Physics-Informed Transformer Networks

Fabricio Dos Santos[1] * **Tara Akhound-Sadegh**[1,2] * **Siamak Ravanbakhsh**[1,2]

[1]McGill University [2]Mila – Québec AI Institute

Abstract

Physics-informed neural networks (PINNs) have been recognized as a viable alternative to conventional numerical solvers for Partial Differential Equations (PDEs). The main appeal of PINNs is that since they directly enforce the PDE equation, one does not require access to costly ground truth solutions for training the model. However, a key challenge is their limited generalization across varied initial conditions. Addressing this, our study presents a novel Physics-Informed Transformer (PIT) model for learning the solution operator for PDEs. Using the attention mechanism, PIT learns to leverage the relationships between its initial condition and query points, resulting in a significant improvement in generalization. Moreover, in contrast to existing physics-informed networks, our model is invariant to the discretization of the input domain, providing great flexibility in problem specification and training. We validated our proposed method on the 1D Burgers' and the 2D Heat equations, demonstrating notable improvement over standard PINN models for operator learning with negligible computational overhead.

1 Introduction

Partial Differential Equations serve as fundamental tools for describing a wide array of physical phenomena. Finding accurate and efficient solutions of PDEs is crucial in numerous scientific and engineering domains, enabling the simulation, prediction, and optimization of complex systems. Since traditional numerical solvers often face significant challenges when dealing with complex spatiotemporal PDEs, innovative and efficient alternatives have become crucial in various areas of science [18, 7, 9].

In machine learning, data-driven PDE solvers have emerged as a promising avenue of research. These solvers harness the capabilities of machine-learning techniques to approximate the solutions of PDEs. One prominent approach is the use of Physics-Informed Neural Networks (PINNs) [15], which combine the power of neural networks with the constraints imposed by the governing PDEs. By enforcing the PDE at selected points, PINNs can effectively capture the underlying physics and yield accurate solutions in regions with limited or no data. With the work of DeepONets [14], the concept of operator learning has emerged as a novel paradigm for solving PDEs. This framework aims to learn the solution operator of PDEs, which enables solving PDEs with different initial or boundary conditions, or coefficients. Various architectures have been proposed in this context, such as Fourier Neural Operators [11], Graph Neural Operators [10] and the transformer-based OFormer [12]. There has also been work in combining operator learning with PINNs [20, 13].

Operator learning with PINNs is the only paradigm that does not need simulation data for training and does not require test time optimization as in vanilla PINNs. While these qualities make this setting quite attractive, the existing solutions do not perform well in practice and make limiting assumptions about the format of the initial and boundary conditions – i.e., they should lie on a fixed grid.

We address these problems using a Transformer architecture [17] while maintaining the tractability of derivatives through automatic differentiation. The proposed *Physics-Informed Transformer*, PIT, is

*Equal Contribution. Corresponding author: fabricio.dossantos@mail.mcgill.ca

DLDE-III Workshop in the 37th Conference on Neural Information Processing Systems (NeurIPS 2023).

invariant to the discretization of both input and query domains and allows for interactions between both domains through cross-attention blocks. Our empirical evaluations show significant improvements over the established physics-informed DeepONet, and suggest that the proposed model can perform well in higher dimensional problems even with a limited training budget.

2 Background

Physics-informed neural networks. In PINNs, we generally consider PDEs of the form:

$$\begin{aligned} \boldsymbol{u}_t + \mathcal{N}[\boldsymbol{u}] &= 0, \qquad t \in [0, T], \boldsymbol{x} \in \Omega, \\ \boldsymbol{u}(0, \boldsymbol{x}) &= \boldsymbol{g}(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega, \\ \mathcal{B}[\boldsymbol{u}] &= 0, \qquad t \in [0, T], \boldsymbol{x} \in \partial\Omega, \end{aligned} \tag{1}$$

where $\boldsymbol{u}(t, \boldsymbol{x}) \in \mathbb{R}^{d_u}$ is the solution to the PDE, t denotes time, $\boldsymbol{x}$ is a vector of spatial coordinates in the domain Ω, and $\mathcal{N}[\cdot]$ is a linear or nonlinear differential operator. The function $\boldsymbol{g}$ describes the initial condition (IC) of the PDE, and $\mathcal{B}[\cdot]$ is a boundary operator corresponding to Dirichlet, Neumann, Robin, or periodic boundary conditions.

In PINNs, the solution $\boldsymbol{u}(t, \boldsymbol{x})$ of the PDE is represented by a neural network $\boldsymbol{u}_\theta(t, \boldsymbol{x})$ with parameters θ. This model can then be trained by minimizing a loss with three components:

$$\mathcal{L}(\theta) = \lambda_{ic}\mathcal{L}_{ic}(\theta) + \lambda_{bc}\mathcal{L}_{bc}(\theta) + \lambda_r\mathcal{L}_r(\theta). \tag{2}$$

The first two terms consist of a supervised loss which guarantees that the function learned by the neural network satisfies the initial and boundary conditions of the problem. That is, given randomly sampled points $\{\boldsymbol{x}_{ic}^i\}_{i=1}^{N_{ic}}$ from Ω and $\{t_{bc}^i, \boldsymbol{x}_{bc}^i\}_{i=1}^{N_{bc}}$ from $[0, T] \times \partial\Omega$, we have:

$$\mathcal{L}_{ic}(\theta) = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} \|\boldsymbol{u}_\theta(0, \boldsymbol{x}_{ic}^i) - \boldsymbol{g}(\boldsymbol{x}_{ic}^i)\|_2^2, \qquad \mathcal{L}_{bc}(\theta) = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} \|\mathcal{B}[\boldsymbol{u}_\theta](t_{bc}^i, \boldsymbol{x}_{bc}^i)\|_2^2. \tag{3}$$

The third term is the physics-informed objective, ensuring that the learned function $\boldsymbol{u}_\theta(t, \boldsymbol{x})$ satisfies the PDE in Eq. (1). This is done by minimizing:

$$\mathcal{L}_r(\theta) = \frac{1}{N_r} \sum_{i=1}^{N_r} \left\| \frac{\partial \boldsymbol{u}_\theta}{\partial t}(t, \boldsymbol{x}) + \mathcal{N}[\boldsymbol{u}_\theta](t, \boldsymbol{x}) \right\|_2^2, \tag{4}$$

where the derivatives of $\boldsymbol{u}_\theta$ are calculated through automatic differentiation. This penalty is enforced on a set $\{t_r^i, \boldsymbol{x}_r^i\}_{i=1}^{N_r}$ of randomly sampled points from the domain. Furthermore, these losses are weighted by hyper-parameters $\{\lambda_{ic}, \lambda_{bc}, \lambda_r\}$, leading to more flexibility during training.

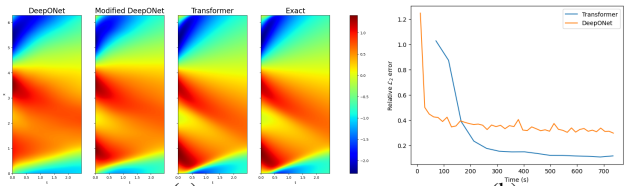
We provide some background on operator learning and transformers in §A.1 and §A.2.

3 Methods

Problem Setting. Given a PDE as described by Eq. (1), we want to use a transformer to learn an operator G_θ, taking an input function f and query point $(t, \boldsymbol{x})$, outputting the solution $\boldsymbol{u}(t, \boldsymbol{x})$ of the PDE at the query point, $G_\theta[f](t, \boldsymbol{x}) = \boldsymbol{u}(t, \boldsymbol{x})$. The transformer model takes a collection of query points $\{q_i\}$, for $q_i \in [0, T] \times \Omega$ in the domain, and samples $\{p_j, f(p_j)\}$ from an input function f, for p_j in the domain of f, and outputs the solution of the PDE at the query locations. The input and query domains can be the same, for example, when f is the initial condition (IC) of the PDE, or they can be different. We also note that since transformers operate on sets, the number of query and input samples and the discretization of the input function f can vary for each training instance. This is not possible for other operator learning architectures such as DeepONets [14].

Positional encoding. To make use of the spatial nature of the problem, we use positional encoding. While more recent and sophisticated types of positional encoding exist, like Rotary Position Embedding [16], we use sinusoidal positional encoding [17] for our experiments. This is performed on query points q and input points p and encodes each independent variable with sine and cosine functions of different frequencies. That is, each entry $v^{(i)}$ of a given vector $\boldsymbol{v}$ is assigned a vector $\boldsymbol{h}^{(i)}$ with entries $h_j^{(i)}$ calculated by:

$$\alpha_j = 10000^{-2\lfloor j/2 \rfloor / d_{pe}}, \qquad h_j^{(i)} = \begin{cases} \sin(\alpha_j v^{(i)}) & \text{if } j \text{ is even} \\ \cos(\alpha_j v^{(i)}) & \text{if } j \text{ is odd} \end{cases} \tag{5}$$



(a) (b)

Figure 1: 1D Burgers' equation: (a) Models' prediction and exact solution for a sample from the test dataset (b) Relative $\mathcal{L}_2$ error over the test data as a function of time for PIT and DeepONet.

where d_{pe} is the dimensionality of $\boldsymbol{h}^{(i)}$. The final output $\boldsymbol{h}$ of the positional encoding of the vector $\boldsymbol{v}$ is then a vectorization of the matrix with vectors $\boldsymbol{h}^{(i)}$ as columns. We thus obtain vectors $\boldsymbol{h}_q$ and $\boldsymbol{h}_p$, the positional encoding of the query and input points, respectively. In addition, we concatenate the value $f(p)$ to the corresponding positional encoding $\boldsymbol{h}_p$ to include it in the cross-attention operation.

Transformer architecture. The model architecture we use consists of an encoder and a decoder. The encoder is composed of two parts: point-wise MLPs ϕ and ψ, which produce embedding vectors for query points and input points, respectively, and L cross-attention blocks, which capture interactions between these two embeddings. Unlike the model proposed by Vaswani et al. [17] and Li et al. [12], we do not perform self-attention on the query and input points, as this would create interdependencies, increasing the cost of automatic differentiation needed to calculate the PINN loss. Although this computational cost would not be prohibitive for low dimensional PDEs, we didn't observe any benefits from including this self-attention.

Given positional encodings $\boldsymbol{h}_q$ and $\boldsymbol{h}_p$ of the query and input points, respectively, we first obtain high-dimensional embeddings $\boldsymbol{z}^{(0)} = \phi(\boldsymbol{h}_q)$ and $\boldsymbol{y} = \psi(\boldsymbol{h}_p)$, used to update the query embeddings:

$$\hat{\boldsymbol{z}}^{(l)} = \text{LayerNorm}(\boldsymbol{z}^{(l)}), \qquad \boldsymbol{z}^{(l+1)} = \boldsymbol{z}^{(l)} + \text{MLP}(\hat{\boldsymbol{z}}^{(l)} + \text{Cross-attention}(\hat{\boldsymbol{z}}^{(l)}, \boldsymbol{y})), \tag{6}$$

for $l = 1, ..., L-1$, where L is the number of cross-attention blocks, LayerNorm$(\cdot)$ is layer normalization [1], and Cross-attention$(\cdot)$ is the cross-attention mechanism discussed in this section.

The final output, $\boldsymbol{z}^{(L)}$, of the encoder then goes through a point-wise MLP γ whose output dimension is the same as the codomain of the PDE's solution, $\boldsymbol{u}(t, \boldsymbol{x})$. The predicted solution, $\boldsymbol{u}_\theta(t, \boldsymbol{x})$, is then given by $\gamma(\boldsymbol{z}^{(L)})$, for $\boldsymbol{z}^{(L)}$ obtained from the query point $q = (t, \boldsymbol{x})$ in question.

Attention mechanism. The attention mechanism [2, 17] is a mapping of query vectors $\{\boldsymbol{q}_i\}_{i=1}^m$ and of key-value pairs of vectors, $\{\boldsymbol{k}_j\}_{j=1}^n$ and $\{\boldsymbol{v}_j\}_{j=1}^n$. The attention map outputs a weighted sum of the values, where each value's weight is given by a function w and is a measure of the closeness of the query with the corresponding key. In this paper, we choose w to be the scaled dot-product [17] $w(\boldsymbol{q}_i, \boldsymbol{k}_j) = (\boldsymbol{q}_i \cdot \boldsymbol{k}_j)/\sqrt{d}$ with d being the dimension of queries and keys. A query vector $\boldsymbol{q}_i$ is thus mapped to an output $\boldsymbol{z}_i$ as follows:

$$\boldsymbol{z}_i = \sum_{j=1}^{n} \text{softmax}\left(\frac{\boldsymbol{q}_i \cdot \boldsymbol{k}_j}{\sqrt{d}}\right) \boldsymbol{v}_j. \tag{7}$$

We can compute the attention function for all queries simultaneously: $\boldsymbol{Z} = \text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top/\sqrt{d})\boldsymbol{V}$, by writing each i-th vector as the i-th column of its corresponding matrix. Although we acknowledge the recent improvements brought by alternative types of attention like Galerkin and Fourier [3], we use the standard softmax-normalized attention in our experiments.

In self-attention, the matrices $\boldsymbol{Q}, \boldsymbol{K}$ and $\boldsymbol{V}$ are computed as linear projections of the same input feature embedding. To capture interactions between the input domain and query domains (i.e. capture how solutions inside the domain are dependent on the IC), we use cross-attention, where the matrix $\boldsymbol{Q}$ is obtained from a different input than $\boldsymbol{K}$ and $\boldsymbol{V}$. In particular, we use a linear projection of an embedding of the query points $\{q_i\}$ to obtain $\boldsymbol{Q}$. Similarly, $\boldsymbol{K}$ and $\boldsymbol{V}$ are obtained through linear projections of an embedding of the samples $\{p_j, f(p_j)\}$ from the input function f. Furthermore, a linear projection of the query embeddings, $\boldsymbol{U}$, is added to the output of the attention operation to accentuate the influence of the query points further. In our experiments, MLPs are used instead of linear projections as they lead to better generalization.

Training. The model parameters, θ, are trained by minimizing the loss in Eq. (2). Each instance used for training contains a set $\{p_j, f(p_j)\}_{j=1}^M$ of samples from the input function, and query points

$\{r_i\}_{i=1}^{N_r}$, $\{s_i\}_{i=1}^{N_{ic}}$ and $\{b_i\}_{i=1}^{N_{bc}}$, taken from the PDE domain, used to evaluate $\mathcal{L}_r$, $\mathcal{L}_{ic}$ and $\mathcal{L}_{bc}$ respectively. As f is the IC of the PDE, the need for the points $\{s_i\}$ is removed. In contrast with other works [14], in PIT the number of samples, M, N_r, N_{ic} and N_{bc}, and their location, p_j, r_i, s_i and b_i, do not need to be the same throughout different instances. This means that the discretization of the PDE input domains can change for each training sample.

4 Experiments

In this section, we compare the performance of PIT with the conventional physics-informed DeepONet [20], and the physics-informed DeepONet with improved architecture introduced in Wang et al. [19], which we refer as "modified DeepONet". We conduct experiments on two PDEs: 1D Burgers' and 2D heat equations. PIT shows significant improvements compared to the other models, without introducing significant computing costs.

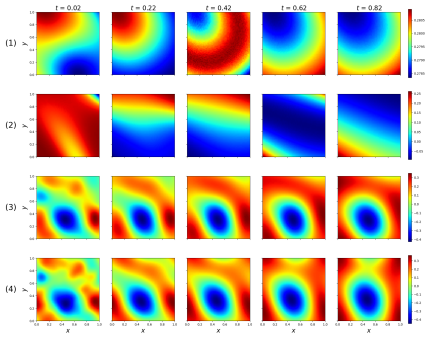


Figure 2: Predictions for an IC of 2D heat equation test dataset for (1) DeepONet, (2) Modified DeepONet, (3) PIT ; and exact solution (4).

Burgers' equation The 1D Burgers' equation describes the behaviour of fluid flow in one dimension:

$$u_t(t,x) + u(t,x)u_x(t,x) - \nu u_{xx}(t,x) = 0, \quad (8)$$

for $\nu > 0$ the diffusion coefficient. We consider $\nu = 0.01$, and the spatial and time domains are $\Omega = [0, 2\pi]$ and $[0, T] = [0, 2.475]$. Additional details on the experiment can be found in §A.3.

In Table 1, we compare the performance of PIT, DeepONet and modified DeepONet on the test data. The PIT outperforms the other models with a relative $\mathcal{L}_2$ error of 0.044 over 400 test examples. A sample of the models' predictions can be found in Fig. 1. In addition, Fig. 1 shows that PIT is not significantly more expensive than DeepONet.

Heat equation The 2D heat equation describes the heat conduction in a 2D spatial domain:

$$u_t(t,x,y) - \nu(u_{xx}(t,x,y) + u_{yy}(t,x,y)) = 0, \quad (9)$$

for $\nu > 0$ the thermal diffusivity constant. We use $\nu = 0.01$, and $\Omega = [0, 1]^2$ and $[0, T] = [0, 1]$. We assume periodic spatial boundaries. We provide additional details on this experiment in §A.3.

We find that physics-informed DeepONet is not expressive enough for this problem and does not yield meaningful results. Introducing the modified architecture does not sufficiently improve results and is more costly in this higher-dimensional case. On the other hand, PIT performs well even though the number of training samples is not large. We compare the models' performance in Table 1 and show a sample of their predictions in Fig. 2.

Table 1: Mean and standard deviation of the relative $\mathcal{L}_2$ errors, calculated over 400 examples in the test dataset for Burgers' and 200 examples for the heat equation.

	Relative $\mathcal{L}_2$ error	
Model	1D Burgers	2D Heat
DeepONet	0.266 ± 0.096	4.506 ± 1.281
Modified DeepONet	0.158 ± 0.092	4.202 ± 1.167
PIT (ours)	$\mathbf{0.044 \pm 0.023}$	$\mathbf{0.340 \pm 0.066}$

5 Conclusion

We introduce PIT for PDE operator learning. This model is fully trained with a PINN loss and thus does not require access to the ground-truth solution, which can be difficult and costly to obtain. In addition, our method allows for any discretization of both the input function domain and PDE domain, providing flexibility in both specifying the initial conditions and sampling the input domain during training and deployment. Due to cross-attention, PIT has linear complexity in the points sampled from the input domain and the initial condition points. While for a large number of sampled points and initial conditions, this may become prohibitive, for common use cases considered in our experiments, the cost is comparable to other operator-learning PINNs. We show that PIT performs well even with a limited number of sampled points used to enforce the loss, which is especially useful when dealing with higher-dimensional PDEs.

Acknowledgments

This research is in part supported by the Canada CIFAR AI Chair, NSERC discovery grant, and SURA Undergraduate Research Awards. Computational resources are provided by Mila and the Digital Research Alliance of Canada.

References

[1] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] S. Cao. Choose a transformer: Fourier or galerkin, 2021.

[4] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE transactions on neural networks*, 6 4:911–7, 1995. URL `https://api.semanticscholar.org/CorpusID:15993548`.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[6] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021. doi: 10.1038/s41586-021-03819-2.

[7] K. Kashinath, M. Mustafa, A. Albert, J.-L. Wu, C. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, R. Wang, A. Chattopadhyay, A. Singh, A. Manepalli, D. Chirila, R. Yu, R. Walters, B. White, H. Xiao, H. A. Tchelepi, P. Marcus, A. Anandkumar, P. Hassanzadeh, and Prabhat. Physics-informed machine learning: case studies for weather and climate modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*.

[8] G. Kissas, J. Seidman, L. F. Guilhoto, V. M. Preciado, G. J. Pappas, and P. Perdikaris. Learning operators with coupled attention, 2022.

[9] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. Machine learning – accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021. doi: 10.1073/pnas.2101784118.

[10] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations, 2020.

[11] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

[12] Z. Li, K. Meidani, and A. B. Farimani. Transformer for partial differential equations' operator learning, 2023.

[13] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar. Physics-informed neural operator for learning partial differential equations, 2023.

[14] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, mar 2021. doi: 10.1038/s42256-021-00302-5. URL `https://doi.org/10.1038%2Fs42256-021-00302-5`.

[15] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL `https://www.sciencedirect.com/science/article/pii/S0021999118307125`.

[16] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding, 2022.

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

[18] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu. Towards physics-informed deep learning for turbulent flow prediction, 2020.

[19] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021. doi: 10.1137/20M1318043. URL https://doi.org/10.1137/20M1318043.

[20] S. Wang, H. Wang, and P. Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets, 2021.

A Appendix

A.1 Background on Operator Learning

In DeepONets [14], the universal operator approximation theorem [4] is applied to PDE solving with different initial or boundary conditions. The goal is to learn an operator G between two infinite dimensional spaces from a finite collection of observed input-output pairs. This operator G takes an input function f and outputs a function $G[f] : [0, T] \times \Omega \to \mathbb{R}^{d_u}$ which corresponds to the solution $\boldsymbol{u}(t, \boldsymbol{x})$ of the PDE when evaluated at the point $(t, \boldsymbol{x})$. This framework can be combined with the PINN loss [20] to ensure physical consistency and reduce the need for large training datasets. Many other works explore the use of neural operators to solve PDEs. One such work is Fourier Neural Operators (FNO) [11], where an integral kernel parametrized in Fourier space is used. The authors in Physics-Informed Neural Operators (PINO) [13] further combine FNO with physical constraints. Most relevant to us is the use of attention in this setting [12, 3, 8] where the transformer architecture is applied to operator learning for PDEs.

A.2 Background on Transformers

Transformer models have become increasingly popular in the machine-learning world. Originally developed for natural language processing, the attention mechanism [2, 17] has produced promising results in different tasks, such as computer vision [5] and protein folding [6]. In particular, multiple works have investigated the use of attention in physical systems. In Kissas et al. [8], attention weights are coupled with an integral transform to approximate nonlinear operators. In Cao [3], Galerkin-type and Fourier-type attention are studied in the context of operator learning. In particular, in OFormers [12], a transformer architecture is used for PDEs' operator learning. In contrast to these works, we consider the application of transformers for PINNs, where we do not assume any access to ground truth data for training.

A.3 Experiment details

Burgers' equation. We represent the IC functions by truncated Fourier series with coefficients A_k, l_k, ϕ_k sampled randomly, and $K = 10$:

$$u(0, x) = \sum_{k=1}^{K} A_k \sin(2\pi l_k x/L + \phi_k).$$

The data used for evaluation is obtained using the Fourier Spectral method with periodic spatial boundaries, and $\nu = 0.01$ for the diffusion coefficient. The spatial domain $[0, 2\pi]$ and time domain $[0, 2.475]$ are discretized uniformly into 256 and 100 points, respectively. We use 600 ICs for training and 400 for testing. In addition, we take $N_r = 250$, $N_{ic} = 200$ and $N_{bc} = 200$ for enforcing the residual, initial and boundary losses, respectively.

Heat equation. Initial conditions are sampled from a Gaussian Random Field prior. Data used for the evaluation of the model is generated using the Fourier Spectral method. The spatial domain $[0, 1]^2$ is discretized by a uniform 64×64 grid, and the time domain $[0, 1]$ is discretized into 50 points. We use 800 ICs for training and 200 for testing. For training, we use 400 random points from inside the grid to enforce the residual loss and 400 points for the IC loss. The boundary loss is enforced directly by computing $|u(t, 0, y) - u(t, 1, y)|$ and $|u(t, x, 0) - u(t, x, 1)|$ on 50 different points for each.