715$^{\pm 0.00080}$	0.00212$^{\pm 0.00003}$	0.00258$^{\pm 0.00005}$	0.00192$^{\pm 0.00001}$
	RLT	0.03113$^{\pm 0.00018}$	0.08673$^{\pm 0.00046}$	**0.00137**$^{\pm 0.00005}$	0.56955$^{\pm 0.00435}$	**0.01499**$^{\pm 0.00026}$	**0.00210**$^{\pm 0.00005}$	**0.00251**$^{\pm 0.00003}$	0.00188$^{\pm 0.00001}$
	PDE FNet L	0.03184$^{\pm 0.00028}$	0.09044$^{\pm 0.00077}$	0.00201$^{\pm 0.00006}$	0.72189$^{\pm 0.01364}$	0.01549$^{\pm 0.00031}$	0.00322$^{\pm 0.00005}$	0.00344$^{\pm 0.00007}$	0.00185$^{\pm 0.00001}$
	TFormer	**0.02633**$^{\pm 0.00210}$	**0.06837**$^{\pm 0.00538}$	0.00356$^{\pm 0.00031}$	0.75804$^{\pm 0.03406}$	0.01658$^{\pm 0.00089}$	0.00372$^{\pm 0.00032}$	0.00349$^{\pm 0.00030}$	**0.00143**$^{\pm 0.00009}$
$\nu = 0.002$	cFNO	0.03564$^{\pm 0.00057}$	0.09972$^{\pm 0.00164}$	0.00201$^{\pm 0.00017}$	0.56319$^{\pm 0.00574}$	0.02200$^{\pm 0.00089}$	**0.00203**$^{\pm 0.00005}$	0.00312$^{\pm 0.00011}$	0.00195$^{\pm 0.00002}$
	cOFormer	0.03065$^{\pm 0.00344}$	0.09009$^{\pm 0.01133}$	0.00292$^{\pm 0.00064}$	0.62220$^{\pm 0.02883}$	0.01984$^{\pm 0.00218}$	0.00299$^{\pm 0.00057}$	0.00364$^{\pm 0.00048}$	0.00161$^{\pm 0.00012}$
	Vanilla T	0.03039$^{\pm 0.00034}$	0.08318$^{\pm 0.00081}$	0.00138$^{\pm 0.00004}$	**0.54259**$^{\pm 0.00396}$	0.01654$^{\pm 0.00079}$	0.00207$^{\pm 0.00003}$	0.00253$^{\pm 0.00005}$	0.00179$^{\pm 0.00001}$
	RLT	0.02938$^{\pm 0.00018}$	0.08067$^{\pm 0.00048}$	**0.00133**$^{\pm 0.00005}$	0.55276$^{\pm 0.00451}$	**0.01437**$^{\pm 0.00026}$	0.00206$^{\pm 0.00006}$	**0.00246**$^{\pm 0.00003}$	0.00175$^{\pm 0.00001}$
	PDE FNet L	0.03014$^{\pm 0.00029}$	0.08473$^{\pm 0.00079}$	0.00198$^{\pm 0.00006}$	0.70596$^{\pm 0.01435}$	0.01493$^{\pm 0.00030}$	0.00319$^{\pm 0.00005}$	0.00340$^{\pm 0.00007}$	0.00171$^{\pm 0.00001}$
	TFormer	**0.02469**$^{\pm 0.00213}$	**0.06250**$^{\pm 0.00545}$	0.00354$^{\pm 0.00031}$	0.73869$^{\pm 0.03457}$	0.01609$^{\pm 0.00088}$	0.00370$^{\pm 0.00032}$	0.00345$^{\pm 0.00030}$	**0.00130**$^{\pm 0.00009}$
$\nu = 0.004$	cFNO	0.03221$^{\pm 0.00059}$	0.08854$^{\pm 0.00173}$	0.00194$^{\pm 0.00016}$	0.53295$^{\pm 0.00448}$	0.02047$^{\pm 0.00089}$	**0.00192**$^{\pm 0.00005}$	0.00293$^{\pm 0.00010}$	0.00168$^{\pm 0.00002}$
	cOFormer	0.02717$^{\pm 0.00354}$	0.07903$^{\pm 0.01181}$	0.00280$^{\pm 0.00063}$	0.57538$^{\pm 0.03406}$	0.01830$^{\pm 0.00219}$	0.00285$^{\pm 0.00056}$	0.00344$^{\pm 0.00048}$	0.00135$^{\pm 0.00012}$
	Vanilla T	0.02692$^{\pm 0.00034}$	0.07206$^{\pm 0.00081}$	0.00132$^{\pm 0.00004}$	**0.50847**$^{\pm 0.00472}$	0.01528$^{\pm 0.00076}$	0.00201$^{\pm 0.00003}$	0.00243$^{\pm 0.00005}$	0.00153$^{\pm 0.00001}$
	RLT	0.02589$^{\pm 0.00019}$	0.06943$^{\pm 0.00050}$	**0.00127**$^{\pm 0.00005}$	0.51754$^{\pm 0.00424}$	**0.01313**$^{\pm 0.00024}$	0.00200$^{\pm 0.00006}$	**0.00237**$^{\pm 0.00004}$	0.00148$^{\pm 0.00001}$
	PDE FNet L	0.02685$^{\pm 0.00030}$	0.07452$^{\pm 0.00082}$	0.00194$^{\pm 0.00006}$	0.67111$^{\pm 0.01566}$	0.01384$^{\pm 0.00028}$	0.00314$^{\pm 0.00005}$	0.00330$^{\pm 0.00007}$	0.00145$^{\pm 0.00001}$
	TFormer	**0.02179**$^{\pm 0.00210}$	**0.05352**$^{\pm 0.00517}$	0.00350$^{\pm 0.00031}$	0.69552$^{\pm 0.03591}$	0.01515$^{\pm 0.00087}$	0.00366$^{\pm 0.00032}$	0.00336$^{\pm 0.00030}$	**0.00106**$^{\pm 0.00008}$
$\nu = 0.01$	cFNO	0.02323$^{\pm 0.00060}$	0.06075$^{\pm 0.00177}$	0.00181$^{\pm 0.00013}$	0.44556$^{\pm 0.00171}$	0.01655$^{\pm 0.00085}$	**0.00174**$^{\pm 0.00005}$	0.00244$^{\pm 0.00009}$	0.00108$^{\pm 0.00002}$
	cOFormer	0.01915$^{\pm 0.00327}$	0.05550$^{\pm 0.01119}$	0.00257$^{\pm 0.00061}$	0.44718$^{\pm 0.04216}$	0.01443$^{\pm 0.00207}$	0.00257$^{\pm 0.00054}$	0.00292$^{\pm 0.00045}$	0.00078$^{\pm 0.00010}$
	Vanilla T	0.01913$^{\pm 0.00029}$	0.04986$^{\pm 0.00066}$	0.00122$^{\pm 0.00003}$	**0.40820**$^{\pm 0.00469}$	0.01222$^{\pm 0.00063}$	0.00187$^{\pm 0.00003}$	0.00211$^{\pm 0.00004}$	0.00096$^{\pm 0.00001}$
	RLT	0.01821$^{\pm 0.00019}$	0.04752$^{\pm 0.00048}$	**0.00118**$^{\pm 0.00005}$	0.41442$^{\pm 0.00374}$	**0.01019**$^{\pm 0.00018}$	0.00187$^{\pm 0.00006}$	**0.00207**$^{\pm 0.00004}$	0.00092$^{\pm 0.00001}$
	PDE FNet L	0.02005$^{\pm 0.00027}$	0.05575$^{\pm 0.00068}$	0.00186$^{\pm 0.00006}$	0.56221$^{\pm 0.01743}$	0.01148$^{\pm 0.00022}$	0.00303$^{\pm 0.00005}$	0.00300$^{\pm 0.00007}$	0.00091$^{\pm 0.00001}$
	TFormer	**0.01679**$^{\pm 0.00169}$	**0.04100**$^{\pm 0.00382}$	0.00342$^{\pm 0.00031}$	0.56028$^{\pm 0.03838}$	0.01316$^{\pm 0.00080}$	0.00354$^{\pm 0.00031}$	0.00301$^{\pm 0.00029}$	**0.00063**$^{\pm 0.00006}$
$\nu = 0.02$	cFNO	0.01429$^{\pm 0.00046}$	0.03653$^{\pm 0.00120}$	0.00171$^{\pm 0.00010}$	0.33526$^{\pm 0.00073}$	0.01279$^{\pm 0.00077}$	**0.00165**$^{\pm 0.00006}$	0.00203$^{\pm 0.00006}$	0.00057$^{\pm 0.00002}$
	cOFormer	0.01284$^{\pm 0.00211}$	0.03922$^{\pm 0.00816}$	0.00239$^{\pm 0.00058}$	0.30647$^{\pm 0.03643}$	0.01074$^{\pm 0.00165}$	0.00233$^{\pm 0.00051}$	0.00232$^{\pm 0.00035}$	0.00036$^{\pm 0.00005}$
	Vanilla T	0.01254$^{\pm 0.00020}$	0.03345$^{\pm 0.00044}$	0.00110$^{\pm 0.00003}$	**0.28576**$^{\pm 0.00308}$	0.00922$^{\pm 0.00042}$	0.00169$^{\pm 0.00003}$	0.00168$^{\pm 0.00003}$	0.00053$^{\pm 0.00001}$
	RLT	**0.01195**$^{\pm 0.00015}$	**0.03210**$^{\pm 0.00043}$	**0.00107**$^{\pm 0.00005}$	0.28962$^{\pm 0.00400}$	**0.00752**$^{\pm 0.00014}$	0.00170$^{\pm 0.00005}$	**0.00167**$^{\pm 0.00003}$	0.00049$^{\pm 0.00000}$
	PDE FNet L	0.01471$^{\pm 0.00021}$	0.04280$^{\pm 0.00050}$	0.00176$^{\pm 0.00006}$	0.42181$^{\pm 0.01681}$	0.00945$^{\pm 0.00016}$	0.00288$^{\pm 0.00005}$	0.00255$^{\pm 0.00006}$	0.00050$^{\pm 0.00001}$
	TFormer	0.01292$^{\pm 0.00133}$	0.03251$^{\pm 0.00333}$	0.00329$^{\pm 0.00032}$	0.39547$^{\pm 0.03434}$	0.01129$^{\pm 0.00073}$	0.00334$^{\pm 0.00031}$	0.00249$^{\pm 0.00026}$	**0.00035**$^{\pm 0.00003}$
$\nu = 0.04$	cFNO	0.01023$^{\pm 0.00020}$	0.03151$^{\pm 0.00129}$	0.00173$^{\pm 0.00008}$	0.20806$^{\pm 0.00214}$	0.01058$^{\pm 0.00067}$	0.00169$^{\pm 0.00007}$	0.00204$^{\pm 0.00006}$	0.00023$^{\pm 0.00001}$
	cOFormer	0.00996$^{\pm 0.00124}$	0.03333$^{\pm 0.00630}$	0.00240$^{\pm 0.00061}$	0.19648$^{\pm 0.02383}$	0.00838$^{\pm 0.00118}$	0.00223$^{\pm 0.00050}$	0.00176$^{\pm 0.00024}$	0.00020$^{\pm 0.00001}$
	Vanilla T	0.00813$^{\pm 0.00011}$	0.02361$^{\pm 0.00028}$	0.00098$^{\pm 0.00003}$	**0.15865**$^{\pm 0.00286}$	0.00674$^{\pm 0.00016}$	**0.00146**$^{\pm 0.00003}$	**0.00126**$^{\pm 0.00002}$	0.00025$^{\pm 0.00000}$
	RLT	**0.00796**$^{\pm 0.00013}$	**0.02335**$^{\pm 0.00042}$	**0.00096**$^{\pm 0.00005}$	0.16432$^{\pm 0.00266}$	**0.00566**$^{\pm 0.00018}$	0.00150$^{\pm 0.00005}$	0.00127$^{\pm 0.00002}$	0.00023$^{\pm 0.00000}$
	PDE FNet L	0.01102$^{\pm 0.00011}$	0.03428$^{\pm 0.00027}$	0.00165$^{\pm 0.00006}$	0.26431$^{\pm 0.01305}$	0.00785$^{\pm 0.00011}$	0.00264$^{\pm 0.00005}$	0.00194$^{\pm 0.00004}$	0.00026$^{\pm 0.00000}$
	TFormer	0.01022$^{\pm 0.00090}$	0.02805$^{\pm 0.00262}$	0.00318$^{\pm 0.00033}$	0.24491$^{\pm 0.02359}$	0.00976$^{\pm 0.00059}$	0.00312$^{\pm 0.00030}$	0.00191$^{\pm 0.00017}$	**0.00018**$^{\pm 0.00001}$
$\nu = 0.1$	cFNO	0.00980$^{\pm 0.00040}$	0.03267$^{\pm 0.00273}$	0.00190$^{\pm 0.00019}$	0.11768$^{\pm 0.00378}$	0.00925$^{\pm 0.00071}$	0.00174$^{\pm 0.00010}$	0.00188$^{\pm 0.00010}$	0.00015$^{\pm 0.00001}$
	cOFormer	0.00818$^{\pm 0.00177}$	0.03394$^{\pm 0.00765}$	0.00282$^{\pm 0.00098}$	0.12205$^{\pm 0.02858}$	0.00716$^{\pm 0.00119}$	0.00239$^{\pm 0.00064}$	0.00122$^{\pm 0.00021}$	0.00011$^{\pm 0.00001}$
	Vanilla T	**0.00528**$^{\pm 0.00010}$	**0.01765**$^{\pm 0.00054}$	**0.00084**$^{\pm 0.00003}$	**0.06849**$^{\pm 0.00157}$	0.00451$^{\pm 0.00011}$	**0.00119**$^{\pm 0.00003}$	0.00080$^{\pm 0.00001}$	0.00013$^{\pm 0.00000}$
	RLT	0.00531$^{\pm 0.00012}$	0.01777$^{\pm 0.00055}$	0.00084$^{\pm 0.00005}$	0.07414$^{\pm 0.00111}$	**0.00416**$^{\pm 0.00016}$	0.00126$^{\pm 0.00004}$	**0.00079**$^{\pm 0.00001}$	0.00012$^{\pm 0.00000}$
	PDE FNet L	0.00781$^{\pm 0.00002}$	0.02841$^{\pm 0.00036}$	0.00151$^{\pm 0.00004}$	0.12336$^{\pm 0.00527}$	0.00606$^{\pm 0.00007}$	0.00224$^{\pm 0.00002}$	0.00113$^{\pm 0.00001}$	0.00014$^{\pm 0.00000}$
	TFormer	0.00981$^{\pm 0.00069}$	0.03279$^{\pm 0.00261}$	0.00327$^{\pm 0.00039}$	0.13294$^{\pm 0.00795}$	0.00921$^{\pm 0.00059}$	0.00319$^{\pm 0.00020}$	0.00154$^{\pm 0.00014}$	**0.00009**$^{\pm 0.00000}$
$\nu = 0.2$	cFNO	0.00638$^{\pm 0.00044}$	0.02449$^{\pm 0.00210}$	0.00196$^{\pm 0.00024}$	0.08119$^{\pm 0.00440}$	0.00710$^{\pm 0.00086}$	0.00152$^{\pm 0.00010}$	0.00115$^{\pm 0.00013}$	0.00009$^{\pm 0.00001}$
	cOFormer	0.00553$^{\pm 0.00121}$	0.02422$^{\pm 0.00650}$	0.00248$^{\pm 0.00077}$	0.06259$^{\pm 0.01842}$	0.00549$^{\pm 0.00105}$	0.00191$^{\pm 0.00047}$	0.00075$^{\pm 0.00012}$	**0.00004**$^{\pm 0.00001}$
	Vanilla T	**0.00320**$^{\pm 0.00006}$	**0.01205**$^{\pm 0.00023}$	**0.00070**$^{\pm 0.00004}$	**0.03883**$^{\pm 0.00076}$	0.00291$^{\pm 0.00012}$	**0.00087**$^{\pm 0.00003}$	0.00044$^{\pm 0.00001}$	0.00007$^{\pm 0.00000}$
	RLT	0.00325$^{\pm 0.00008}$	0.01264$^{\pm 0.00049}$	0.00072$^{\pm 0.00005}$	0.04084$^{\pm 0.00070}$	**0.00260**$^{\pm 0.00009}$	0.00093$^{\pm 0.00003}$	**0.00043**$^{\pm 0.00001}$	0.00007$^{\pm 0.00000}$
	PDE FNet L	0.00496$^{\pm 0.00004}$	0.02032$^{\pm 0.00038}$	0.00129$^{\pm 0.00003}$	0.06038$^{\pm 0.00123}$	0.00409$^{\pm 0.00004}$	0.00163$^{\pm 0.00002}$	0.00061$^{\pm 0.00001}$	0.00008$^{\pm 0.00000}$
	TFormer	0.00574$^{\pm 0.00040}$	0.02202$^{\pm 0.00285}$	0.00299$^{\pm 0.00040}$	0.06912$^{\pm 0.00213}$	0.00633$^{\pm 0.00054}$	0.00229$^{\pm 0.00016}$	0.00065$^{\pm 0.00002}$	0.00004$^{\pm 0.00000}$
$\nu = 0.4$	cFNO	0.00500$^{\pm 0.00037}$	0.02260$^{\pm 0.00137}$	0.00203$^{\pm 0.00014}$	0.06044$^{\pm 0.00541}$	0.00726$^{\pm 0.00074}$	0.00141$^{\pm 0.00010}$	0.00090$^{\pm 0.00013}$	0.00005$^{\pm 0.00001}$
	cOFormer	0.00457$^{\pm 0.00087}$	0.02241$^{\pm 0.00576}$	0.00259$^{\pm 0.00086}$	0.03858$^{\pm 0.00781}$	0.00472$^{\pm 0.00103}$	0.00170$^{\pm 0.00039}$	0.00051$^{\pm 0.00005}$	0.00003$^{\pm 0.00000}$
	Vanilla T	**0.00217**$^{\pm 0.00004}$	**0.00976**$^{\pm 0.00019}$	**0.00066**$^{\pm 0.00004}$	**0.02241**$^{\pm 0.00053}$	0.00200$^{\pm 0.00012}$	**0.00066**$^{\pm 0.00002}$	0.00027$^{\pm 0.00000}$	0.00005$^{\pm 0.00000}$
	RLT	0.00224$^{\pm 0.00007}$	0.01031$^{\pm 0.00028}$	0.00067$^{\pm 0.00005}$	0.02271$^{\pm 0.00071}$	**0.00179**$^{\pm 0.00008}$	0.00070$^{\pm 0.00003}$	**0.00026**$^{\pm 0.00001}$	0.00004$^{\pm 0.00000}$
	PDE FNet L	0.00348$^{\pm 0.00002}$	0.01769$^{\pm 0.00034}$	0.00116$^{\pm 0.00004}$	0.03198$^{\pm 0.00076}$	0.00301$^{\pm 0.00011}$	0.00121$^{\pm 0.00002}$	0.00036$^{\pm 0.00000}$	0.00005$^{\pm 0.00000}$
	TFormer	0.00440$^{\pm 0.00035}$	0.02091$^{\pm 0.00348}$	0.00291$^{\pm 0.00042}$	0.03756$^{\pm 0.00213}$	0.00505$^{\pm 0.00047}$	0.00178$^{\pm 0.00014}$	0.00037$^{\pm 0.00002}$	**0.00003**$^{\pm 0.00000}$
$\nu = 1.0$	cFNO	0.02253$^{\pm 0.00743}$	0.19532$^{\pm 0.09071}$	0.02457$^{\pm 0.00878}$	0.10598$^{\pm 0.01432}$	0.02387$^{\pm 0.00802}$	0.00734$^{\pm 0.00214}$	0.00070$^{\pm 0.00006}$	0.00004$^{\pm 0.00001}$
	cOFormer	0.02344$^{\pm 0.01450}$	0.15105$^{\pm 0.07657}$	0.02324$^{\pm 0.01932}$	0.08757$^{\pm 0.04888}$	0.02512$^{\pm 0.01677}$	0.00846$^{\pm 0.00522}$	0.00049$^{\pm 0.00006}$	**0.00003**$^{\pm 0.00000}$
	Vanilla T	0.00566$^{\pm 0.00065}$	0.03273$^{\pm 0.00398}$	0.00455$^{\pm 0.00112}$	0.02639$^{\pm 0.00250}$	0.00486$^{\pm 0.00090}$	0.00203$^{\pm 0.00028}$	0.00034$^{\pm 0.00002}$	0.00005$^{\pm 0.00000}$
	RLT	**0.00460**$^{\pm 0.00034}$	**0.02788**$^{\pm 0.00424}$	0.00292$^{\pm 0.00040}$	**0.02569**$^{\pm 0.00270}$	**0.00362**$^{\pm 0.00028}$	**0.00165**$^{\pm 0.00013}$	0.00033$^{\pm 0.00003}$	0.00005$^{\pm 0.00000}$
	PDE FNet L	0.00503$^{\pm 0.00011}$	0.03570$^{\pm 0.00266}$	**0.00246**$^{\pm 0.00015}$	0.03006$^{\pm 0.00168}$	0.00444$^{\pm 0.00034}$	0.00182$^{\pm 0.00005}$	**0.00032**$^{\pm 0.00002}$	0.00005$^{\pm 0.00000}$
	TFormer	0.00884$^{\pm 0.00082}$	0.06013$^{\pm 0.00766}$	0.00497$^{\pm 0.00087}$	0.04741$^{\pm 0.00474}$	0.00869$^{\pm 0.00076}$	0.00332$^{\pm 0.00025}$	0.00036$^{\pm 0.00002}$	0.00003$^{\pm 0.00000}$
$\nu = 2.0$	cFNO	0.00357$^{\pm 0.00017}$	0.02330$^{\pm 0.00273}$	0.00213$^{\pm 0.00014}$	0.03601$^{\pm 0.00176}$	0.00620$^{\pm 0.00125}$	0.00130$^{\pm 0.00006}$	0.00042$^{\pm 0.00004}$	0.00004$^{\pm 0.00001}$
	cOFormer	0.00345$^{\pm 0.00068}$	0.02246$^{\pm 0.00545}$	0.00241$^{\pm 0.00097}$	0.02491$^{\pm 0.00358}$	0.00404$^{\pm 0.00107}$	0.00131$^{\pm 0.00029}$	0.00032$^{\pm 0.00004}$	0.00003$^{\pm 0.00001}$
	Vanilla T	0.00178$^{\pm 0.00002}$	0.00916$^{\pm 0.00022}$	0.00067$^{\pm 0.00006}$	0.01528$^{\pm 0.00027}$	0.00188$^{\pm 0.00009}$	0.00064$^{\pm 0.00002}$	0.00014$^{\pm 0.00001}$	0.00003$^{\pm 0.00000}$
	RLT	**0.00172**$^{\pm 0.00003}$	**0.00859**$^{\pm 0.00022}$	**0.00053**$^{\pm 0.00004}$	**0.01420**$^{\pm 0.00031}$	**0.00153**$^{\pm 0.00001}$	**0.00059**$^{\pm 0.00002}$	**0.00014**$^{\pm 0.00000}$	0.00003$^{\pm 0.00000}$
	PDE FNet L	0.00218$^{\pm 0.00004}$	0.01452$^{\pm 0.00041}$	0.00089$^{\pm 0.00003}$	0.01699$^{\pm 0.00052}$	0.00197$^{\pm 0.00013}$	0.00080$^{\pm 0.00001}$	0.00017$^{\pm 0.00001}$	0.00003$^{\pm 0.00000}$
	TFormer	0.00372$^{\pm 0.00065}$	0.02989$^{\pm 0.01152}$	0.00318$^{\pm 0.00075}$	0.01989$^{\pm 0.00235}$	0.00419$^{\pm 0.00075}$	0.00146$^{\pm 0.00021}$	0.00018$^{\pm 0.00001}$	**0.00002**$^{\pm 0.00000}$
$\nu = 4.0$	cFNO	0.52389$^{\pm 0.27001}$	5.99772$^{\pm 3.68348}$	0.55047$^{\pm 0.25670}$	0.79522$^{\pm 0.21174}$	0.53006$^{\pm 0.27187}$	0.14081$^{\pm 0.06372}$	0.00149$^{\pm 0.00021}$	0.00007$^{\pm 0.00003}$
	cOFormer	0.15337$^{\pm 0.08312}$	1.42134$^{\pm 0.97772}$	0.17178$^{\pm 0.09685}$	0.40760$^{\pm 0.23914}$	0.14382$^{\pm 0.07982}$	0.04788$^{\pm 0.02507}$	**0.00085**$^{\pm 0.00018}$	**0.00005**$^{\pm 0.00002}$
	Vanilla T	0.09770$^{\pm 0.06544}$	0.28902$^{\pm 0.11971}$	0.13961$^{\pm 0.09734}$	0.51194$^{\pm 0.28732}$	0.09467$^{\pm 0.06963}$	0.03888$^{\pm 0.02460}$	0.00119$^{\pm 0.00040}$	0.00019$^{\pm 0.00006}$
	RLT	0.03169$^{\pm 0.00906}$	**0.15804**$^{\pm 0.05663}$	0.03823$^{\pm 0.01306}$	0.16386$^{\pm 0.06746}$	0.02801$^{\pm 0.00686}$	0.01186$^{\pm 0.00370}$	0.00088$^{\pm 0.00010}$	0.00015$^{\pm 0.00002}$
	PDE FNet L	**0.02700**$^{\pm 0.00838}$	0.27361$^{\pm 0.08362}$	**0.02340**$^{\pm 0.00711}$	**0.10651**$^{\pm 0.04327}$	**0.02561**$^{\pm 0.01202}$	**0.00989**$^{\pm 0.00323}$	0.00126$^{\pm 0.00047}$	0.00018$^{\pm 0.00004}$
	TFormer	0.48880$^{\pm 0.56291}$	5.62934$^{\pm 6.34668}$	0.46775$^{\pm 0.58939}$	0.95069$^{\pm 0.86531}$	0.47401$^{\pm 0.54969}$	0.14802$^{\pm 0.14868}$	0.00836$^{\pm 0.00403}$	0.00110$^{\pm 0.00052}$

Table C.8: Errors of the models for different evaluation metrics, including RMSE, normalized RMSE (nRMSE), RMSE from conserved value (cRMSE), maximum error, RMSE at the boundaries (bRMSE), and RMSE in Fourier space (fRMSE) on the 1D Burgers' equation. The unseen PDE parameters are highlighted in light grey.

PDE Parameter	Model	RMSE	nRMSE	cRMSE	max error	bRMSE	fRMSE low	fRMSE mid	fRMSE high
$\eta = \zeta = 0.1$	cFNO	**0.13815**$^{\pm 0.02740}$	**0.22822**$^{\pm 0.04567}$	0.03621$^{\pm 0.01177}$	**1.30708**$^{\pm 0.24899}$	**0.13538**$^{\pm 0.01964}$	**0.02174**$^{\pm 0.00406}$	**0.03206**$^{\pm 0.00684}$	0.00071$^{\pm 0.00004}$
	cOFormer	0.21770$^{\pm 0.06134}$	0.35954$^{\pm 0.10766}$	0.06333$^{\pm 0.04487}$	1.37601$^{\pm 0.44165}$	0.19703$^{\pm 0.07985}$	0.04605$^{\pm 0.01930}$	0.04311$^{\pm 0.01516}$	0.00083$^{\pm 0.00026}$
	Vanilla T	0.56658$^{\pm 0.00740}$	0.85041$^{\pm 0.01120}$	0.00338$^{\pm 0.00125}$	2.42848$^{\pm 0.02265}$	0.44764$^{\pm 0.00538}$	0.13758$^{\pm 0.00260}$	0.06191$^{\pm 0.00054}$	0.00055$^{\pm 0.00001}$
	RLT	0.56903$^{\pm 0.01346}$	0.85107$^{\pm 0.01963}$	**0.00189**$^{\pm 0.00028}$	2.46180$^{\pm 0.03914}$	0.44764$^{\pm 0.01095}$	0.13852$^{\pm 0.00448}$	0.06249$^{\pm 0.00096}$	**0.00054**$^{\pm 0.00000}$
	PDE FNet L	0.54161$^{\pm 0.01421}$	0.81292$^{\pm 0.02414}$	0.00447$^{\pm 0.00066}$	2.37581$^{\pm 0.02309}$	0.43243$^{\pm 0.01354}$	0.12912$^{\pm 0.00438}$	0.06268$^{\pm 0.00076}$	0.00070$^{\pm 0.00008}$
	TFormer	0.56060$^{\pm 0.00222}$	0.84121$^{\pm 0.00385}$	0.01181$^{\pm 0.00381}$	2.43544$^{\pm 0.02413}$	0.44122$^{\pm 0.00522}$	0.13765$^{\pm 0.00133}$	0.06226$^{\pm 0.00035}$	0.00071$^{\pm 0.00003}$
$\eta = \zeta = 0.2$	cFNO	0.01803$^{\pm 0.00288}$	0.03116$^{\pm 0.00449}$	0.00289$^{\pm 0.00112}$	0.44270$^{\pm 0.03068}$	0.02886$^{\pm 0.00151}$	0.00230$^{\pm 0.00068}$	0.00422$^{\pm 0.00099}$	0.00061$^{\pm 0.00000}$
	cOFormer	0.01464$^{\pm 0.00177}$	0.02415$^{\pm 0.00266}$	0.00202$^{\pm 0.00058}$	0.34068$^{\pm 0.02758}$	0.01497$^{\pm 0.00221}$	**0.00194**$^{\pm 0.00045}$	**0.00284**$^{\pm 0.00055}$	0.00048$^{\pm 0.00001}$
	Vanilla T	0.02131$^{\pm 0.00098}$	0.03773$^{\pm 0.00144}$	0.00190$^{\pm 0.00015}$	0.43832$^{\pm 0.00399}$	0.01623$^{\pm 0.00140}$	0.00352$^{\pm 0.00027}$	0.00517$^{\pm 0.00021}$	0.00054$^{\pm 0.00001}$
	RLT	**0.01353**$^{\pm 0.00116}$	**0.02388**$^{\pm 0.00168}$	**0.00141**$^{\pm 0.00018}$	**0.32888**$^{\pm 0.01200}$	**0.00928**$^{\pm 0.00090}$	0.00222$^{\pm 0.00030}$	0.00319$^{\pm 0.00043}$	**0.00047**$^{\pm 0.00001}$
	PDE FNet L	0.01710$^{\pm 0.00059}$	0.03061$^{\pm 0.00102}$	0.00203$^{\pm 0.00008}$	0.40255$^{\pm 0.01481}$	0.01587$^{\pm 0.00189}$	0.00282$^{\pm 0.00010}$	0.00424$^{\pm 0.00033}$	0.00059$^{\pm 0.00000}$
	TFormer	0.03306$^{\pm 0.00559}$	0.05410$^{\pm 0.00917}$	0.00727$^{\pm 0.00233}$	0.52504$^{\pm 0.07036}$	0.03205$^{\pm 0.00532}$	0.00734$^{\pm 0.00166}$	0.00766$^{\pm 0.00121}$	0.00056$^{\pm 0.00005}$
$\eta = \zeta = 0.4$	cFNO	0.01747$^{\pm 0.00283}$	0.03049$^{\pm 0.00450}$	0.00271$^{\pm 0.00095}$	0.44950$^{\pm 0.02850}$	0.02997$^{\pm 0.00322}$	0.00205$^{\pm 0.00056}$	0.00421$^{\pm 0.00104}$	0.00061$^{\pm 0.00001}$
	cOFormer	**0.01266**$^{\pm 0.00093}$	**0.02125**$^{\pm 0.00124}$	0.00155$^{\pm 0.00037}$	**0.34690**$^{\pm 0.01751}$	0.01157$^{\pm 0.00130}$	**0.00151**$^{\pm 0.00028}$	**0.00237**$^{\pm 0.00028}$	**0.00047**$^{\pm 0.00001}$
	Vanilla T	0.01871$^{\pm 0.00049}$	0.03437$^{\pm 0.00084}$	0.00182$^{\pm 0.00015}$	0.44210$^{\pm 0.00196}$	0.01553$^{\pm 0.00109}$	0.00300$^{\pm 0.00015}$	0.00479$^{\pm 0.00017}$	0.00054$^{\pm 0.00001}$
	RLT	0.01306$^{\pm 0.00067}$	0.02338$^{\pm 0.00101}$	**0.00137**$^{\pm 0.00019}$	0.34803$^{\pm 0.00908}$	**0.00961**$^{\pm 0.00056}$	0.00202$^{\pm 0.00017}$	0.00323$^{\pm 0.00039}$	0.00048$^{\pm 0.00000}$
	PDE FNet L	0.01742$^{\pm 0.00066}$	0.03129$^{\pm 0.00114}$	0.00201$^{\pm 0.00008}$	0.42661$^{\pm 0.01047}$	0.01638$^{\pm 0.00143}$	0.00280$^{\pm 0.00013}$	0.00443$^{\pm 0.00034}$	0.00058$^{\pm 0.00000}$
	TFormer	0.03379$^{\pm 0.00515}$	0.05575$^{\pm 0.00839}$	0.00759$^{\pm 0.00218}$	0.53684$^{\pm 0.06084}$	0.03533$^{\pm 0.00514}$	0.00765$^{\pm 0.00174}$	0.00791$^{\pm 0.00083}$	0.00057$^{\pm 0.00005}$
$\eta = \zeta = 0.7$	cFNO	0.01982$^{\pm 0.00371}$	0.03427$^{\pm 0.00596}$	0.00332$^{\pm 0.00092}$	0.49379$^{\pm 0.03073}$	0.03148$^{\pm 0.00418}$	0.00246$^{\pm 0.00062}$	0.00505$^{\pm 0.00132}$	0.00061$^{\pm 0.00001}$
	cOFormer	**0.01172**$^{\pm 0.00101}$	**0.01975**$^{\pm 0.00161}$	**0.00140**$^{\pm 0.00039}$	**0.33953**$^{\pm 0.02982}$	0.01332$^{\pm 0.00275}$	**0.00142**$^{\pm 0.00027}$	**0.00209**$^{\pm 0.00034}$	**0.00047**$^{\pm 0.00002}$
	Vanilla T	0.01890$^{\pm 0.00043}$	0.03475$^{\pm 0.00080}$	0.00185$^{\pm 0.00014}$	0.46037$^{\pm 0.00140}$	0.01592$^{\pm 0.00089}$	0.00302$^{\pm 0.00013}$	0.00489$^{\pm 0.00013}$	0.00055$^{\pm 0.00001}$
	RLT	0.01329$^{\pm 0.00055}$	0.02382$^{\pm 0.00092}$	0.00146$^{\pm 0.00020}$	0.36764$^{\pm 0.00798}$	**0.01004**$^{\pm 0.00046}$	0.00205$^{\pm 0.00015}$	0.00335$^{\pm 0.00037}$	0.00049$^{\pm 0.00001}$
	PDE FNet L	0.01803$^{\pm 0.00069}$	0.03239$^{\pm 0.00123}$	0.00199$^{\pm 0.00009}$	0.45180$^{\pm 0.00802}$	0.01692$^{\pm 0.00128}$	0.00294$^{\pm 0.00016}$	0.00468$^{\pm 0.00031}$	0.00058$^{\pm 0.00000}$
	TFormer	0.03752$^{\pm 0.00660}$	0.06206$^{\pm 0.01084}$	0.00819$^{\pm 0.00162}$	0.57750$^{\pm 0.05240}$	0.03779$^{\pm 0.00641}$	0.00861$^{\pm 0.00175}$	0.00891$^{\pm 0.00107}$	0.00060$^{\pm 0.00006}$
$\eta = \zeta = 1.0$	cFNO	**0.45158**$^{\pm 0.12302}$	**0.71939**$^{\pm 0.18974}$	0.12723$^{\pm 0.03803}$	**1.72558**$^{\pm 0.34632}$	**0.39199**$^{\pm 0.10348}$	**0.11393**$^{\pm 0.03166}$	**0.05206**$^{\pm 0.01013}$	0.00064$^{\pm 0.00004}$
	cOFormer	1.68027$^{\pm 0.70413}$	3.02688$^{\pm 1.32607}$	0.54854$^{\pm 0.63949}$	5.14314$^{\pm 1.69888}$	1.43922$^{\pm 0.61395}$	0.28045$^{\pm 0.21747}$	0.06680$^{\pm 0.03149}$	0.01042$^{\pm 0.00498}$
	Vanilla T	0.60066$^{\pm 0.00264}$	0.89767$^{\pm 0.00422}$	0.00433$^{\pm 0.00154}$	2.61121$^{\pm 0.01220}$	0.51701$^{\pm 0.00240}$	0.16112$^{\pm 0.00135}$	0.05540$^{\pm 0.00061}$	0.00055$^{\pm 0.00001}$
	RLT	0.62245$^{\pm 0.00403}$	0.92993$^{\pm 0.00575}$	**0.00270**$^{\pm 0.00050}$	2.69136$^{\pm 0.01116}$	0.53317$^{\pm 0.00310}$	0.16622$^{\pm 0.00142}$	0.05832$^{\pm 0.00032}$	**0.00055**$^{\pm 0.00000}$
	PDE FNet L	0.53243$^{\pm 0.04797}$	0.79934$^{\pm 0.07401}$	0.01731$^{\pm 0.00897}$	2.42449$^{\pm 0.08221}$	0.45746$^{\pm 0.04361}$	0.14525$^{\pm 0.01285}$	0.05765$^{\pm 0.00099}$	0.00072$^{\pm 0.00005}$
	TFormer	0.58862$^{\pm 0.01134}$	0.87474$^{\pm 0.01611}$	0.03808$^{\pm 0.01015}$	2.64855$^{\pm 0.03581}$	0.51016$^{\pm 0.00916}$	0.16795$^{\pm 0.00234}$	0.05831$^{\pm 0.00100}$	0.00118$^{\pm 0.00020}$
$\eta = \zeta = 2.0$	cFNO	0.02787$^{\pm 0.00711}$	0.04855$^{\pm 0.01321}$	0.00446$^{\pm 0.00130}$	0.55316$^{\pm 0.06572}$	0.03794$^{\pm 0.00696}$	0.00349$^{\pm 0.00080}$	0.00747$^{\pm 0.00253}$	0.00063$^{\pm 0.00001}$
	cOFormer	**0.01167**$^{\pm 0.00097}$	**0.01962**$^{\pm 0.00146}$	0.00164$^{\pm 0.00038}$	0.35326$^{\pm 0.01655}$	0.01219$^{\pm 0.00107}$	**0.00153**$^{\pm 0.00027}$	**0.00192**$^{\pm 0.00035}$	**0.00048**$^{\pm 0.00001}$
	Vanilla T	0.01851$^{\pm 0.00047}$	0.03441$^{\pm 0.00105}$	0.00198$^{\pm 0.00011}$	0.46080$^{\pm 0.00378}$	0.01605$^{\pm 0.00100}$	0.00312$^{\pm 0.00013}$	0.00461$^{\pm 0.00023}$	0.00055$^{\pm 0.00000}$
	RLT	0.01312$^{\pm 0.00028}$	0.02342$^{\pm 0.00052}$	**0.00151**$^{\pm 0.00016}$	**0.34249**$^{\pm 0.01009}$	**0.01029**$^{\pm 0.00031}$	0.00220$^{\pm 0.00015}$	0.00309$^{\pm 0.00038}$	0.00048$^{\pm 0.00001}$
	PDE FNet L	0.01903$^{\pm 0.00075}$	0.03387$^{\pm 0.00133}$	0.00186$^{\pm 0.00013}$	0.44730$^{\pm 0.00649}$	0.01748$^{\pm 0.00101}$	0.00356$^{\pm 0.00022}$	0.00484$^{\pm 0.00032}$	0.00058$^{\pm 0.00002}$
	TFormer	0.04998$^{\pm 0.00841}$	0.08228$^{\pm 0.01400}$	0.01144$^{\pm 0.00192}$	0.66380$^{\pm 0.04607}$	0.05091$^{\pm 0.00835}$	0.01161$^{\pm 0.00201}$	0.01150$^{\pm 0.00157}$	0.00071$^{\pm 0.00005}$
$\eta = \zeta = 4.0$	cFNO	0.02437$^{\pm 0.00495}$	0.03955$^{\pm 0.00796}$	0.00524$^{\pm 0.00207}$	0.44606$^{\pm 0.04474}$	0.03653$^{\pm 0.00517}$	0.00433$^{\pm 0.00117}$	0.00499$^{\pm 0.00150}$	0.00065$^{\pm 0.00000}$
	cOFormer	**0.01174**$^{\pm 0.00122}$	**0.01931**$^{\pm 0.00193}$	0.00150$^{\pm 0.00044}$	0.31111$^{\pm 0.03837}$	0.01168$^{\pm 0.00127}$	**0.00152**$^{\pm 0.00035}$	**0.00183**$^{\pm 0.00036}$	**0.00047**$^{\pm 0.00002}$
	Vanilla T	0.01814$^{\pm 0.00044}$	0.03347$^{\pm 0.00100}$	0.00183$^{\pm 0.00007}$	0.43623$^{\pm 0.00322}$	0.01583$^{\pm 0.00101}$	0.00300$^{\pm 0.00017}$	0.00423$^{\pm 0.00018}$	0.00055$^{\pm 0.00001}$
	RLT	0.01297$^{\pm 0.00040}$	0.02275$^{\pm 0.00054}$	**0.00139**$^{\pm 0.00019}$	**0.30364**$^{\pm 0.01408}$	**0.01023**$^{\pm 0.00035}$	0.00212$^{\pm 0.00018}$	0.00285$^{\pm 0.00041}$	0.00047$^{\pm 0.00002}$
	PDE FNet L	0.01798$^{\pm 0.00073}$	0.03188$^{\pm 0.00124}$	0.00169$^{\pm 0.00012}$	0.41074$^{\pm 0.01737}$	0.01508$^{\pm 0.00061}$	0.00314$^{\pm 0.00018}$	0.00447$^{\pm 0.00030}$	0.00054$^{\pm 0.00001}$
	TFormer	0.05964$^{\pm 0.01244}$	0.09731$^{\pm 0.02081}$	0.01359$^{\pm 0.00271}$	0.66971$^{\pm 0.06689}$	0.06221$^{\pm 0.01135}$	0.01452$^{\pm 0.00291}$	0.01250$^{\pm 0.00258}$	0.00086$^{\pm 0.00007}$
$\eta = \zeta = 7.0$	cFNO	4.02187$^{\pm 1.15225}$	7.28715$^{\pm 2.23417}$	3.38601$^{\pm 1.69630}$	7.82273$^{\pm 1.44253}$	3.71412$^{\pm 1.16096}$	1.12352$^{\pm 0.31925}$	0.20572$^{\pm 0.13465}$	0.00166$^{\pm 0.00088}$
	cOFormer	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A [(1)]
	Vanilla T	0.61439$^{\pm 0.03931}$	0.90613$^{\pm 0.05064}$	0.36827$^{\pm 0.08806}$	2.37940$^{\pm 0.15731}$	0.52418$^{\pm 0.02684}$	0.22668$^{\pm 0.01948}$	**0.05061**$^{\pm 0.00392}$	**0.00060**$^{\pm 0.00002}$
	RLT	**0.52842**$^{\pm 0.01352}$	**0.81613**$^{\pm 0.03495}$	0.10137$^{\pm 0.02383}$	**2.18063**$^{\pm 0.12265}$	**0.43363**$^{\pm 0.00856}$	**0.15981**$^{\pm 0.00536}$	0.06324$^{\pm 0.00681}$	0.00100$^{\pm 0.00020}$
	PDE FNet L	0.69885$^{\pm 0.02577}$	1.10154$^{\pm 0.05179}$	**0.02647**$^{\pm 0.00988}$	2.83635$^{\pm 0.07785}$	0.59522$^{\pm 0.03081}$	0.19080$^{\pm 0.00592}$	0.06763$^{\pm 0.00853}$	0.00090$^{\pm 0.00008}$
	TFormer	0.59857$^{\pm 0.03037}$	0.92697$^{\pm 0.06412}$	0.16914$^{\pm 0.06409}$	2.51047$^{\pm 0.09527}$	0.52201$^{\pm 0.01881}$	0.19954$^{\pm 0.01464}$	0.05258$^{\pm 0.00141}$	0.00149$^{\pm 0.00020}$

Table C.9: Errors of the baseline and proposed models for different evaluation metrics, including RMSE, normalized RMSE (nRMSE), RMSE from conserved value (cRMSE), maximum error, RMSE at the boundaries (bRMSE), RMSE in Fourier space at low (fRMSE low), medium (fRMSE mid), and high frequency (fRMSE high) ranges on the 1D Advection equation. The unseen PDE parameters are highlighted in light grey. (1) The values for cOFormer and $\beta = 7.0$ are unavailable since the model produced an invalid error (NaN).

PDE Parameter	Model	RMSE	nRMSE	cRMSE	max error	bRMSE	fRMSE low	fRMSE mid	fRMSE high
$\eta = \zeta = 1.e-8$	cFNO	$2.58155^{\pm 0.13566}$	$0.44192^{\pm 0.03380}$	$0.40916^{\pm 0.10446}$	$21.69426^{\pm 0.18070}$	$2.04999^{\pm 0.13452}$	$0.77685^{\pm 0.05857}$	$0.37228^{\pm 0.00582}$	$0.03598^{\pm 0.00037}$
	cOFormer	$3.02712^{\pm 0.38680}$	$0.49915^{\pm 0.07259}$	$0.88899^{\pm 0.30046}$	$22.99324^{\pm 1.04214}$	$2.49086^{\pm 0.34437}$	$1.01242^{\pm 0.15942}$	$0.38426^{\pm 0.00095}$	$0.03538^{\pm 0.00008}$
	Vanilla T	$1.51246^{\pm 0.01903}$	$0.35558^{\pm 0.00669}$	$0.15108^{\pm 0.00119}$	$16.83115^{\pm 0.09133}$	$1.10144^{\pm 0.01817}$	$0.26918^{\pm 0.00293}$	$0.31178^{\pm 0.00415}$	$0.04001^{\pm 0.00028}$
	RLT	$\mathbf{1.39417}^{\pm 0.03924}$	$0.35618^{\pm 0.00845}$	$\mathbf{0.12506}^{\pm 0.00506}$	$\mathbf{16.37702}^{\pm 0.24448}$	$\mathbf{0.99421}^{\pm 0.02946}$	$\mathbf{0.24947}^{\pm 0.00997}$	$\mathbf{0.28381}^{\pm 0.00939}$	$0.03795^{\pm 0.00012}$
	PDE FNet L	$1.61054^{\pm 0.09977}$	$0.37169^{\pm 0.02259}$	$0.17184^{\pm 0.02377}$	$17.64064^{\pm 0.47819}$	$1.17303^{\pm 0.06655}$	$0.33987^{\pm 0.03740}$	$0.31868^{\pm 0.01322}$	$0.03795^{\pm 0.00082}$
	TFormer	$1.60045^{\pm 0.05223}$	$\mathbf{0.27225}^{\pm 0.01539}$	$0.34304^{\pm 0.03605}$	$17.85673^{\pm 0.24546}$	$1.29771^{\pm 0.05541}$	$0.41611^{\pm 0.01821}$	$0.29208^{\pm 0.00935}$	$\mathbf{0.03509}^{\pm 0.00016}$
$\eta = \zeta = 0.001$	cFNO	$2.51804^{\pm 0.14816}$	$0.44073^{\pm 0.03757}$	$0.41761^{\pm 0.11721}$	$21.81414^{\pm 0.36824}$	$1.98166^{\pm 0.14606}$	$0.75706^{\pm 0.06530}$	$0.36384^{\pm 0.00652}$	$0.03498^{\pm 0.00036}$
	cOFormer	$2.98272^{\pm 0.38360}$	$0.49961^{\pm 0.07356}$	$0.92098^{\pm 0.29383}$	$23.66670^{\pm 1.04164}$	$2.44187^{\pm 0.34301}$	$1.00637^{\pm 0.15565}$	$0.37713^{\pm 0.00093}$	$0.03439^{\pm 0.00008}$
	Vanilla T	$1.47247^{\pm 0.01996}$	$0.35629^{\pm 0.00657}$	$0.14759^{\pm 0.00167}$	$17.35032^{\pm 0.19198}$	$1.05734^{\pm 0.01850}$	$0.25801^{\pm 0.00328}$	$0.30452^{\pm 0.00413}$	$0.03885^{\pm 0.00026}$
	RLT	$\mathbf{1.35471}^{\pm 0.04058}$	$0.35773^{\pm 0.00864}$	$\mathbf{0.12322}^{\pm 0.00540}$	$\mathbf{16.91763}^{\pm 0.17560}$	$\mathbf{0.95569}^{\pm 0.03154}$	$\mathbf{0.24013}^{\pm 0.01119}$	$\mathbf{0.27592}^{\pm 0.00975}$	$0.03690^{\pm 0.00012}$
	PDE FNet L	$1.56243^{\pm 0.09521}$	$0.37346^{\pm 0.02358}$	$0.17271^{\pm 0.02221}$	$18.11287^{\pm 0.38416}$	$1.12648^{\pm 0.06222}$	$0.32711^{\pm 0.03350}$	$0.31033^{\pm 0.01312}$	$0.03690^{\pm 0.00080}$
	TFormer	$1.56350^{\pm 0.06419}$	$\mathbf{0.27466}^{\pm 0.01662}$	$0.33230^{\pm 0.03448}$	$17.98571^{\pm 0.58577}$	$1.26045^{\pm 0.05437}$	$0.40647^{\pm 0.02194}$	$0.28436^{\pm 0.01140}$	$\mathbf{0.03411}^{\pm 0.00021}$
$\eta = \zeta = 0.004$	cFNO	$2.35751^{\pm 0.15259}$	$0.43956^{\pm 0.04125}$	$0.41408^{\pm 0.11754}$	$20.26792^{\pm 0.36480}$	$1.86015^{\pm 0.14880}$	$0.74267^{\pm 0.06493}$	$0.33243^{\pm 0.00648}$	$0.01893^{\pm 0.00044}$
	cOFormer	$2.83530^{\pm 0.38996}$	$0.50092^{\pm 0.07599}$	$0.92075^{\pm 0.29394}$	$22.18693^{\pm 1.11060}$	$2.32945^{\pm 0.34806}$	$0.99183^{\pm 0.15551}$	$0.34550^{\pm 0.00101}$	$0.01837^{\pm 0.00015}$
	Vanilla T	$1.26284^{\pm 0.02046}$	$0.34836^{\pm 0.00719}$	$0.14525^{\pm 0.00173}$	$15.12255^{\pm 0.18402}$	$0.90945^{\pm 0.01807}$	$0.24611^{\pm 0.00311}$	$0.27340^{\pm 0.00408}$	$0.02352^{\pm 0.00030}$
	RLT	$\mathbf{1.14453}^{\pm 0.04210}$	$0.35055^{\pm 0.00906}$	$\mathbf{0.12017}^{\pm 0.00553}$	$\mathbf{14.59898}^{\pm 0.20032}$	$\mathbf{0.80990}^{\pm 0.03134}$	$\mathbf{0.22864}^{\pm 0.01101}$	$\mathbf{0.24609}^{\pm 0.00943}$	$0.02131^{\pm 0.00013}$
	PDE FNet L	$1.36383^{\pm 0.09836}$	$0.36813^{\pm 0.02487}$	$0.17042^{\pm 0.02213}$	$15.98553^{\pm 0.43091}$	$0.98626^{\pm 0.06300}$	$0.31555^{\pm 0.03299}$	$0.28001^{\pm 0.01275}$	$0.02144^{\pm 0.00101}$
	TFormer	$1.37717^{\pm 0.06344}$	$\mathbf{0.26320}^{\pm 0.01769}$	$0.32395^{\pm 0.03429}$	$15.70433^{\pm 0.65684}$	$1.12718^{\pm 0.05258}$	$0.39170^{\pm 0.02131}$	$0.25475^{\pm 0.01075}$	$\mathbf{0.01830}^{\pm 0.00021}$
$\eta = \zeta = 0.007$	cFNO	$2.22366^{\pm 0.15543}$	$0.44008^{\pm 0.04482}$	$0.41139^{\pm 0.11783}$	$19.06415^{\pm 0.35463}$	$1.76068^{\pm 0.15045}$	$0.72686^{\pm 0.06443}$	$0.29806^{\pm 0.00640}$	$0.01268^{\pm 0.00048}$
	cOFormer	$2.71108^{\pm 0.39483}$	$0.50343^{\pm 0.07832}$	$0.92066^{\pm 0.29409}$	$21.04013^{\pm 1.15553}$	$2.23511^{\pm 0.35207}$	$0.97543^{\pm 0.15524}$	$0.31076^{\pm 0.00111}$	$\mathbf{0.01215}^{\pm 0.00019}$
	Vanilla T	$1.10451^{\pm 0.01999}$	$0.34216^{\pm 0.00789}$	$0.14310^{\pm 0.00178}$	$13.43059^{\pm 0.17927}$	$0.79643^{\pm 0.01703}$	$0.23596^{\pm 0.00300}$	$0.24116^{\pm 0.00391}$	$0.01733^{\pm 0.00031}$
	RLT	$\mathbf{0.99061}^{\pm 0.04106}$	$0.34387^{\pm 0.00942}$	$\mathbf{0.11725}^{\pm 0.00565}$	$\mathbf{12.88513}^{\pm 0.22066}$	$\mathbf{0.70170}^{\pm 0.02987}$	$\mathbf{0.21843}^{\pm 0.01077}$	$\mathbf{0.21577}^{\pm 0.00868}$	$0.01505^{\pm 0.00013}$
	PDE FNet L	$1.21495^{\pm 0.09829}$	$0.36449^{\pm 0.02608}$	$0.16835^{\pm 0.02216}$	$14.39030^{\pm 0.43278}$	$0.88096^{\pm 0.06267}$	$0.30483^{\pm 0.03237}$	$0.24844^{\pm 0.01198}$	$0.01527^{\pm 0.00110}$
	TFormer	$1.24004^{\pm 0.06029}$	$\mathbf{0.25506}^{\pm 0.01810}$	$0.31594^{\pm 0.03415}$	$14.06615^{\pm 0.66414}$	$1.02836^{\pm 0.04931}$	$0.37738^{\pm 0.02064}$	$0.22396^{\pm 0.00990}$	$0.01227^{\pm 0.00020}$
$\eta = \zeta = 0.01$	cFNO	$2.16902^{\pm 0.14411}$	$0.43798^{\pm 0.04119}$	$0.40302^{\pm 0.10573}$	$18.35204^{\pm 0.21586}$	$1.73917^{\pm 0.13839}$	$0.72946^{\pm 0.05757}$	$0.27273^{\pm 0.00559}$	$0.00977^{\pm 0.00053}$
	cOFormer	$2.64474^{\pm 0.40166}$	$0.50261^{\pm 0.07848}$	$0.88830^{\pm 0.30135}$	$19.75586^{\pm 1.15662}$	$2.19530^{\pm 0.35649}$	$0.96347^{\pm 0.15838}$	$0.28370^{\pm 0.00125}$	$\mathbf{0.00922}^{\pm 0.00022}$
	Vanilla T	$1.01409^{\pm 0.01944}$	$0.33234^{\pm 0.00864}$	$0.14210^{\pm 0.00139}$	$11.82409^{\pm 0.16544}$	$0.74767^{\pm 0.01513}$	$0.23354^{\pm 0.00341}$	$0.21780^{\pm 0.00377}$	$0.01432^{\pm 0.00032}$
	RLT	$\mathbf{0.90573}^{\pm 0.03804}$	$0.33160^{\pm 0.00969}$	$\mathbf{0.11523}^{\pm 0.00518}$	$\mathbf{11.16491}^{\pm 0.33908}$	$\mathbf{0.65217}^{\pm 0.02705}$	$\mathbf{0.21464}^{\pm 0.00967}$	$\mathbf{0.19475}^{\pm 0.00770}$	$0.01196^{\pm 0.00013}$
	PDE FNet L	$1.13759^{\pm 0.10164}$	$0.35488^{\pm 0.02654}$	$0.16508^{\pm 0.02361}$	$12.88058^{\pm 0.63310}$	$0.83687^{\pm 0.06561}$	$0.30326^{\pm 0.03562}$	$0.22575^{\pm 0.01145}$	$0.01227^{\pm 0.00115}$
	TFormer	$1.17383^{\pm 0.05417}$	$\mathbf{0.24642}^{\pm 0.01735}$	$0.31710^{\pm 0.03593}$	$12.87278^{\pm 0.48319}$	$0.98981^{\pm 0.04920}$	$0.37273^{\pm 0.01771}$	$0.20406^{\pm 0.00896}$	$0.00956^{\pm 0.00019}$
$\eta = \zeta = 0.04$	cFNO	$1.55144^{\pm 0.13750}$	$0.47518^{\pm 0.06908}$	$0.40488^{\pm 0.11875}$	$13.43986^{\pm 0.23647}$	$1.26527^{\pm 0.13056}$	$0.57708^{\pm 0.05553}$	$0.13460^{\pm 0.00411}$	$0.00578^{\pm 0.00059}$
	cOFormer	$2.02555^{\pm 0.39876}$	$0.54523^{\pm 0.09438}$	$0.92286^{\pm 0.29630}$	$15.37358^{\pm 1.34566}$	$1.69482^{\pm 0.35794}$	$0.80480^{\pm 0.14644}$	$0.13796^{\pm 0.00237}$	$\mathbf{0.00492}^{\pm 0.00028}$
	Vanilla T	$0.60103^{\pm 0.01274}$	$0.32090^{\pm 0.01509}$	$0.13085^{\pm 0.00226}$	$6.86545^{\pm 0.11613}$	$0.42921^{\pm 0.00883}$	$0.17242^{\pm 0.00343}$	$0.10693^{\pm 0.00150}$	$0.00773^{\pm 0.00038}$
	RLT	$\mathbf{0.53251}^{\pm 0.02144}$	$0.30997^{\pm 0.00942}$	$\mathbf{0.10231}^{\pm 0.00614}$	$\mathbf{6.27388}^{\pm 0.17334}$	$\mathbf{0.37033}^{\pm 0.01529}$	$\mathbf{0.15135}^{\pm 0.00834}$	$\mathbf{0.09593}^{\pm 0.00255}$	$0.00616^{\pm 0.00017}$
	PDE FNet L	$0.71688^{\pm 0.07307}$	$0.36654^{\pm 0.03103}$	$0.15576^{\pm 0.02190}$	$7.72942^{\pm 0.39241}$	$0.52043^{\pm 0.04700}$	$0.22505^{\pm 0.02664}$	$0.11427^{\pm 0.00545}$	$0.00679^{\pm 0.00097}$
	TFormer	$0.75831^{\pm 0.03565}$	$\mathbf{0.23247}^{\pm 0.01387}$	$0.26741^{\pm 0.03051}$	$7.79036^{\pm 0.45149}$	$0.65429^{\pm 0.02462}$	$0.27417^{\pm 0.01545}$	$0.09845^{\pm 0.00350}$	$0.00501^{\pm 0.00014}$
$\eta = \zeta = 0.07$	cFNO	$1.39346^{\pm 0.10913}$	$0.51188^{\pm 0.07929}$	$0.40968^{\pm 0.11839}$	$11.56728^{\pm 0.23506}$	$1.15472^{\pm 0.10519}$	$0.51692^{\pm 0.04843}$	$0.11495^{\pm 0.00373}$	$0.00566^{\pm 0.00060}$
	cOFormer	$1.80251^{\pm 0.36396}$	$0.58325^{\pm 0.09955}$	$0.92577^{\pm 0.29767}$	$13.33813^{\pm 1.29769}$	$1.50776^{\pm 0.33282}$	$0.72062^{\pm 0.13186}$	$0.10975^{\pm 0.00333}$	$0.00481^{\pm 0.00028}$
	Vanilla T	$0.52379^{\pm 0.01173}$	$0.32858^{\pm 0.01958}$	$0.12904^{\pm 0.00273}$	$5.50682^{\pm 0.09296}$	$0.37487^{\pm 0.00738}$	$0.14979^{\pm 0.00338}$	$0.08882^{\pm 0.00122}$	$0.00700^{\pm 0.00042}$
	RLT	$\mathbf{0.46030}^{\pm 0.01811}$	$0.30654^{\pm 0.00778}$	$\mathbf{0.10081}^{\pm 0.00606}$	$\mathbf{5.04279}^{\pm 0.12646}$	$\mathbf{0.31894}^{\pm 0.01321}$	$\mathbf{0.12814}^{\pm 0.00719}$	$0.08069^{\pm 0.00175}$	$0.00568^{\pm 0.00019}$
	PDE FNet L	$0.62423^{\pm 0.06214}$	$0.38370^{\pm 0.03136}$	$0.15249^{\pm 0.02117}$	$6.21621^{\pm 0.34844}$	$0.45092^{\pm 0.03806}$	$0.19345^{\pm 0.02364}$	$0.09631^{\pm 0.00409}$	$0.00635^{\pm 0.00085}$
	TFormer	$0.64916^{\pm 0.03200}$	$\mathbf{0.23221}^{\pm 0.01060}$	$0.25328^{\pm 0.02768}$	$6.35505^{\pm 0.34889}$	$0.56360^{\pm 0.02262}$	$0.23236^{\pm 0.01383}$	$\mathbf{0.08024}^{\pm 0.00227}$	$\mathbf{0.00449}^{\pm 0.00010}$
$\eta = \zeta = 1.0$	cFNO	$1.37875^{\pm 0.09177}$	$0.53848^{\pm 0.07615}$	$0.41409^{\pm 0.10356}$	$10.88430^{\pm 0.30351}$	$1.15472^{\pm 0.09180}$	$0.50854^{\pm 0.04293}$	$0.11181^{\pm 0.00398}$	$0.00561^{\pm 0.00061}$
	cOFormer	$1.72455^{\pm 0.31995}$	$0.62663^{\pm 0.09846}$	$0.89359^{\pm 0.30599}$	$11.93673^{\pm 1.06532}$	$1.44018^{\pm 0.29960}$	$0.68130^{\pm 0.11717}$	$0.10244^{\pm 0.00399}$	$0.00483^{\pm 0.00028}$
	Vanilla T	$0.50313^{\pm 0.01081}$	$0.34896^{\pm 0.02736}$	$0.13415^{\pm 0.00286}$	$5.17994^{\pm 0.05993}$	$0.36573^{\pm 0.00475}$	$0.14250^{\pm 0.00360}$	$0.08466^{\pm 0.00128}$	$0.00680^{\pm 0.00046}$
	RLT	$\mathbf{0.43985}^{\pm 0.01576}$	$0.31424^{\pm 0.00776}$	$\mathbf{0.10289}^{\pm 0.00517}$	$\mathbf{4.86254}^{\pm 0.12619}$	$\mathbf{0.30727}^{\pm 0.01140}$	$\mathbf{0.12079}^{\pm 0.00607}$	$0.07764^{\pm 0.00139}$	$0.00552^{\pm 0.00020}$
	PDE FNet L	$0.59589^{\pm 0.05937}$	$0.40086^{\pm 0.02917}$	$0.15100^{\pm 0.02292}$	$5.89127^{\pm 0.35370}$	$0.43143^{\pm 0.03720}$	$0.18382^{\pm 0.02447}$	$0.09164^{\pm 0.00334}$	$0.00621^{\pm 0.00078}$
	TFormer	$0.61170^{\pm 0.02772}$	$\mathbf{0.23482}^{\pm 0.00793}$	$0.25140^{\pm 0.02553}$	$5.70179^{\pm 0.19700}$	$0.53171^{\pm 0.02517}$	$0.21448^{\pm 0.01091}$	$\mathbf{0.07654}^{\pm 0.00227}$	$\mathbf{0.00436}^{\pm 0.00010}$

Table C.10: Errors of the baseline and proposed models for different evaluation metrics, including RMSE, normalized RMSE (nRMSE), RMSE from conserved value (cRMSE), maximum error, RMSE at the boundaries (bRMSE), RMSE in Fourier space at low (fRMSE low), medium (fRMSE mid), and high frequency (fRMSE high) ranges on the 1D CFD equation. The unseen PDE parameters are highlighted in light grey.

D Software Versions

Listing D.1 Used software versions for the experiments.

```
Python @ 3.10.12
PyTorch @ 2.2.0.dev20231003
CUDA @ 11.7
```