Composing Partial Differential Equations with Physics-Aware Neural Networks

Matthias Karlbauer[*,1] **Timothy Praditia**[*,2] **Sebastian Otte**[1] **Sergey Oladyshkin**[2] **Wolfgang Nowak**[2] **Martin V. Butz**[1]

Abstract

We introduce a compositional physics-aware FInite volume Neural Network (FINN) for learning spatiotemporal advection-diffusion processes. FINN implements a new way of combining the learning abilities of artificial neural networks with physical and structural knowledge from numerical simulation by modeling the constituents of partial differential equations (PDEs) in a compositional manner. Results on both one- and two-dimensional PDEs (Burgers', diffusion-sorption, diffusion-reaction, Allen–Cahn) demonstrate FINN's superior modeling accuracy and excellent out-of-distribution generalization ability beyond initial and boundary conditions. With only one tenth of the number of parameters on average, FINN outperforms pure machine learning and other state-of-the-art physics-aware models in all cases—often even by multiple orders of magnitude. Moreover, FINN outperforms a calibrated physical model when approximating sparse real-world data in a diffusion-sorption scenario, confirming its generalization abilities and showing explanatory potential by revealing the unknown retardation factor of the observed process.

1. Introduction

Artificial neural networks (ANNs) are considered universal function approximators (Cybenko, 1989). Their effective learning ability, however, greatly depends on domain and task-specific prestructuring and methodological modifications referred to as inductive biases (Battaglia et al., 2018). Typically, inductive biases limit the space of possible models by reducing the opportunities for computational shortcuts that might otherwise lead to erroneous implications derived from a potentially limited dataset (overfitting). The recently evolving field of physics-informed machine learning employs physical knowledge as inductive bias providing vast generalization advantages in contrast to pure machine learning (ML) in physical domains (Raissi et al., 2019). While numerous approaches have been introduced to augment ANNs with physical knowledge, these methods either do not allow the incorporation of explicitly defined physical equations (Long et al., 2018; Seo et al., 2019; Guen & Thome, 2020; Li et al., 2020a; Sitzmann et al., 2020) or cannot generalize to other initial and boundary conditions than those encountered during training (Raissi et al., 2019).

In this work, we present the FInite volume Neural Network (FINN)—a physics-aware ANN adhering to the idea of spatial and temporal discretization in numerical simulation. FINN consists of multiple neural network modules that interact in a distributed, compositional manner (Lake et al., 2017; Battaglia et al., 2018; Lake, 2019). The modules are designed to account for specific parts of advection-diffusion equations, a class of partial differential equations (PDEs). This modularization allows us to combine two advantages that are not yet met by state-of-the-art models: The explicit incorporation of physical knowledge and the generalization over initial and boundary conditions. To the best of our knowledge, FINN's ability to adjust to different initial and boundary conditions and to explicitly learn constitutive relationships and reaction terms is unique, yielding excellent out-of-distribution generalization. The core contributions of this work are:

- Introduction of FINN, a physics-aware ANN, explicitly designed to generalize over initial and boundary conditions, yielding excellent generalization ability.
- Evaluation of state-of-the-art pure ML and physics-aware models, contrasted to FINN on one- and two-dimensional benchmarks, demonstrating the benefit of explicit model design.
- Application of FINN to a real-world contamination-diffusion problem, verifying its applicability to real, spatially and temporally constrained training data.

[*]Equal contribution [1]Neuro-Cognitive Modeling, University of Tübingen, Tübingen, Germany [2]Department of Stochastic Simulation and Safety Research for Hydrosystems, University of Stuttgart, Stuttgart, Germany. Correspondence to: Matthias Karlbauer <matthias.karlbauer@uni-tuebingen.de>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

2. Related Work

Non-Physics-Aware ANNs Pure ML models that are designed for spatiotemporal data processing can be separated into temporal convolution (TCN, Kalchbrenner et al., 2016) and recurrent neural networks. While the former perform convolutions over space and time, representatives of the latter, e.g., convolutional LSTM (ConvLSTM, Shi et al., 2015) or DISTANA (Karlbauer et al., 2019), aggregate spatial neighbor information to further process the temporal component with recurrent units. Since pure ML models do not adhere to physical principles, they require large amounts of training data and parameters in order to approximate a desired physical process; but still are not guaranteed to behave consistently outside the regime of the training data.

Physics-Aware ANNs When designed to satisfy physical equations, physics-aware ANNs are reported to have greater robustness in terms of physical plausibility. For example, the physics-informed neural network (PINN, Raissi et al., 2019) consists of an MLP that satisfies an explicitly defined PDE with specific initial and boundary conditions, using automatic differentiation. However, the remarkable results beyond the time steps encountered during training are limited to the very particular PDE and its conditions. A trained PINN cannot be applied to different initial conditions, which limits its applicability in real-world scenarios.

Other methods by Long et al. (PDENet, 2018), Guen & Thome (PhyDNet, 2020), or Sitzmann et al. (SIREN, 2020) learn the first n derivatives to achieve a physically plausible behavior, but lack the option to include physical equations. The same limitation holds when operators are learned to approximate PDEs (Li et al., 2020a;b), or when physics-aware graph neural networks are applied (Seo et al., 2019). Yin et al. (APHYNITY, 2020) suggest to approximate equations with an appropriate physical model and to augment the result by an ANN, preventing the ANN to approximate a distinct part within the physical model. For more comparison to related work, please refer to subsection B.1.

In summary, none of the above methods can explicitly learn particular constitutive relationships or reaction terms while simultaneously generalizing beyond different initial and boundary conditions.

3. Finite Volume Neural Network (FINN)

Problem Formulation Here, we focus on modeling spatiotemporal physical processes. Specifically, we consider systems governed by advection-diffusion type equations (Smolarkiewicz, 1983), defined as:

$$\frac{\partial u}{\partial t} = D(u)\frac{\partial^2 u}{\partial x^2} - v(u)\frac{\partial u}{\partial x} + q(u), \tag{1}$$

where u is the quantity of interest (e.g. salt distributed in water), t is time, x is the spatial coordinate, D is the diffusion coefficient, v is the advection velocity, and q is the source/sink term. Eq. 1 can be partitioned into three parts: The storage term $\frac{\partial u}{\partial t}$ describes the change of the quantity u over time. The flux terms are the advective flux $v(u)\frac{\partial u}{\partial x}$ and the diffusive flux $D(u)\frac{\partial^2 u}{\partial x^2}$. Both calculate the amount of u exchanged between neighboring volumes. The source/sink term $q(u)$ describes the generation or elimination of the quantity u. Eq. 1 is a general form of PDEs with up to second order spatial derivatives, but it has a wide range of applicability due to the flexibility of defining $D(u)$, $v(u)$, and $q(u)$ as different functions of u, as is shown by the numerical experiments in this work.

The finite volume method (FVM, Moukalled et al., 2016) discretizes a simulation domain into control volumes ($i = 1, \dots, N_x$), where exchange fluxes are calculated using a surface integral (Riley et al., 2009). In order to match this structure in FINN, we introduce two different kernels, which are (spatially) casted across the discretized control volumes: the flux kernel, modeling the flux terms (i.e. lateral quantity exchange), and the state kernel, modeling the source/sink term as well as the storage term. The overall FINN architecture is shown in Figure 1.

Flux Kernel The flux kernel $\mathcal{F}$ approximates the surface integral for each control volume i with boundary Ω by a composition of multiple subkernels f_j, each representing the flux through a discretized surface element j:

$$\mathcal{F}_i = \sum_{j=1}^{N_{s_i}} f_j \approx \oint_{\omega \subseteq \Omega} \left(D(u)\frac{\partial^2 u}{\partial x^2} - v(u)\frac{\partial u}{\partial x} \right) \cdot \hat{n}\, d\Gamma, \tag{2}$$

where N_{s_i} is the number of discrete surface elements of control volume i, ω is a continuous surface element (a subset of Ω), f_j are subkernels (consisting of feedforward network modules), and $\hat{n}$ is the unit normal vector pointing outwards of ω.

In our exemplary one-dimensional arrangement, two subkernels f_{i-} and f_{i+} (see Figure 1) contain the modules $\varphi_{\mathcal{D}}$, $\varphi_{\mathcal{A}}$, and $\varphi_{\mathcal{N}}$. Module $\varphi_{\mathcal{N}}$ is a linear layer with the purpose to approximate the first spatial derivative $\frac{\partial u}{\partial x}$, i.e.

$$\frac{\partial u_i}{\partial x} \approx \begin{cases} \varphi_{\mathcal{N}}(u_i, u_{i-1}) & \text{on } f_{i-} \\ \varphi_{\mathcal{N}}(u_i, u_{i+1}) & \text{on } f_{i+} \end{cases}. \tag{3}$$

Technically, $\varphi_{\mathcal{N}}$ is supposed to learn the numerical FVM stencil, being nothing else but the difference between its inputs, i.e. the quantity at two neighboring control volumes in ideal one-dimensional problems. This signifies that the weights of $\varphi_{\mathcal{N}}$ should amount to $[-1, 1]$ with respect to $[u_i, u_{i-1}]$ and $[u_i, u_{i+1}]$ in order to output their difference.

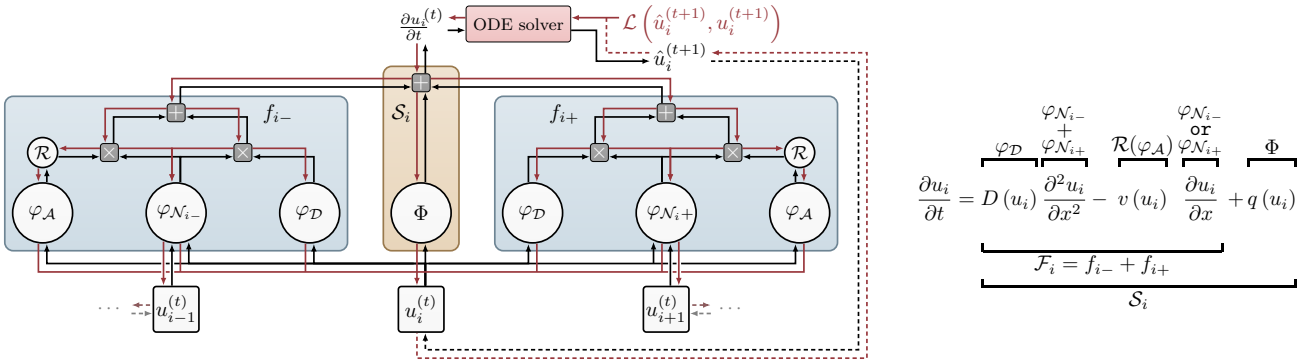


Figure 1: Flux (blue) and state (orange) kernels in FINN for the one-dimensional case (left)—red lines indicate gradient flow—and detailed assignment of the individual modules with their contribution to Eq. 1 (right).

Module $\varphi_{\mathcal{D}}$, receiving u_i as input, is responsible for the diffusive flux (the process of quantity homogenization from areas of high to low concentration). In case the diffusion coefficient D depends on u, the module is designed as feedforward network, such that $\varphi_{\mathcal{D}}(u) \approx D(u)$. Otherwise, $\varphi_{\mathcal{D}}$ is set as scalar value, which can be set trainable if the value of D is unknown.

Things are more complicated for advective flux, representing bulk motion and thus quantity that enters volume i either from left *or* right; module $\mathcal{R}$ decides on this based on the output of $\varphi_{\mathcal{A}}$ (which itself is a feedforward network, similarly to $\varphi_{\mathcal{D}}$). Technically, module $\mathcal{R}$ applies an upwind differencing scheme to prevent numerical instability in the first order spatial derivative calculation (Versteeg & Malalasekera, 1995) and is computed as

$$\mathcal{R}\left(\varphi_{\mathcal{A}}\left(u_i\right)\right) = \begin{cases} \mathrm{ReLU}\left(\varphi_{\mathcal{A}}\left(u_i\right)\right) & \text{on } f_{i-} \\ -\mathrm{ReLU}\left(-\varphi_{\mathcal{A}}\left(u_i\right)\right) & \text{on } f_{i+} \end{cases}. \tag{4}$$

It ensures that further processing of the advective flux from $\varphi_{\mathcal{A}}(u_i)$ is performed only on one control volume surface (either left or right), by switching on only the left surface element when $\varphi_{\mathcal{A}} > 0$ or only the right surface element when $\varphi_{\mathcal{A}} < 0$.

Finally, considering Eq. 4, the flux calculations turn into

$$f_{i-} = \varphi_{\mathcal{N}}(u_i, u_{i-1}) \cdot \Big(\varphi_{\mathcal{D}}(u_i) + \mathcal{R}\big(\varphi_{\mathcal{A}}(u_i)\big)\Big) \tag{5}$$

$$f_{i+} = \varphi_{\mathcal{N}}(u_i, u_{i+1}) \cdot \Big(\varphi_{\mathcal{D}}(u_i) + \mathcal{R}\big(\varphi_{\mathcal{A}}(u_i)\big)\Big) \tag{6}$$

$$\mathcal{F}_i = f_{i-} + f_{i+}. \tag{7}$$

Since the advective flux is only considered on one side of volume i (due to module $\mathcal{R}$), the summation of the numerical stencil from both f_{i-} and f_{i+} in Eq. 7 leads to $[-1, 1]$, being applied to $[u_i, u_{i-1}]$ when $\varphi_{\mathcal{A}} > 0$, or $[u_i, u_{i+1}]$ when $\varphi_{\mathcal{A}} < 0$ (i.e. only first order spatial derivative). For the diffusive flux calculation, on the other hand, the summation of the numerical stencil leads to the classical one-dimensional numerical Laplacian with $[1, -2, 1]$ applied to $[u_{i-1}, u_i, u_{i+1}]$, representing the second order spatial derivative, since the calculation of $\varphi_{\mathcal{D}}$ is performed on both surfaces (see subsection B.3 for a derivation). Altogether, module $\mathcal{R}$ generates the inductive bias to make $\varphi_{\mathcal{A}}$ only approximate the advective, and $\varphi_{\mathcal{D}}$ the diffusive flux.

Boundary Conditions A means of applying boundary conditions in a model is essential when solving PDEs. Currently available models mostly adopt convolution operations to model spatiotemporal processes. However, convolutions only allow a constant value to be padded at the domain boundaries (e.g. zero-padding or mirror-padding), which is only appropriate for the implementation of Dirichlet or periodic boundary condition types. Other types of frequently used boundary conditions are Neumann and Cauchy. These are defined as a derivative of the quantity of interest, and hence cannot be easily implemented in convolutional models. However, with certain pre-defined boundary condition types in FINN, the flux kernels at the boundaries are adjusted accordingly to allow for straightforward boundary condition consideration. For Dirichlet boundary condition, a constant value $u = u_b$ is set as the input u_{i-1} (for the flux kernel f_{i-}) or u_{i+1} (for f_{i+}) at the corresponding boundary. For Neumann boundary condition ν, the output of the flux kernel f_{i-} or f_{i+} at the corresponding boundary is set to be equal to ν. With Cauchy boundary condition, the solution-dependent derivative is calculated and set as u_{i-1} or u_{i+1} at the corresponding boundary.

State Kernel The state kernel $\mathcal{S}$ calculates the source/sink and storage terms of Eq. 1. The source/sink (if required) is learned using the module $\Phi(u) \approx q(u)$. The storage, $\frac{\partial u}{\partial t}$, is then calculated using the output of the flux kernel and module Φ of the state kernel:

$$\mathcal{S}_i = \mathcal{F}_i(u_{i-1}, u_i, u_{i+1}) + \Phi(u_i) \approx \frac{\partial u_i}{\partial t}. \tag{8}$$

By doing so, the PDE in Eq. 1 is now reduced to a system of coupled ordinary differential equations (ODEs), which are functions of u_{i-1}, u_i, u_{i+1}, and t. Thus, the solutions of the coupled ODE system can be computed via numerical integration over time. Since first order explicit approaches, such as the Euler method (Butcher, 2008), suffer from numerical instability (Courant et al., 1967; Isaacson & Keller, 1994), we employ the neural ordinary differential equation method (Neural ODE, Chen et al., 2018) to reduce numerical instability via the Runge–Kutta adaptive time-stepping strategy. The Neural ODE evaluates $\frac{\partial u_i}{\partial t}$ in form of FINN at an arbitrarily fine Δt and integrates FINN's output over time from t to $t+1$; the resulting u_i^{t+1} is fed back into the network as input in the next time step, until the last time step is reached. Thus, the entire training is performed in closed-loop, improving stability and accuracy of the prediction compared to networks trained with teacher forcing, i.e. one-step-ahead prediction (Praditia et al., 2020). The weight update is based on the mean squared error (MSE) as loss and realized by applying backpropagation through time (indicated by red arrows in Figure 1). In short, FINN takes only the initial condition u at time $t = 0$ and propagates the dynamics forward interactively with Neural ODE.

4. Experiments, Results, and Discussion

4.1. Synthetic Dataset

To demonstrate FINN's performance in comparison to other models, four different equations are considered as applications. First, the two-dimensional *Burgers' equation* (Basdevant et al., 1986) is chosen as a challenging function, as it is a non-linear PDE with $v(u) = u$ that could lead to a shock in the solution $u(x, y, t)$. Second, the *diffusion-sorption equation* (Nowak & Guthke, 2016) is selected with the non-linear retardation factor $R(u)$ as coefficient for the storage term, which contains a singularity $R(u) \to \infty$ for $u \to 0$ due to the parameter choice. Third, the two-dimensional Fitzhugh–Nagumo equation (Klaasen & Troy, 1984) as candidate for a *diffusion-reaction equation* (Turing, 1952) is selected, which is challenging because it consists of two non-linearly coupled PDEs to solve two main unknowns: the activator u_1 and the inhibitor u_2. Fourth, the Allen–Cahn equation with a cubic reaction term is chosen, leading to multiple jumps in the solution $u(x, t)$. Details on all four equations, data generation, and architecture designs can be found in Appendix C.

For each problem, three different datasets are generated (by conventional numerical simulation): *train*, used to train the models, in-distribution test (*in-dis-test*), being the train data simulated with a longer time span to test the models' generalization ability (extrapolation), and out-of-distribution test (*out-dis-test*). *Out-dis-test* data are used to test a trained ML model under conditions that are far away from training conditions, not only in terms of querying outputs for unseen inputs. Instead, *out-dis-test* data query outputs with regards to changes *not* captured by the inputs. These are changes that the ML tool per definition cannot be made aware of during training. In this work, they are represented by data generated with different initial or boundary condition, to test the generalization ability of the models outside the training distributions. FINN is trained and compared with both spatiotemporal deep learning models such as TCN, ConvLSTM, DISTANA and physics-aware neural network models such as PINN and PhyDNet. All models are trained with ten different random seeds using PyTorch's default weight initialization. Mean and standard deviation of the prediction errors are summarized in Table 1 for *train*, *in-dis-test* and *out-dis-test*. Details of each run are reported in the appendix. It is noteworthy that PINN cannot be tested on the *out-dis-test* dataset, since PINN assumes that the unknown variable u is an explicit function of x and t, and hence, when the initial or boundary condition is changed, the function will also be different and no longer valid.

4.1.1. Results

Burgers' The predictions of the best trained model of each method for the *in-dis-test* and the *out-dis-test* data are shown in Figure 2 and Figure 3, respectively. Both TCN and ConvLSTM fail to produce reasonable predictions, but qualitatively the other models manage to capture the shape of the data sufficiently, even towards the end of the *in-dis-test* period, where closed loop prediction has been applied for 380 time steps (after 20 steps of teacher forcing, see subsection C.1 for details). DISTANA, PINN, and FINN stand out in particular, but FINN produces more consistent and accurate predictions, evidenced by the mean value of the prediction errors. When tested against data generated with a different initial condition (*out-dis-test*), all models except for TCN and PhyDNet perform well. However, FINN still outperforms the other models with a significantly lower prediction error. The advective velocity learned by FINN's module $\varphi_{\mathcal{A}}$ is shown in Figure 5 (top left) and verifies that it successfully learned the advective velocity to be described by an identity function.

Diffusion-Sorption The predictions of the best trained model of each method for the concentration u from the diffusion-sorption equation are shown in Figure 12 of the appendix. TCN and ConvLSTM are shown to perform poorly even on the *train* data, evidenced by the high mean value of the prediction errors. On *in-dis-test* data, all models successfully produce relatively accurate predictions. However, when tested against different boundary conditions (*out-dis-test*), only FINN is able to capture the modifications and generalize well. The other models are shown to still overfit to the different boundary condition used in

Table 1: Comparison of relative MSE (rMSE, which is the MSE divided by the variance) and standard deviation scores across ten repetitions between different deep learning (above dashed line) and physics-aware neural networks (below dashed line) on different equations. Best results reported in bold.

Equation	Model	Params	Dataset		
			Train	*In-dis-test*	*Out-dis-test*
Burgers' 2D	TCN	29 690	$(3.9 \pm 1.3) \times 10^{-2}$	$(3.1 \pm 0.8) \times 10^{-1}$	$(9.3 \pm 2.1) \times 10^{-2}$
	ConvLSTM	22 600	$(2.4 \pm 3.7) \times 10^{-1}$	$(6.0 \pm 6.7) \times 10^{0}$	$(4.2 \pm 3.5) \times 10^{-1}$
	DISTANA	19 100	$(7.3 \pm 11.0) \times 10^{-3}$	$(2.9 \pm 4.3) \times 10^{-1}$	$(3.6 \pm 3.6) \times 10^{-2}$
	PINN	3 041	$(1.8 \pm 0.8) \times 10^{-1}$	$(5.8 \pm 2.5) \times 10^{-1}$	—
	PhyDNet	185 300	$(1.1 \pm 0.3) \times 10^{-3}$	$(2.6 \pm 2.4) \times 10^{-1}$	$(3.5 \pm 1.5) \times 10^{-1}$
	FINN	**421**	$\mathbf{(1.0 \pm 0.6) \times 10^{-4}}$	$\mathbf{(1.1 \pm 0.4) \times 10^{-2}}$	$\mathbf{(2.4 \pm 1.0) \times 10^{-5}}$
Diffusion-sorption 1D	TCN	3 834	$(1.6 \pm 2.2) \times 10^{0}$	$(1.8 \pm 2.5) \times 10^{0}$	$(3.3 \pm 4.2) \times 10^{0}$
	ConvLSTM	3 960	$(5.1 \pm 4.6) \times 10^{-1}$	$(4.4 \pm 3.4) \times 10^{-1}$	$(1.8 \pm 1.2) \times 10^{0}$
	DISTANA	3 739	$(7.4 \pm 3.9) \times 10^{-4}$	$(3.5 \pm 3.6) \times 10^{-2}$	$(1.4 \pm 1.4) \times 10^{-1}$
	PINN	3 042	$(7.5 \pm 13.5) \times 10^{-4}$	$(6.1 \pm 12.8) \times 10^{-2}$	—
	PhyDNet	37 815	$\mathbf{(5.6 \pm 2.7) \times 10^{-4}}$	$(1.3 \pm 2.3) \times 10^{-1}$	$(5.0 \pm 2.8) \times 10^{-1}$
	FINN	**528**	$(7.6 \pm 7.4) \times 10^{-4}$	$\mathbf{(1.9 \pm 1.9) \times 10^{-3}}$	$\mathbf{(1.2 \pm 1.1) \times 10^{-3}}$
Diffusion-reaction 2D	TCN	31 734	$(1.9 \pm 1.2) \times 10^{-1}$	$(3.8 \pm 1.7) \times 10^{0}$	$(4.4 \pm 2.6) \times 10^{0}$
	ConvLSTM	24 440	$(1.2 \pm 2.8) \times 10^{-1}$	$(7.4 \pm 3.9) \times 10^{-1}$	$(4.4 \pm 3.8) \times 10^{-1}$
	DISTANA	75 629	$(5.3 \pm 4.5) \times 10^{-2}$	$(1.4 \pm 0.5) \times 10^{0}$	$(3.9 \pm 2.6) \times 10^{-1}$
	PINN	3 062	$(3.6 \pm 2.5) \times 10^{-3}$	$(5.6 \pm 5.0) \times 10^{-1}$	—
	PhyDNet	185 589	$\mathbf{(1.0 \pm 0.1 \times 10^{-3}}$	$(6.2 \pm 1.4) \times 10^{-1}$	$(1.0 \pm 0.4) \times 10^{0}$
	FINN	**882**	$(1.7 \pm 0.4) \times 10^{-3}$	$\mathbf{(1.6 \pm 0.4) \times 10^{-2}}$	$\mathbf{(1.8 \pm 0.1) \times 10^{-1}}$
Allen–Cahn 1D	TCN	10 052	$(4.4 \pm 9.1) \times 10^{-1}$	$(5.1 \pm 5.4) \times 10^{-1}$	$(2.8 \pm 3.8) \times 10^{-1}$
	ConvLSTM	7 600	$(2.8 \pm 5.3) \times 10^{-1}$	$(8.8 \pm 5.4) \times 10^{-1}$	$(4.0 \pm 4.5) \times 10^{-1}$
	DISTANA	6 422	$(2.3 \pm 1.0) \times 10^{-3}$	$(1.3 \pm 0.5) \times 10^{-1}$	$(5.4 \pm 3.5) \times 10^{-2}$
	PINN	3 021	$(9.7 \pm 6.2) \times 10^{-5}$	$(1.2 \pm 1.9) \times 10^{-1}$	—
	PhyDNet	37 718	$(4.6 \pm 3.0) \times 10^{-4}$	$(1.7 \pm 0.6) \times 10^{-1}$	$(7.5 \pm 2.1) \times 10^{-1}$
	FINN	**422**	$\mathbf{(3.2 \pm 3.5) \times 10^{-5}}$	$\mathbf{(3.5 \pm 4.2) \times 10^{-5}}$	$\mathbf{(3.6 \pm 4.5) \times 10^{-5}}$

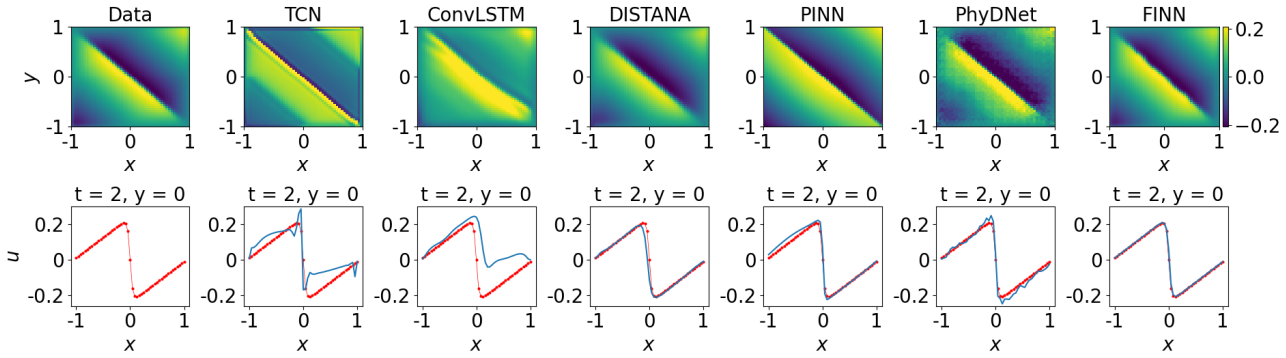


Figure 2: Plots of Burgers' data (red) and *in-dis-test* (blue) prediction using different models. The first row shows the solution over x and y at $t = 2$, and the second row visualizes the best model's solution distributed in x at $y = 0$ and $t = 2$.

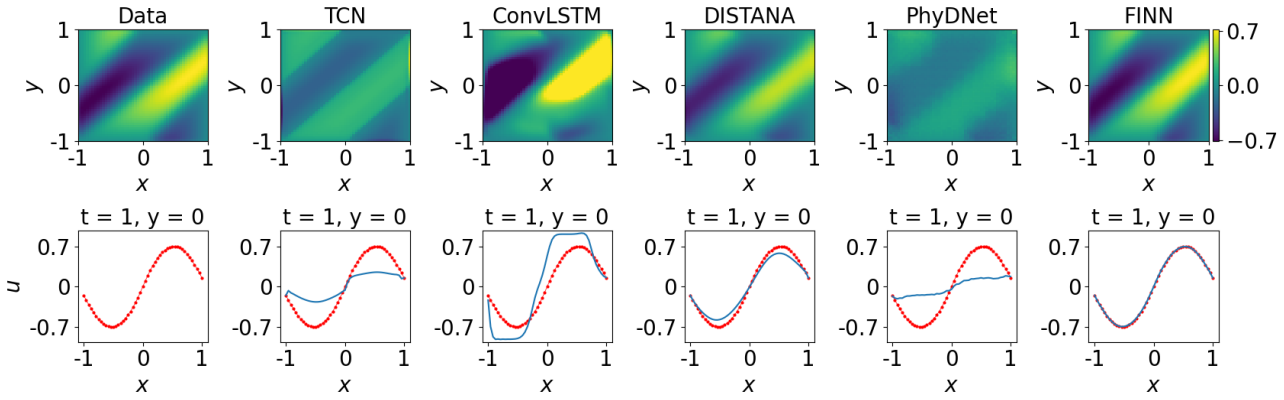


Figure 3: Plots of Burgers' data (red) and prediction (blue) of *out-dis-test* data using different models. The first row shows the solution over x and y at $t = 1$, and the second row visualizes the best model's solution over x at $y = 0$ and $t = 1$.

the train data (as detailed in subsection C.2). The retardation factor $R(u)$ learned by FINN's $\varphi_{\mathcal{D}}$ module is shown in Figure 5 (top center). The plot shows that the module learned the Freundlich retardation factor with reasonable accuracy.

Diffusion-Reaction The predictions of the best trained model of each method for activator u_1 in the diffusion-reaction equation are shown in Figure 15 (appendix) and Figure 4. TCN is again shown to fail to learn sufficiently from the *train* data. On *in-dis-test* data, DISTANA and the physics-aware models all predict with reasonable accuracy. When tested against data with different initial condition (*out-dis-test*), however, DISTANA and PhyDNet produce predictions with lower accuracy, and we find that FINN is the only model producing relatively low prediction errors. The reaction functions learned by FINN's Φ module are shown in Figure 5 (bottom). The plots show that the module successfully learned the Fitzhugh–Nagumo reaction function, both for the activator and inhibitor.

Allen–Cahn Results on the Allen–Cahn equation mostly align with those on Burgers' equation, confirming prior findings. However, it is worth noting that, again, only FINN is able to clearly represent the multiple nonlinearities caused by the exponent of third order in the reaction term, as visualized in Figure 16 and Figure 17 of the appendix. Again, Figure 5 (top right) shows that FINN accurately learned the dataset's reaction function.

4.1.2. Discussion

Overall, even with a high number of parameters, the prediction errors obtained using the pure ML methods (TCN, ConvLSTM, DISTANA) are worse compared to the physics-aware models, as shown in Table 1. As a physics-aware model, PhyDNet also possesses a high number of parameters. However, most of the parameters are allocated to the data-driven part (i.e. ConvLSTM branch), compared to the physics-aware part (i.e. PhyCell branch). In contrast to the other pure ML methods, DISTANA predicts with higher accuracy. This could act as an evidence that appropriate structural design of a neural network is as essential as physical knowledge to regularize the model training.

Among the physics-aware models, PINN and PhyDNet lie on different extremes. On one side, PINN requires complete knowledge of the modelled system in form of the equation. This means that all the functions, such as the advective velocity in Burgers', the retardation factor in the diffusion-sorption, and the reaction functions in the diffusion-reaction and Allen–Cahn equations, have to be pre-defined in order to train the model. This leaves less room for learning from data and could be error-prone if the designer's assumption is incorrect. On the other side, PhyDNet relies more heavily on the data-driven part and, therefore, could overfit the data. This can be shown by the fact that PhyDNet reaches the lowest training errors for the diffusion-sorption and diffusion-reaction equation predictions compared to the other models, but its performance significantly deteriorates when applied to *in-* and *out-dis-test* data. Our proposed model, FINN, lies somewhere in between these two extremes, compromising between putting sufficient physical knowledge into the model and leaving room for learning from data. As a consequence, we observe FINN showing excellent generalization ability. It outperforms other models up to multiple orders of magnitude, especially on *out-dis-test* data, when tested with different initial and boundary conditions; which is considered a particularly challenging task for ML models. Furthermore, the structure of FINN allows the extractions of learned functions such as the advective velocity, retardation factor, and reaction functions, showing good model interpretability.

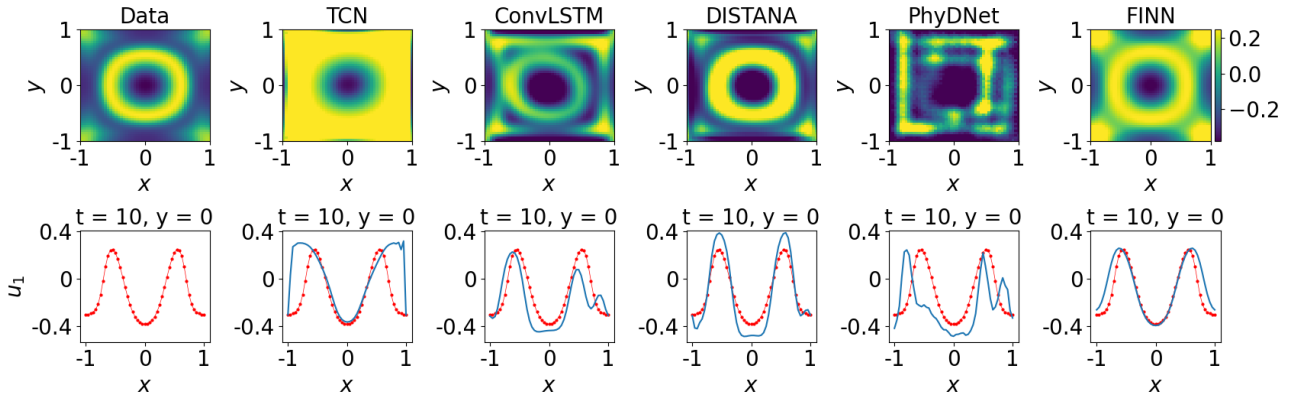


Figure 4: Plots of diffusion-reaction's activator data (red) and prediction (blue) of unseen dataset using different models. The plots in the first row show the solution distributed over x and y at $t = 10$, and the plots in the second row show the solution distributed in x at $y = 0$ and $t = 10$.

We also show that FINN properly handles the provided numerical boundary condition, as evidenced when applied to the test data that is generated with a different left boundary condition value, visualized in Figure 12 (appendix). Here, the test data is generated with a Dirichlet boundary condition $u(0, t) = 0.7$, which is lower than the value used in the train data, $u(0, t) = 1.0$. In fact, FINN is the only model that appropriately processes this different boundary condition value so that the prediction fits the test data nicely. The other models overestimate their prediction by consistently showing a tendency to still fit the prediction to the higher boundary condition value encountered during training.

Even though the spatial resolution used for the synthetic data generation is relatively coarse, leading to sparse data availability, FINN generalizes well. PINN, on the other hand, slightly suffers from the low resolution of the train data, although it still shows reasonable test performance.

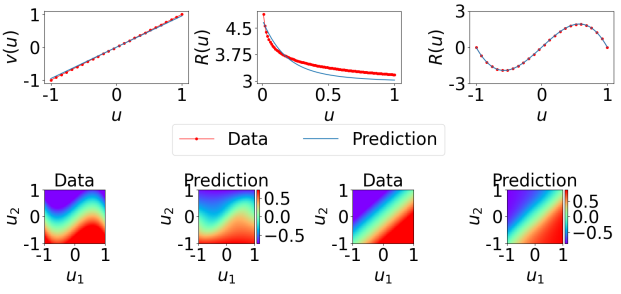


Figure 5: Plots of the learned functions (blue) as a function of u compared to the data (red) for Burgers' (top left), diffusion-sorption (top center), and Allen–Cahn (top right). The learned activator u_1 and inhibitor u_2 reaction functions in the diffusion-reaction equation are contrasted to the corresponding ground truth (second row).

Nevertheless, we conducted an experiment showing that PINN performs slightly better and more consistently when trained on higher resolution data (see appendix, subsection D.4), albeit still worse than FINN on coarse data. Therefore, we conclude that FINN is also applicable to real observation data that are often available only in low resolution, and/or in limited quantity. We demonstrate this further in subsection 4.2, when we apply FINN to real experimental data. FINN's generalization ability is superior to PINN, due to the fact that it is not possible to apply a trained PINN model to predict data with different initial or boundary condition. Additionally, we compare conservation errors on Burgers' for all models and find that FINN has the lowest with 7.12×10^{-3} (cf. PINN with 9.72×10^{-2}), even though PINN is explicitly trained to minimize conservation errors.

In terms of interpretability, FINN allows the extraction of functions learned by its dedicated modules. These learned functions can be compared with the real functions that generated the data (at least in the synthetic data case); examples are shown in Figure 5. The learned functions are the main data-driven discovery part of FINN and can also be used as a physical plausibility check. PhyDNet also comprises of a physics-aware and a data-driven part. However, it is difficult, if not impossible, to infer the learned equation from the model. Furthermore, the data-driven part of PhyDNet does not possess comparable interpretability, and could lead to overfitting as discussed earlier.

4.1.3. Ablations

In a number of ablation studies, we shed light on the relevance of particular choices in FINN: We substantiate the use of feedforward modules over polynomial regression in subsection D.1, proof FINN's ability to adequately learn unknown constituents under noisy conditions in subsec-

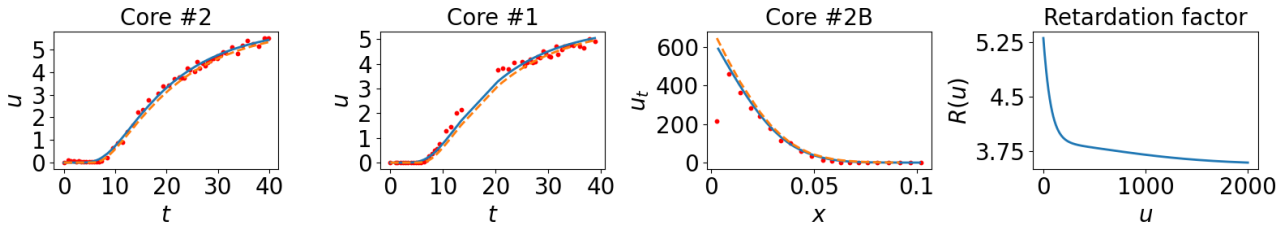


Figure 6: Breakthrough curve prediction of FINN (blue line) during training using data from core sample #2 (left), during testing using data from core sample #1 (second from left) and total concentration profile prediction using data from core sample #2B (second from right). The predictions are compared with the experimental data (red circles) and the results obtained using the physical model (orange dashed line). The right-most plot shows the learned retardation factor $R(u)$.

tion D.2, and consolidate the advantages of applying Neural ODE compared to traditional closed-loop application and Euler integration in subsection D.3.

4.2. Experimental Dataset

Real observation data are often available in limited amount, and they only provide partial information about the system of interest. To demonstrate how FINN can cope with real observation data, we use experimental data for the diffusion-sorption problem. The experimental data are collected from three different core samples that are taken from the same geographical area: #1,#2, and #2B (see subsection C.5 in the appendix for details). The objective of this experiment is to learn the retardation factor function from one of the core samples that concurrently applies to all the other samples. For this particular purpose, we only implement FINN since the other models have no means of learning the retardation factor explicitly. Here, we use the module $\varphi_{\mathcal{D}}$ to learn the retardation factor function, and we assume that the diffusion coefficient values of all the core samples are known. FINN is trained with the breakthrough curve of u, which is the dissolved concentration only at $x = L|_{0 \leq t \leq t_{\text{end}}}$ (i.e. only 55 data points).

Results and Discussion FINN reaches a higher accuracy for the training with core sample #2, with $\text{rMSE} = 3.38 \times 10^{-3}$ compared to a calibrated physical model from Nowak & Guthke (2016) with $\text{rMSE} = 7.42 \times 10^{-3}$, because the latter has to assume a specific function $R(u)$ with a few parameters for calibration. Our learned retardation factor is then applied and tested to core samples #1 and #2B. Figure 6 shows that FINN's prediction accuracy is competitive compared to the calibrated physical model. For core sample #1, FINN's prediction has an accuracy of 8.28×10^{-3} compared to the physical model that underestimates the breakthough curve (i.e. concentration profile) with $\text{rMSE} = 1.52 \times 10^{-2}$. Core sample #2B has significantly longer length than the other samples, and therefore a no-flow Neumann boundary condition was assumed at the bottom of the core. Because there is no breakthrough curve data available for this specific sample, we compare the prediction against the so-called total concentration profile $u(x, t_{\text{end}})$ at the end of the experiment. FINN produced a prediction with an accuracy of 6.39×10^{-2}, whereas the physical model overestimates the concentration with $\text{rMSE} = 1.50 \times 10^{-1}$. To briefly summarize, FINN is able to learn the retardation factor from a sparse experimental dataset and apply it to other core samples with similar soil properties with reasonable accuracy, even with a different boundary condition type.

5. Conclusion

Spatiotemporal dynamics often can be described by means of advection-diffusion type equations, such as Burgers', diffusion-sorption, or diffusion-reaction equations. When modeling those dynamics with ANNs, large efforts must be taken to prevent the models from overfitting (given the model is able to learn the problem at all). The incorporation of physical knowledge as regularization yields robust predictions beyond training data distributions.

With FINN, we have introduced a modular, physics-aware ANN with excellent generalization abilities beyond different initial and boundary conditions, when contrasted to pure ML models and other physics-aware models. FINN is able to model and extract unknown constituents of differential equations, allowing high interpretability and an assessment of the plausibility of the model's out-of-distribution predictions. As next steps we seek to apply FINN beyond second order spatial derivatives, improve its scalability to large datasets, and make it applicable to heterogeneously distributed data (i.e. represented as graphs) by modifying the module $\varphi_{\mathcal{N}}$ to approximate variable and location-specific stencils (for more details on limitations of FINN, the reader is referred to subsection B.2). Another promising future direction is the application of FINN to real-world sea-surface temperature and weather data.

Software and Data

Code and data that are used for this paper can be found in the repository https://github.com/CognitiveModeling/finn.

Acknowledgements

We thank the reviewers for constructive feedback, leading us to evaluate all methods on two-dimensional instead of one-dimensional Burgers', to additionally benchmark APHINITY (Yin et al., 2020), FNO (Li et al., 2020a), and cvPINN (Patel et al., 2022), and to perform a CNN-NODE ablation study. Results are reported in Figure 7 and Table 2 and discussed in the according sections.

This work is funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2075 – 390740016 as well as EXC 2064 – 390727645. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech). Moreover, we thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Matthias Karlbauer.

References

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

Basdevant, C., Deville, M., Haldenwang, P., Lacroix, J., Ouazzani, J., Peyret, R., Orlandi, P., and Patera, A. Spectral and finite difference solutions of the burgers equation. *Computers & Fluids*, 14(1):23–41, 1986. doi: https://doi.org/10.1016/0045-7930(86)90036-8.

Battaglia, P., Hamrick, J., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Butcher, J. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2008. ISBN 9780470753750.

Chen, R., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

Courant, R., Friedrichs, K., and Lewy, H. On the partial difference equations of mathematical physics. *IBM Journal of Research and Development*, 11(2):215–234, 1967. doi: 10.1147/rd.112.0215.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Fornberg, B. Generation of finite difference formulas on arbitrarily spaced grids. *Mathematics of computation*, 51(184):699–706, 1988.

Guen, V. and Thome, N. Disentangling physical dynamics from unknown factors for unsupervised video prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11474–11484, 2020.

Isaacson, E. and Keller, H. *Analysis of Numerical Methods*. Dover Books on Mathematics. Dover Publications, 1994. ISBN 9780486680293.

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A., Graves, A., and Kavukcuoglu, K. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

Karlbauer, M., Otte, S., Lensch, H., Scholten, T., Wulfmeyer, V., and Butz, M. A distributed neural network architecture for robust non-linear spatio-temporal prediction. *arXiv preprint arXiv:1912.11141*, 2019.

Klaasen, G. and Troy, W. Stationary wave solutions of a system of reaction-diffusion equations derived from the fitzhugh–nagumo equations. *SIAM Journal on Applied Mathematics*, 44(1):96–110, 1984. doi: 10.1137/0144008.

Kochkov, D., Smith, J., Alieva, A., Wang, Q., Brenner, M., and Hoyer, S. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

Lake, B. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems 32*, pp. 9791–9801. 2019.

Lake, B., Ullman, T., Tenenbaum, J., and Gershman, S. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 2017. doi: 10.1017/S0140525X16001837.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

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

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

Moukalled, F., Mangani, L., and Darwish, M. *The Finite Volume Method in Computational Fluid Dynamics*. Springer, 1 edition, 2016. doi: 10.1007/978-3-319-16874-6.

Nowak, W. and Guthke, A. Entropy-based experimental design for optimal model discrimination in the geosciences. *Entropy*, 18(11), 2016. doi: 10.3390/e18110409.

Patel, R. G., Manickam, I., Trask, N. A., Wood, M. A., Lee, M., Tomas, I., and Cyr, E. C. Thermodynamically consistent physics-informed neural networks for hyperbolic systems. *Journal of Computational Physics*, 449: 110754, 2022.

Praditia, T., Walser, T., Oladyshkin, S., and Nowak, W. Improving thermochemical energy storage dynamics forecast with physics-inspired neural network architecture. *Energies*, 13(15):3873, 2020. doi: 10.3390/en13153873.

Praditia, T., Karlbauer, M., Otte, S., Oladyshkin, S., Butz, M., and Nowak, W. Finite volume neural network: Modeling subsurface contaminant transport. *arXiv preprint arXiv:2104.06010*, 2021.

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

Riley, K., Hobson, M., and Bence, S. *Mathematical Methods for Physics and Engineering*. Cambridge University Press, 2009. ISBN 9780521139878.

Seo, S., Meng, C., and Liu, Y. Physics-aware difference graph networks for sparsely-observed dynamics. In *International Conference on Learning Representations*, 2019.

Shi, X., Chen, Z., Wang, H., Yeung, D., Wong, W., and Woo, W. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *arXiv preprint arXiv:1506.04214*, 2015.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

Smolarkiewicz, P. A simple positive definite advection scheme with small implicit diffusion. *Monthly Weather Review*, 111(3):479 – 486, 1983. doi: 10.1175/1520-0493.

Turing, A. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society B*, 237:37–72, 1952.

Versteeg, H. and Malalasekera, W. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. New York, 1995. ISBN 9780470235157.

Yin, Y., Guen, V., Dona, J., Ayed, I., de Bézenac, E., Thome, N., and Gallinari, P. Augmenting physical models with deep networks for complex dynamics forecasting. *arXiv preprint arXiv:2010.04456*, 2020.

Zhuang, J., Kochkov, D., Bar-Sinai, Y., Brenner, M., and Hoyer, S. Learned discretizations for passive scalar advection in a two-dimensional turbulent flow. *Physical Review Fluids*, 6(6):064605, 2021.

A. Appendix

This appendix is structured as follows: Additional detailed differences between FINN and the benchmark models used in this work are presented in subsection B.1. Then, limitations of FINN are discussed briefly in subsection B.2. Detailed numerical derivation of the first and second order spatial derivative is shown in subsection B.3. Additional details on the equations used as well as model specifications and in-depth results are presented in subsection C.1 (Burgers'), subsection C.2 (diffusion-sorption), subsection C.3 (diffusion-reaction), and subsection C.4 (Allen–Cahn). Moreover, the soil parameters and simulation domain used in the diffusion-sorption laboratory experiment are presented in subsection C.5. The results of our ablation studies are reported in Appendix D: Additional polynomial-fitting baselines to account for the equations are outlined in subsection D.1, including an ablation where we replace the neural network modules of FINN by polynomials. A robustness analysis of FINN can be found in subsection D.2, where our method is evaluated on noisy data from all equations. The role of the Neural ODE module is evaluated in subsection D.3, accompanied with a runtime analysis. Finally, results of training PINN with high-resolution data and training PhyDNet with the original architecture (more parameters) are reported in subsection D.4 and subsection D.5, respectively.

B. Methodological Supplements

B.1. Distinction to Related Work

Pure ML models Originally, FINN was inspired by DISTANA, a pure ML model proposed by Karlbauer et al. (2019). While DISTANA has large similarities to Shi et al. (2015)'s ConvLSTM, it propagates lateral information through an additional latent map—instead of reading lateral information from the input map directly, as done in ConvLSTM—and aggregates this lateral information by means of a user-defined combination of arbitrary layers. Accordingly, the processing of the two-point flux approximation in FINN, using the lateral information, is more akin to DISTANA, albeit motivated and augmented by physical knowledge. As a result, the lateral information flow is guaranteed to behave in a physically plausible manner. That is, quantity can either be locally generated by a source (increased), absorbed by a sink (decreased), or spatially distributed (move to neighboring cells).

Terminology separates the spatially distribution of quantity into diffusion and advection. Diffusion describes the equalization of quantity from high to low concentration levels, whereas advection is defined as the bulk motion of a large group of particles/atoms caused by external forces. FINN ensures the conservation of laterally propagating quantity, i.e. what flows from left to right will effectively cause a decrease of quantity at the left and an increase at the right. Pure ML models (without physical constraints) cannot be guaranteed to adhere to these fundamental rules.

PINN Since ML models guided by physical knowledge not only behave empirically plausible but also require fewer parameters and generalize to much broader extent, numerous approaches have been proposed recently to combine artificial neural networks with physical knowledge. Raissi et al. (2019) introduced the physics-informed neural network (PINN), a concrete and outstanding model that explicitly learns a provided PDE. As a result, PINN mimics e.g. Burgers' equation (see Eq. 15) by learning the quantity function $u(x, y, t)$ for defined initial and boundary conditions with a feedforward network. The partial derivatives are implicitly realized by automatic differentiation of the feedforward network (representing u) with respect to the input variables x, y, or t. The learned neural network thus satisfies the constraints defined by the partial derivatives that are formulated in the loss term.

Accordingly, PINN has the advantage to explicitly provide a (continuous) solution of the desired function $u(x, y, t)$ and, correspondingly, predictions can be queried for an arbitrary combination of input values, circumventing the need for simulating the entire domain with e.g. a carefully chosen simulation step size. However, this advantage prevents a trained PINN from being applied to data with different initial and boundary conditions than those of the training data; a limitation that still holds for successors such as the recently proposed control volume discretization PINN (cvPINN, Patel et al., 2022) that is reported to behave well on challenging shock problems. While cvPINN is able to maintain the effect of a particular boundary condition, it still cannot generalize to unseen boundary conditions, as outlined in Figure 7. In contrast, FINN does not learn the explicit function $u(x, y, t)$ defined for particular initial and boundary conditions, but approximates the distinct components of the function. These components are combined—as suggested by the physical equation—to result in a compositional model that is more universally applicable. That is, in stark contrast to PINN, the compositional function learned by FINN can be applied to varying initial and boundary conditions, since the learned individual components provide the same functionality as the corresponding components of the PDE when processed by a numerical solver. Applying the conventional numerical solver (e.g. FVM) would require either complete knowledge of the equation or careful calibration of the unknowns by choosing equations from a library of possible solutions and tuning the parameters. But FINN's data driven component can reveal unknown relations, such as the retardation factor of a function, without the need for subjective prior assumptions.

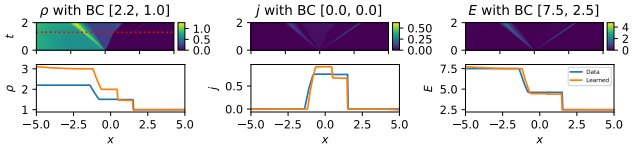


Figure 7: Results of cvPINN with left BC of ρ changed from 3.0 to 2.2. Top: MSE between ground truth and prediction. Bottom: u states in timestep $t = 1.0$ (step 128 in simulation). Code adapted from https://github.com/rgp62/cvpinns.

Thus, FINN paves the way for non-experts in numerical simulation to accurately model physical phenomena by learning unknown constituents on the fly that otherwise require both sophisticated domain knowledge and intensive analysis of the problem at hand.

Learning Derivatives via Convolution An alternative and much addressed approach of implanting physical plausibilty into artificial neural networks is to implicitly learn the first n derivatives using appropriately designed convolution kernels (Long et al., 2018; Li et al., 2020a;b; Guen & Thome, 2020; Yin et al., 2020; Sitzmann et al., 2020). These methods exploit the link that most PDEs, such as $u(t, x, y, \dots)$, can be reformulated as

$$\frac{\partial u}{\partial t} = F\left(t, x, y, \dots, u, \frac{\partial u}{\partial x}, \frac{\partial u}{y}, \frac{\partial u}{\partial x \partial y}, \frac{\partial^2 u}{\partial x^2}, \dots\right). \tag{9}$$

When learning partial derivatives up to order n and setting irrelevant features to zero, these methods have, in principle, the capacity to represent most PDEs. However, the degrees of freedom in these methods are still very high and can fail to safely guide the ML algorithm. FINN accounts for the first and second derivative by learning the corresponding stencil in the module $\varphi_{\mathcal{N}}$ and combining this stencil with the case-sensitive ReLU module $\mathcal{R}$, which allows a precise control of the information flow resulting from the first and second derivative. More importantly, convolutions only allows implementation of Dirichlet and periodic boundary conditions (by means of zero- or mirror-padding), and is not appropriate for implementing other boundary condition types.

Learning ODE Coefficients The group around Brenner and Hoyer (Bar-Sinai et al., 2019; Kochkov et al., 2021; Zhuang et al., 2021) follow another line of research about learning the coefficients of ordinary differential equations (ODEs); note that PDEs can be transformed into a set of coupled ODEs by means of polynomial expansion or spectral differentiation as shown in Bar-Sinai et al. (2019). The physical constraint in these works is mostly realized by satisfying the temporal derivative as loss.

While this approach shares similarities with our method by incorporating physically motivated inductive bias into the model, it uses this bias mainly to improve interpolation from coarse to finer grid resolutions and thus to accelerate simulation. Our work focuses on discovering unknown relationships/laws (or re-discovering laws in the case of the synthetic examples), such as the advective velocity in the Burgers' example, the retardation factor function in the diffusion-sorption example, and the reaction function in the diffusion-reaction and Allen–Cahn example. Additionally, the works by Brenner and Hoyer employ a convolutional structure, which is only applicable to Dirichlet or periodic boundary conditions, and it suffers from a slight instability during training when the training data trajectory is unrolled for a longer period. In contrast, FINN employs the flux kernel, calculated at all control volume surfaces, which enables the implementation and discovery of various boundary conditions. Furthermore and in contrast to Brenner and Hoyer, FINN employs the Neural ODE method as the time integrator to reduce numerical instability during training with long time series. In accord with this line of research, FINN is also able to generalize well when trained with a relatively sparse dataset (coarse resolution), reducing the computational burden.

Numerical ODE Solvers Traditional numerical solvers for ordinary differential equations (ODEs) can be seen as the pure-physics contrast to FINN. However, these do not have a learning capacity to reveal unknown dependencies or functions from data. Also, as shown in (Yin et al., 2020), a pure Neural ODE without physical inductive bias does not reach the same level of accuracy as physics-aware neural network models. We confirm this finding with an ablation (see subsection D.3), where FINN is replaced by a CNN whose output is processed by Neural ODE (NODE), reaching worse results. Furthermore, the relevance of the data-driven component in FINN is underlined by our field experiment, where FINN reached lower errors on a real-world dataset compared to a conventional FVM model.

APHYNITY and FNO We also have conducted another series of experiments to quantitatively compare two additional state-of-the-art methods. APHYNITY is a physics-aware model that consists of a physics-part that can be augmented by a neural network component to compensate phenomena that are not covered by the physics-part. It was developed by Yin et al. (2020), the same group that has developed PhyDNet before. On the other hand, the Fourier neural operator (FNO) model, introduced by Li et al. (2020a), is a pure ML model that learns concrete functions of space and time in frequency domain, e.g. $u = f(x, t)$. This is similar to e.g. PINN and SIREN (Raissi et al., 2019; Sitzmann et al., 2020) and results in a continuous representation of the trained simulation domain for any input combi-

Table 2: Comparison of rMSE and standard deviation scores across ten repetitions between CNN-NODE, FNO, APHYNITY, and FINN on different equations. Best results reported in bold.

Equation	Model	Params	Dataset		
			Train	*In-dis-test*	*Out-dis-test*
Burgers' 2D	CNN-NODE	2 657	$(2.0 \pm 0.7) \times 10^{-3}$	$(4.7 \pm 6.5) \times 10^{-1}$	$(9.1 \pm 9.5) \times 10^{-1}$
	FNO	116 115	$(6.4 \pm 9.2) \times 10^{-4}$	$(5.9 \pm 2.6) \times 10^{-2}$	$(9.7 \pm 0.9) \times 10^{-1}$
	APHYNITY	2 658	$(1.0 \pm 0.3) \times 10^{-2}$	$(4.3 \pm 0.1) \times 10^{-2}$	$(4.4 \pm 0.5) \times 10^{-4}$
	FINN	**421**	$\mathbf{(1.0 \pm 0.6) \times 10^{-4}}$	$\mathbf{(1.1 \pm 0.4) \times 10^{-2}}$	$\mathbf{(2.4 \pm 1.0) \times 10^{-5}}$
Diffusion-sorption 1D	CNN-NODE	1 026	$(6.8 \pm 14.3) \times 10^{-2}$	$(3.8 \pm 7.7) \times 10^{0}$	$(6.6 \pm 12.9) \times 10^{0}$
	FNO	5 878	$(1.1 \pm 2.8) \times 10^{-2}$	$(2.1 \pm 3.9) \times 10^{-2}$	$(4.1 \pm 1.4) \times 10^{-1}$
	APHYNITY	—	—	—	—
	FINN	**528**	$\mathbf{(7.6 \pm 7.4) \times 10^{-4}}$	$\mathbf{(1.9 \pm 1.9) \times 10^{-3}}$	$\mathbf{(1.2 \pm 1.1) \times 10^{-3}}$
Diffusion-reaction 2D	CNN-NODE	2 946	$(5.1 \pm 0.8) \times 10^{-1}$	$(2.3 \pm 0.9) \times 10^{1}$	$(3.3 \pm 0.8) \times 10^{0}$
	FNO	116 288	$(1.3 \pm 0.5) \times 10^{-4}$	$(4.8 \pm 11.0) \times 10^{1}$	$(2.9 \pm 1.0) \times 10^{0}$
	APHYNITY	2 949	$(2.9 \pm 0.5) \times 10^{-3}$	$(4.0 \pm 1.2) \times 10^{-1}$	$\mathbf{(8.0 \pm 1.8) \times 10^{-2}}$
	FINN	**882**	$\mathbf{(1.7 \pm 0.4) \times 10^{-3}}$	$\mathbf{(1.6 \pm 0.4) \times 10^{-2}}$	$(1.8 \pm 0.1) \times 10^{-1}$
Allen–Cahn 1D	CNN-NODE	929	$(4.2 \pm 2.6) \times 10^{-5}$	$(7.7 \pm 3.2) \times 10^{-2}$	$(8.1 \pm 5.4) \times 10^{-1}$
	FNO	5 745	$(3.3 \pm 3.2) \times 10^{-3}$	$(1.2 \pm 0.2) \times 10^{-1}$	$(1.5 \pm 0.1) \times 10^{0}$
	APHYNITY	930	$(1.2 \pm 0.0) \times 10^{-4}$	$(2.7 \pm 0.0) \times 10^{-3}$	$(6.7 \pm 0.1) \times 10^{-4}$
	FINN	**422**	$\mathbf{(3.2 \pm 3.5) \times 10^{-5}}$	$\mathbf{(3.5 \pm 4.2) \times 10^{-5}}$	$\mathbf{(3.6 \pm 4.5) \times 10^{-5}}$

nation of x and t.

Results are reported in Table 2 and support our previous findings, further fostering and demonstrating FINN's superior performance. Indeed, FNO and APHYNITY perform very well (FNO on extrapolation, APHYNITY on generalization), but they do not reach FINN's accuracy. Note that for the diffusion-reaction equation, APHYNITY outperforms FINN when tested with the *out-dis-test* data. This can be attributed to the fact that APHYNITY is trained assuming complete knowledge of the modelled equation. When parts of the modelled equation are unknown, APHINITY performs worse than FINN. This demonstrates that APHYNITY is very accurate when the full correct equation is known, whereas FINN can learn unknown parts of the equation better. Additionally, APHINITY fails to be trained with diffusion-sorption data due to instability issues.

B.2. Limitations of FINN

First, while the largest limitation of our current method can be seen in the capacity to only represent first and second order spatial derivatives, this is an issue that we will address in follow up work. Still, FINN can already be applied to a very wide range of problems as most equations in fact only depend on up to the second spatial derivative. *Second*, FINN is only applicable to spatially homogeneously distributed data so far—we intend to extend it to heterogeneous data on graphs. *Third*, although we have successfully applied FINN to two-dimensional Burgers' and diffusion-reaction data, the training time is considerable, albeit comparable to other physics-aware ANNs. While, according to our observations, this appears to be a common issue for physics-aware neural networks (cf. Table 17), an optimized implementation on today's hardware-accelerated computation could potentially widen the bottleneck.

B.3. Learning the Numerical Stencil

Semantically, the $\varphi_{\mathcal{N}}$ module learns the numerical stencil, that is the geometrical arrangement of a group of neighboring cells to approximate the derivative numerically, effectively learning the first spatial derivative $\frac{\partial u}{\partial x}$ from both $[u_{i-1}, u_i]$ and $[u_i, u_{i+1}]$, which are the inputs to the $\varphi_{\mathcal{N}}$ module.

The lateral information flowing from u_{i-1} and u_{i+1} toward u_i is controlled by the $\varphi_{\mathcal{A}}$ (advective flux, i.e. bulk motion of many particles/atoms that can either move to the left or to the right) and the $\varphi_{\mathcal{D}}$ (diffusive flux, i.e. drive of particles/atoms to equilibrium from regions of high to low concentration) modules. Since the advective flux can only move either to the left or to the right, it will be considered only in the left flux kernel (f_{i-}) or in the right flux kernel (f_{i+}), and not both at the same time. The case-sensitive ReLU module $\mathcal{R}$ (Eq. 4) decides on this, by setting the advective flux in the irrelevant flux kernel to zero (effectively depending on the sign of the output of $\varphi_{\mathcal{A}}$). Thus, the advective flux is only considered from either u_{i-1} or u_{i+1} to

u_i, which amounts to the first order spatial derivative.

The diffusive flux, on the other hand, can propagate from both sides towards the control volume of interest u_i and, hence, the second order spatial derivative, accounting for the difference between u_{i-1} and u_{i+1}, has to be applied. In our method, this is realized through the combination of the $\varphi_\mathcal{N}$ and $\varphi_\mathcal{D}$ modules, calculating the diffusive fluxes $\delta_- = \varphi_\mathcal{N}(u_i, u_{i-1})\varphi_\mathcal{D}(u_i)$ and $\delta_+ = \varphi_\mathcal{N}(u_i, u_{i+1})\varphi_\mathcal{D}(u_i)$ inside of the respective left and right flux kernel. The combination of these two deltas in the state kernel ensures the consideration of the diffusive fluxes from left (including u_{i-1}) and from right (including u_{i+1}), resulting in the ability to account for the second order spatial derivative.

Technically, the coefficients for the first, i.e. $[-1, 1]$, and second, i.e. $[1, -2, 1]$, order spatial differentiation schemes are common definitions and a derivation can be found, for example, in Fornberg (1988). However, a quick derivation of the Laplace scheme $[1, -2, 1]$ can be formulated as follows. Define the second order spatial derivative as the difference between two first order spatial derivatives, i.e.

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{(\partial u/\partial x)|_{i-} - (\partial u/\partial x)|_{i+}}{\Delta x} \tag{10}$$

with the two first order spatial derivatives—representing the differences between u_{i-1}, u_i (left, i.e. 'minus') and u_i, u_{i+1} (right, i.e. 'plus')—defined as

$$(\partial u/\partial x)|_{i-} \approx (u_{i-1} - u_i)/\Delta x \tag{11}$$

$$(\partial u/\partial x)|_{i+} \approx (u_i - u_{i+1})/\Delta x. \tag{12}$$

Then, substituting Eq. 11 and Eq. 12 into Eq. 10, we get

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{(u_{i-1} - u_i) - (u_i - u_{i+1})}{\Delta x^2} \tag{13}$$

$$\approx \frac{u_{i-1} - 2u_i + u_{i+1}}{\Delta x^2}, \tag{14}$$

hence the $[+1, -2, +1]$ as coefficients in the second order spatial derivative. In fact, in the Burgers' experiment, we find that FINN learns the appropriate stencils, i.e. $[-0.99 \pm 0.04, 0.99 \pm 0.04]$ for the first order spatial derivative.

C. Data and Model Details

C.1. Burgers'

The Burgers' equation is commonly employed in various research areas, including fluid mechanics.

Data The two-dimensional generalized Burgers' equation is written as

$$\frac{\partial u}{\partial t} = -v(u)\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) + D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \tag{15}$$

where the main unknown is u, the advective velocity is denoted as $v(u)$ which is a function of u and the diffusion coefficient is $D = 0.01/\pi$. In the current work, the advective velocity function is chosen to be an identity function $v(u) = u$ to reproduce the experiment conducted in the PINN paper (Raissi et al., 2019).

The simulation domain for the ***train*** **data** is defined with $x = [-1, 1]$, $y = [-1, 1]$, $t = [0, 1]$ and is discretized with $N_x = 49$ and $N_y = 49$ spatial locations, and $N_t = 201$ simulation steps. The initial condition was defined as $u(x, y, 0) = -\sin(\pi(x + y))$, and the corresponding boundary condition was defined as $u(-1, y, t) = u(1, y, t) = u(x, -1, t) = u(x, 1, t) = 0$.

The ***in-dis-test*** **data** is simulated with with $x = [-1, 1]$, $y = [-1, 1]$ and with the time span of $t = [1, 2]$ and $N_t = 201$. Initial condition is taken from the train data at $t = 1$ and boundary conditions are also identical to the train data.

The simulation domain for the ***out-dis-test*** **data** was identical with the train data, except for the initial condition that was defined as $u(x, y, 0) = -\sin(\pi(x - y))$.

Model Architectures **TCN** is designed to have one input- and output neuron, and one hidden layer of size 32. **ConvLSTM** has one input- and output neuron and one hidden layer of size 24. The lateral and dynamic input- and output sizes of the **DISTANA** model are set to one and a hidden layer of size 16 is used. The pure ML models were trained on the first 150 time steps and validated on the remaining 50 time steps of the train data (applying early stopping). Also, to prevent the pure ML models from diverging too much in closed loop, the boundary data are fed into the models as done during teacher forcing. **PINN** is defined as a feedforward network with the size of [3, 20, 20, 20, 20, 20, 20, 20, 20, 1]. **PhyDNet** is defined with the PhyCell containing 32 input dimensions, 49 hidden dimensions, 1 hidden layer, and the ConvLSTM containing 32 input dimensions, 32 hidden dimensions, 1 hidden layer. For **FINN**, the modules $\varphi_\mathcal{N}$, $\varphi_\mathcal{D}$, $\mathcal{R}$ and $\varphi_\mathcal{A}$ were used, with $\varphi_\mathcal{A}$ defined as a feedforward network with the size of [1, 10, 20, 10, 1] that takes u as an input and outputs the advective velocity $v(u)$, and $\varphi_\mathcal{D}$ as a learnable scalar that learns the diffusion coefficient D. All models are trained until convergence using the L-BFGS optimizer, except for PhyDNet, which is trained with the Adam optimizer and a learning rate of 1×10^{-3} due to stability issues when training with the L-BFGS optimizer.

Additional Results Individual errors are reported for the ten different training runs and visualizations are generated for the *train* (Table 3), *in-dis-test* (Table 4 and Figure 8), and *out-dis-test* (Table 5 and Figure 9) datasets.

Table 3: Closed loop rMSE on the *train* data from ten different training runs for each model for the Burgers' equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	6.2×10^{-2}	3.5×10^{-1}	3.6×10^{-3}	4.7×10^{-2}	7.9×10^{-4}	9.9×10^{-5}
2	2.5×10^{-2}	4.1×10^{-2}	7.1×10^{-3}	3.1×10^{-1}	8.6×10^{-4}	8.9×10^{-5}
3	3.9×10^{-2}	5.8×10^{-2}	3.5×10^{-4}	1.7×10^{-1}	1.3×10^{-3}	2.1×10^{-4}
4	3.7×10^{-2}	1.9×10^{-2}	4.5×10^{-3}	2.4×10^{-1}	7.9×10^{-4}	3.7×10^{-5}
5	5.7×10^{-2}	2.6×10^{-2}	1.2×10^{-3}	1.9×10^{-1}	8.8×10^{-4}	5.5×10^{-5}
6	2.5×10^{-2}	5.3×10^{-1}	4.0×10^{-3}	1.2×10^{-1}	1.7×10^{-3}	4.2×10^{-5}
7	4.1×10^{-2}	1.2×10^{0}	2.7×10^{-4}	1.3×10^{-1}	1.1×10^{-3}	1.3×10^{-4}
8	2.2×10^{-2}	2.4×10^{-2}	3.9×10^{-2}	1.7×10^{-1}	1.5×10^{-3}	1.2×10^{-4}
9	4.6×10^{-2}	7.8×10^{-2}	9.2×10^{-3}	2.8×10^{-1}	6.3×10^{-4}	1.9×10^{-4}
10	3.2×10^{-2}	2.6×10^{-2}	3.9×10^{-3}	1.2×10^{-1}	9.1×10^{-4}	6.2×10^{-5}

Table 4: Closed loop rMSE on *in-dis-test* data from ten different training runs for each model for the Burgers' equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	4.1×10^{-1}	1.4×10^{1}	1.8×10^{-1}	7.0×10^{-2}	9.0×10^{-1}	1.0×10^{-2}
2	3.0×10^{-1}	2.6×10^{0}	2.4×10^{-1}	8.6×10^{-1}	2.9×10^{-1}	1.3×10^{-2}
3	2.2×10^{-1}	8.5×10^{-1}	1.5×10^{-2}	9.5×10^{-1}	3.6×10^{-1}	1.4×10^{-2}
4	2.9×10^{-1}	3.9×10^{-1}	3.5×10^{-1}	6.4×10^{-1}	1.5×10^{-1}	3.0×10^{-3}
5	3.5×10^{-1}	4.6×10^{-1}	2.9×10^{-2}	3.6×10^{-1}	1.5×10^{-1}	1.0×10^{-2}
6	3.4×10^{-1}	1.4×10^{1}	4.3×10^{-2}	6.8×10^{-1}	2.4×10^{-1}	7.5×10^{-3}
7	4.4×10^{-1}	1.9×10^{1}	1.5×10^{-2}	4.4×10^{-1}	2.0×10^{-1}	1.6×10^{-2}
8	2.2×10^{-1}	5.4×10^{-1}	1.5×10^{0}	5.3×10^{-1}	2.0×10^{-1}	1.2×10^{-2}
9	3.5×10^{-1}	4.7×10^{0}	4.6×10^{-1}	8.1×10^{-1}	1.1×10^{-2}	1.2×10^{-2}
10	1.9×10^{-1}	3.1×10^{0}	2.9×10^{-2}	4.4×10^{-1}	7.2×10^{-2}	1.1×10^{-2}

Table 5: Closed loop rMSE on *out-dis-test* data from ten different training runs for each model for the Burgers' equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	1.1×10^{-1}	3.4×10^{-1}	5.5×10^{-2}	-	3.8×10^{-1}	2.0×10^{-5}
2	8.4×10^{-2}	1.1×10^{0}	4.9×10^{-2}	-	7.0×10^{-1}	3.0×10^{-5}
3	1.2×10^{-1}	3.1×10^{-1}	3.5×10^{-3}	-	3.5×10^{-1}	2.9×10^{-5}
4	1.1×10^{-1}	1.3×10^{-1}	4.3×10^{-2}	-	3.0×10^{-1}	2.7×10^{-5}
5	1.0×10^{-1}	1.1×10^{-1}	1.3×10^{-2}	-	4.8×10^{-1}	8.4×10^{-6}
6	7.0×10^{-2}	5.8×10^{-1}	9.0×10^{-3}	-	1.9×10^{-1}	1.2×10^{-5}
7	8.2×10^{-2}	1.1×10^{0}	9.8×10^{-3}	-	4.0×10^{-1}	3.7×10^{-5}
8	4.7×10^{-2}	9.8×10^{-2}	1.3×10^{-1}	-	2.3×10^{-1}	2.5×10^{-5}
9	1.1×10^{-1}	1.8×10^{-1}	3.7×10^{-2}	-	1.6×10^{-1}	3.8×10^{-5}
10	9.5×10^{-2}	2.9×10^{-1}	1.4×10^{-2}	-	3.4×10^{-1}	1.6×10^{-5}

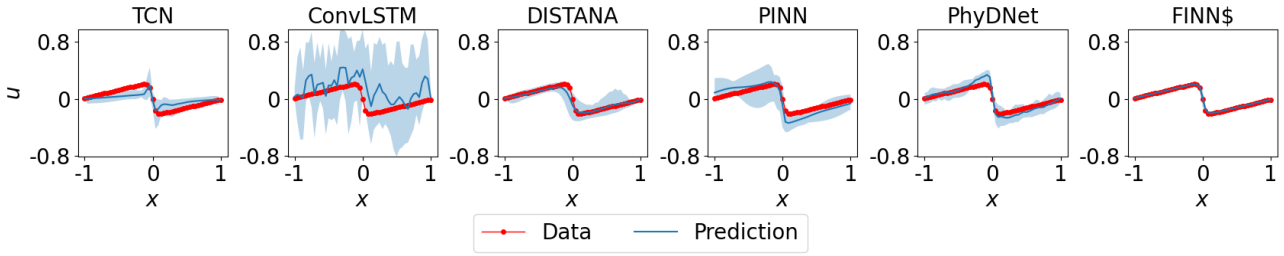


Figure 8: Prediction mean over ten different trained models (with 95% confidence interval) of the Burgers' equation at $t = 2$ for the *in-dis-test* dataset.

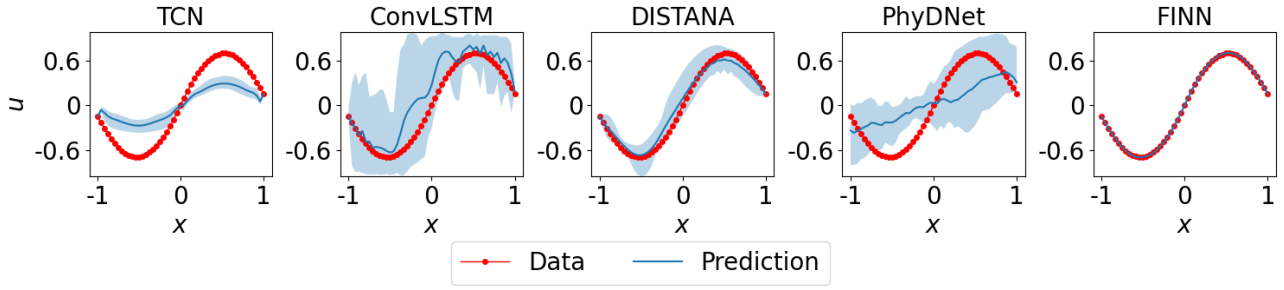


Figure 9: Prediction mean over ten different trained models (with 95% confidence interval) of the Burgers' equation at $t = 1$ for the *out-dis-test* data.

C.2. Diffusion-Sorption

The diffusion-sorption equation is another widely applied equation in fluid mechanics. A practical example of the equation is to model contaminant transport in groundwater. Its retardation factor R can be modelled using different closed parametric relations known as sorption isotherms that should be calibrated to observation data.

Data The one-dimensional diffusion-sorption equation is written as the following coupled system of equations:

$$\frac{\partial u}{\partial t} = \frac{D}{R(u)} \frac{\partial^2 u}{\partial x^2}, \quad (16) \qquad \frac{\partial u_t}{\partial t} = D\phi \frac{\partial^2 u}{\partial x^2}, \quad (17)$$

where the dissolved concentration u and total concentration u_t are the main unknowns. The effective diffusion coefficient is denoted by $D = 5 \times 10^{-4}$, the retardation factor is $R(u)$, a function of u, and the porosity is denoted by $\phi = 0.29$.

In this work, the Freundlich sorption isotherm, see details in (Nowak & Guthke, 2016), was chosen to define the retardation factor:

$$R(u) = 1 + \frac{1-\phi}{\phi} \rho_s k n_f u^{n_f - 1}, \quad (18)$$

where $\rho_s = 2\,880$ is the bulk density, $k = 3.5 \times 10^{-4}$ is the Freundlich's parameter, and $n_f = 0.874$ is the Freundlich's exponent.

The simulation domain for the ***train*** **data** is defined with $x = [0, 1]$, $t = [0, 2\,500]$ and is discretized with $N_x = 26$ spatial locations, and $N_t = 501$ simulation steps. The initial condition is defined as $u(x, 0) = 0$, and the boundary condition is defined as $u(0, t) = 1.0$ and $u(1, t) = D\frac{\partial u}{\partial x}$.

The ***in-dis-test*** **data** was simulated with $x = [0, 1]$ and with the time span of $t = [2\,500, 10\,000]$ and $N_t = 1\,501$. Initial condition is taken from the train data at $t = 2\,500$ and boundary conditions are also identical to the train data.

The simulation domain for the ***out-dis-test*** **data** was identical with the *in-dis-test* data, except for the boundary condition that was defined as $u(0, t) = 0.7$.

Model Architectures **TCN** is designed to have two input neurons, one hidden layer of size 32, and two output neurons. **ConvLSTM** has two input- and output neurons and one hidden layer with 24 neurons. The lateral and dynamic input- and output sizes of the **DISTANA** model are set to one and two, respectively, while a hidden layer of size 16 is used. The pure ML models were trained on the first 400 time steps and validated on the remaining 100 time steps of the train data (applying early stopping). Also, to prevent the pure ML models from diverging too much in closed loop, the boundary data are fed into the models as done

Table 6: Closed loop rMSE on the *train* data from ten different training runs for each model for the diffusion-sorption equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	5.8×10^{-1}	7.6×10^{-4}	3.4×10^{-4}	3.5×10^{-3}	4.9×10^{-4}	1.8×10^{-3}
2	3.3×10^{0}	3.6×10^{-1}	2.5×10^{-4}	3.7×10^{-5}	4.1×10^{-4}	2.3×10^{-4}
3	1.3×10^{0}	7.1×10^{-1}	4.3×10^{-4}	1.9×10^{-5}	5.2×10^{-4}	1.8×10^{-3}
4	3.2×10^{0}	1.6×10^{-1}	1.5×10^{-3}	3.3×10^{-3}	4.6×10^{-4}	6.6×10^{-4}
5	2.8×10^{-2}	1.4×10^{0}	1.0×10^{-3}	4.0×10^{-5}	1.2×10^{-3}	8.4×10^{-5}
6	4.3×10^{-3}	9.4×10^{-1}	6.4×10^{-4}	2.8×10^{-5}	7.1×10^{-4}	3.1×10^{-4}
7	5.1×10^{-3}	1.0×10^{0}	1.2×10^{-3}	1.4×10^{-4}	2.9×10^{-4}	2.4×10^{-4}
8	2.8×10^{-1}	2.2×10^{-3}	4.1×10^{-4}	3.3×10^{-5}	3.8×10^{-4}	1.9×10^{-3}
9	6.9×10^{0}	3.5×10^{-2}	9.5×10^{-4}	1.4×10^{-4}	3.3×10^{-4}	1.9×10^{-4}
10	1.1×10^{-2}	4.4×10^{-1}	6.2×10^{-4}	1.7×10^{-4}	8.4×10^{-4}	2.7×10^{-4}

Table 7: Closed loop rMSE on *in-dis-test* data from ten different training runs for each model for the diffusion-sorption equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	2.9×10^{-1}	1.1×10^{-1}	1.0×10^{-2}	4.4×10^{-1}	5.5×10^{-2}	4.7×10^{-3}
2	3.5×10^{0}	1.5×10^{-1}	1.3×10^{-3}	1.3×10^{-2}	1.2×10^{-1}	5.2×10^{-4}
3	7.0×10^{-1}	3.1×10^{-1}	2.5×10^{-2}	1.4×10^{-3}	4.3×10^{-3}	4.7×10^{-3}
4	3.5×10^{0}	4.2×10^{-1}	5.2×10^{-2}	7.7×10^{-2}	5.3×10^{-3}	1.6×10^{-3}
5	5.2×10^{-1}	1.2×10^{0}	1.1×10^{-1}	1.2×10^{-4}	4.2×10^{-1}	1.4×10^{-4}
6	3.5×10^{-1}	8.0×10^{-1}	4.5×10^{-3}	2.1×10^{-4}	7.1×10^{-1}	7.2×10^{-4}
7	2.0×10^{-2}	6.2×10^{-1}	9.6×10^{-2}	2.7×10^{-3}	1.5×10^{-2}	5.5×10^{-4}
8	5.7×10^{-1}	3.2×10^{-3}	1.4×10^{-2}	1.0×10^{-2}	3.6×10^{-3}	5.0×10^{-3}
9	8.3×10^{0}	2.0×10^{-1}	2.7×10^{-2}	6.6×10^{-2}	3.6×10^{-3}	4.1×10^{-4}
10	5.0×10^{-1}	6.2×10^{-1}	1.5×10^{-2}	5.1×10^{-3}	4.1×10^{-3}	6.1×10^{-4}

Table 8: Closed loop rMSE on *out-dis-test* data from ten different training runs for each model for the diffusion-sorption equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	1.4×10^{0}	6.8×10^{-1}	8.7×10^{-2}	-	3.9×10^{-1}	2.9×10^{-3}
2	6.4×10^{0}	1.3×10^{0}	4.8×10^{-2}	-	3.2×10^{-1}	4.4×10^{-4}
3	2.8×10^{0}	1.6×10^{0}	1.0×10^{-1}	-	4.1×10^{-1}	2.9×10^{-3}
4	6.0×10^{0}	5.8×10^{-1}	6.0×10^{-2}	-	3.3×10^{-1}	1.0×10^{-3}
5	1.3×10^{0}	4.0×10^{0}	1.4×10^{-2}	-	6.4×10^{-1}	1.5×10^{-4}
6	1.2×10^{-2}	3.0×10^{0}	2.6×10^{-1}	-	1.3×10^{0}	5.6×10^{-4}
7	2.8×10^{-2}	2.8×10^{0}	4.9×10^{-1}	-	4.5×10^{-1}	4.5×10^{-4}
8	1.5×10^{-1}	7.9×10^{-2}	2.9×10^{-2}	-	3.7×10^{-1}	3.1×10^{-3}
9	1.4×10^{1}	1.0×10^{0}	2.2×10^{-1}	-	4.0×10^{-1}	3.7×10^{-4}
10	1.2×10^{0}	2.4×10^{0}	7.8×10^{-2}	-	3.9×10^{-1}	4.9×10^{-4}

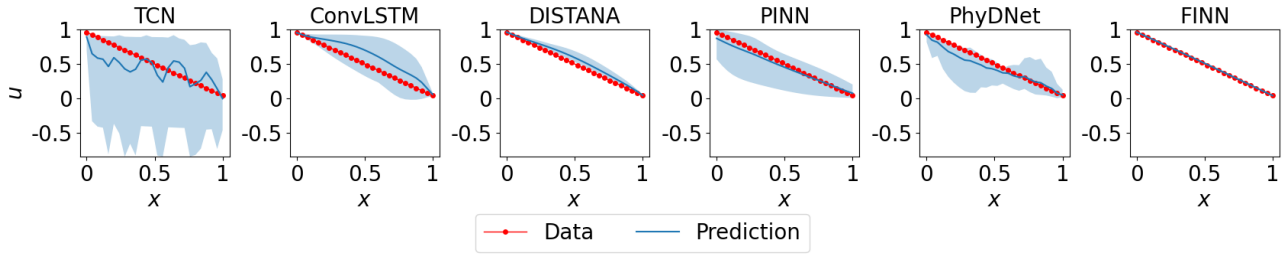


Figure 10: Prediction mean over ten different trained models (with 95% confidence interval) of the dissolved concentration in the diffusion-sorption equation at $t = 10\,000$ for the *in-dis-test* dataset.

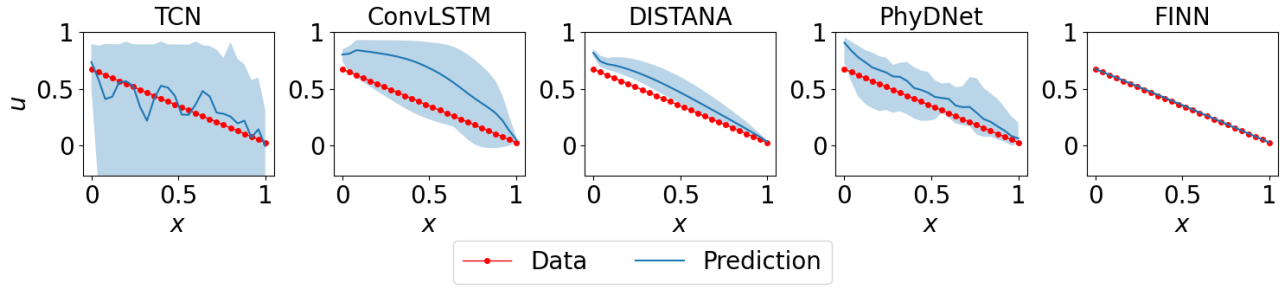


Figure 11: Prediction mean over ten different trained models (with 95% confidence interval) of the dissolved concentration in the diffusion-sorption equation at $t = 10\,000$ for the *out-dis-test* dataset.

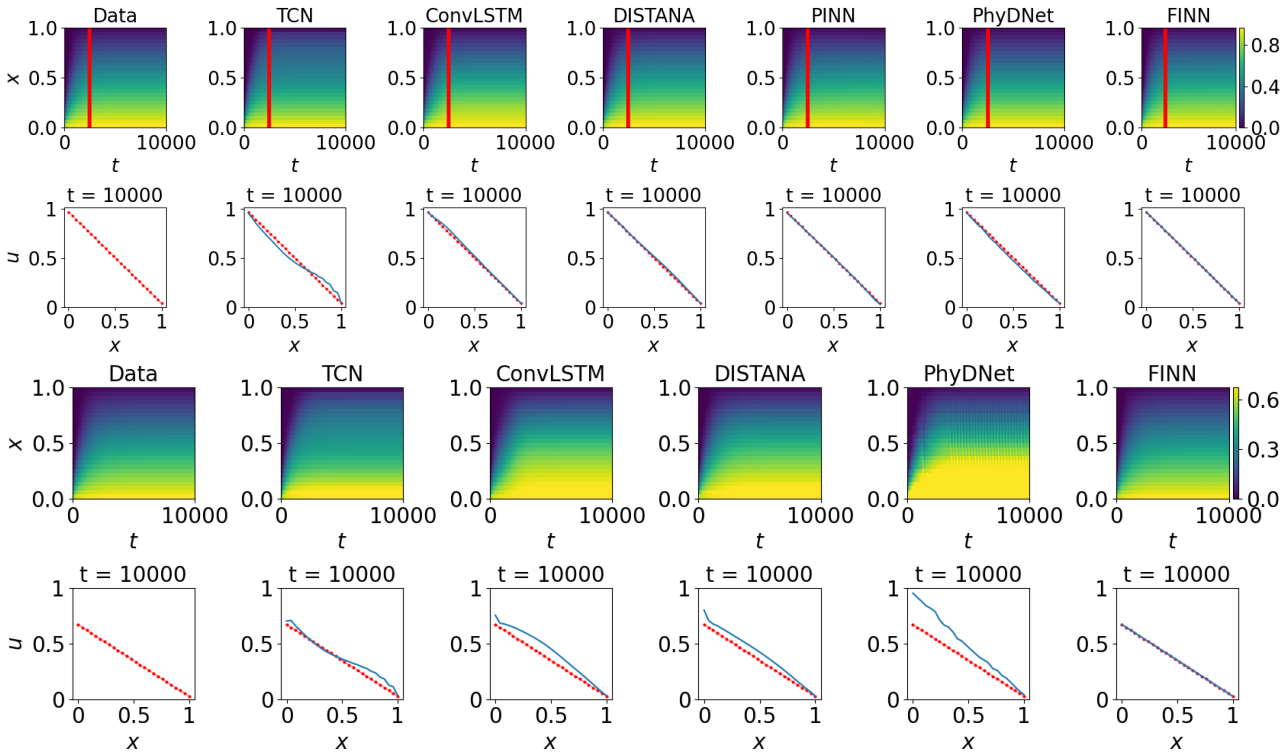


Figure 12: Plots of diffusion-sorption's dissolved concentration (red) and prediction (blue) using different models. The first and third rows show the solution over x and t (vertical red lines mark the transition from *train* to *in-dis-test*). Second (*in-dis-test*) and fourth (*out-dis-test*) rows visualize the solution distributed in x at $t = 10\,000$.

during teacher forcing. **PINN** was defined as a feedforward network with the size of [2, 20, 20, 20, 20, 20, 20, 20, 20, 2]. **PhyDNet** was defined with the PhyCell containing 32 input dimensions, 7 hidden dimensions, 1 hidden layer, and the ConvLSTM containing 32 input dimensions, 32 hidden dimensions, 1 hidden layer. For **FINN**, the modules $\varphi_{\mathcal{N}}$ and $\varphi_{\mathcal{D}}$ were used, with $\varphi_{\mathcal{D}}$ defined as a feedforward network with the size of [1, 10, 20, 10, 1] that takes u as an input and outputs the retardation factor $R(u)$. All models are trained until convergence using the L-BFGS optimizer, except for PhyDNet, which is trained with the Adam optimizer and a learning rate of 1×10^{-3} due to stability issues when training with the L-BFGS optimizer.

Additional Results Individual errors are reported for the ten different training runs and visualizations are generated for the *train* (Table 6), *in-dis-test* (Table 7 and Figure 10), and *out-dis-test* (Table 8 and Figure 11) datasets. Results for the total concentration u_t were omitted due to high similarity to the concentration of the reported contamination solution u.

C.3. Diffusion-Reaction

The diffusion-reaction equation is applicable in physical and biological systems, for example in pattern formation (Turing, 1952).

Data In the current paper, we consider the two-dimensional diffusion-reaction for that class of problems:

$$\frac{\partial u_1}{\partial t} = R_1(u_1, u_2) + D_1 \left(\frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2} \right), \tag{19}$$

$$\frac{\partial u_2}{\partial t} = R_2(u_1, u_2) + D_2 \left(\frac{\partial^2 u_2}{\partial x^2} + \frac{\partial^2 u_2}{\partial y^2} \right). \tag{20}$$

Here, $D_1 = 10^{-3}$ and $D_2 = 5 \times 10^{-3}$ are the diffusion coefficient for the activator and inhibitor, respectively. The system of equations is coupled through the reaction terms $R_1(u_1, u_2)$ and $R_2(u_1, u_2)$ which are both dependent on u_1 and u_2. In this work, the Fitzhugh–Nagumo (Klaasen & Troy, 1984) system was considered to define the reaction function:

$$R_1(u_1, u_2) = u_1 - u_1^3 - k - u_2, \tag{21}$$

$$R_2(u_1, u_2) = u_1 - u_2, \tag{22}$$

with $k = 5 \times 10^{-3}$.

The simulation domain for the ***train*** **data** is defined with $x = [-1, 1]$, $y = [-1, 1]$, $t = [0, 10]$ and is discretized with $N_x = 49$ and $N_y = 49$ spatial locations, and $N_t = 101$ simulation steps. The initial condition was defined as $u_1(x, 0) = u_2(x, 0) = \sin(\pi(x + 1)/2) \sin(\pi(y + 1)/2)$, and the corresponding boundary condition was defined as $\frac{\partial u_1}{\partial x}(-1, y, t) = 0$, $\frac{\partial u_1}{\partial x}(1, y, t) = 0$, $\frac{\partial u_2}{\partial x}(-1, y, t) = 0$, $\frac{\partial u_2}{\partial x}(1, y, t) = 0$, $\frac{\partial u_1}{\partial y}(x, -1, t) = 0$, $\frac{\partial u_1}{\partial y}(x, 1, t) = 0$, $\frac{\partial u_2}{\partial y}(x, -1, t) = 0$, and $\frac{\partial u_2}{\partial y}(x, 1, t) = 0$.

The ***in-dis-test*** **data** is simulated with with $x = [-1, 1]$, $y = [-1, 1]$ and with the time span of $t = [10, 50]$ and $N_t = 401$. Initial condition is taken from the train data at $t = 10$ and boundary conditions are also identical to the train data.

The simulation domain for the ***out-dis-test*** **data** was identical with the train data, except for the initial condition that was defined as $u_1(x, 0) = u_2(x, 0) = \sin(\pi(x + 1)/2) \sin(\pi(y + 1)/2) - 0.5$, i.e. subtracting 0.5 from the original initial condition.

Model Architectures **TCN** is designed to have two input- and output neurons, and one hidden layer of size 32. **ConvLSTM** has two input- and output neurons and one hidden layer of size 24. The lateral and dynamic input- and output sizes of the **DISTANA** model are set to one and two, respectively, while a hidden layer of size 16 is used. The pure ML models were trained on the first 70 time steps and validated on the remaining 30 time steps of the train data (applying early stopping). Also, to prevent the pure ML models from diverging too much in closed loop, the boundary data are fed into the models as done during teacher forcing. **PINN** is defined as a feedforward network with the size of [3, 20, 20, 20, 20, 20, 20, 20, 20, 2]. **PhyDNet** is defined with the PhyCell containing 32 input dimensions, 49 hidden dimensions, 1 hidden layer, and the ConvLSTM containing 32 input dimensions, 32 hidden dimensions, 1 hidden layer. For **FINN**, the modules $\varphi_{\mathcal{N}}$, $\varphi_{\mathcal{D}}$ and Φ are used, with $\varphi_{\mathcal{D}}$ set as two learnable scalars that learn the diffusion coefficients D_1 and D_2, and Φ defined as a feedforward network with the size of [2, 20, 20, 20, 2] that takes u_1 and u_2 as inputs and outputs the reaction functions $R_1(u_1, u_2)$ and $R_2(u_1, u_2)$. All models are trained until convergence using the L-BFGS optimizer, except for PhyDNet, which is trained with the Adam optimizer and a learning rate of 1×10^{-3} due to stability issues when training with the L-BFGS optimizer.

Additional Results Individual errors are reported for the ten different training runs and visualizations are generated for the *train* (Table 9), *in-dis-test* (Table 10 and Figure 13), and *out-dis-test* (Table 11 and Figure 14) datasets. Results for the total inhibitor u_2 were omitted due to high similarity to the reported activator u_1.

C.4. Allen–Cahn

The Allen–Cahn equation is commonly employed in reaction-diffusion systems, e.g. to model phase separation in multi-component alloy systems (Raissi et al., 2019).

Table 9: Closed loop rMSE on *train* data from ten different training runs for each model for the diffusion-reaction equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	1.2×10^{-1}	2.3×10^{-2}	2.3×10^{-2}	3.7×10^{-3}	1.1×10^{-3}	1.3×10^{-3}
2	5.9×10^{-2}	2.2×10^{-2}	5.6×10^{-2}	7.4×10^{-3}	9.5×10^{-4}	2.1×10^{-3}
3	4.3×10^{-1}	1.9×10^{-2}	1.2×10^{-1}	2.8×10^{-3}	7.7×10^{-4}	1.8×10^{-3}
4	1.2×10^{-1}	1.0×10^{-2}	8.7×10^{-3}	3.7×10^{-3}	9.1×10^{-4}	1.6×10^{-3}
5	3.3×10^{-1}	1.9×10^{-2}	6.2×10^{-3}	3.8×10^{-3}	1.0×10^{-3}	1.4×10^{-3}
6	1.1×10^{-1}	6.4×10^{-3}	2.9×10^{-3}	3.3×10^{-3}	8.7×10^{-4}	2.3×10^{-3}
7	3.2×10^{-1}	5.0×10^{-2}	1.1×10^{-1}	68.1×10^{-4}	1.1×10^{-3}	2.2×10^{-3}
8	1.5×10^{-1}	1.4×10^{-2}	3.2×10^{-2}	1.3×10^{-3}	1.0×10^{-3}	1.3×10^{-3}
9	1.8×10^{-1}	2.9×10^{-2}	5.0×10^{-2}	4.2×10^{-4}	1.2×10^{-3}	2.0×10^{-3}
10	7.4×10^{-2}	9.7×10^{-1}	1.2×10^{-1}	8.3×10^{-3}	9.9×10^{-4}	1.5×10^{-3}

Table 10: Closed loop rMSE on *in-dis-test* data from ten different training runs for each model for the diffusion-reaction equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	1.4×10^{0}	6.4×10^{-1}	1.3×10^{0}	6.1×10^{-1}	4.9×10^{-1}	2.3×10^{-2}
2	4.7×10^{0}	4.6×10^{-1}	1.5×10^{0}	1.8×10^{0}	7.0×10^{-1}	1.3×10^{-2}
3	5.2×10^{0}	7.4×10^{-1}	2.4×10^{0}	2.8×10^{-1}	6.2×10^{-1}	1.4×10^{-2}
4	2.4×10^{0}	5.3×10^{-1}	1.6×10^{0}	2.1×10^{-1}	5.4×10^{-1}	2.3×10^{-2}
5	5.8×10^{0}	3.9×10^{-1}	1.5×10^{0}	1.0×10^{0}	4.2×10^{-1}	1.3×10^{-2}
6	1.5×10^{0}	4.3×10^{-1}	2.2×10^{-1}	7.1×10^{-1}	7.9×10^{-1}	1.6×10^{-2}
7	4.8×10^{0}	7.9×10^{-1}	1.7×10^{0}	4.3×10^{-1}	5.3×10^{-1}	1.2×10^{-2}
8	5.7×10^{0}	6.9×10^{-1}	1.5×10^{0}	1.0×10^{-1}	7.3×10^{-1}	1.9×10^{-2}
9	4.4×10^{0}	1.8×10^{0}	1.3×10^{0}	1.8×10^{-1}	5.3×10^{-1}	1.5×10^{-2}
10	1.7×10^{0}	9.4×10^{-1}	1.3×10^{0}	2.1×10^{-1}	8.9×10^{-1}	1.7×10^{-2}

Table 11: Closed loop rMSE on *out-dis-test* data from ten different training runs for each model for the diffusion-reaction equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	6.7×10^{-1}	9.2×10^{-2}	2.7×10^{-1}	-	1.1×10^{0}	1.9×10^{-1}
2	4.9×10^{0}	3.7×10^{-1}	3.9×10^{-1}	-	1.6×10^{0}	1.6×10^{-1}
3	7.4×10^{0}	3.8×10^{-1}	6.1×10^{-1}	-	7.2×10^{-1}	1.7×10^{-1}
4	2.1×10^{0}	2.1×10^{-1}	2.3×10^{-1}	-	1.2×10^{0}	1.9×10^{-1}
5	7.4×10^{0}	1.7×10^{-1}	7.5×10^{-2}	-	8.0×10^{-1}	1.8×10^{-1}
6	1.9×10^{0}	4.4×10^{-1}	6.0×10^{-2}	-	8.3×10^{-1}	1.8×10^{-1}
7	6.5×10^{0}	3.0×10^{-1}	5.6×10^{-1}	-	8.3×10^{-1}	1.7×10^{-1}
8	5.7×10^{0}	1.1×10^{-1}	3.4×10^{-1}	-	6.2×10^{-1}	1.8×10^{-1}
9	6.6×10^{0}	1.2×10^{0}	3.7×10^{-1}	-	6.0×10^{-1}	1.7×10^{-1}
10	1.1×10^{0}	1.1×10^{0}	9.7×10^{-1}	-	1.7×10^{0}	1.8×10^{-1}

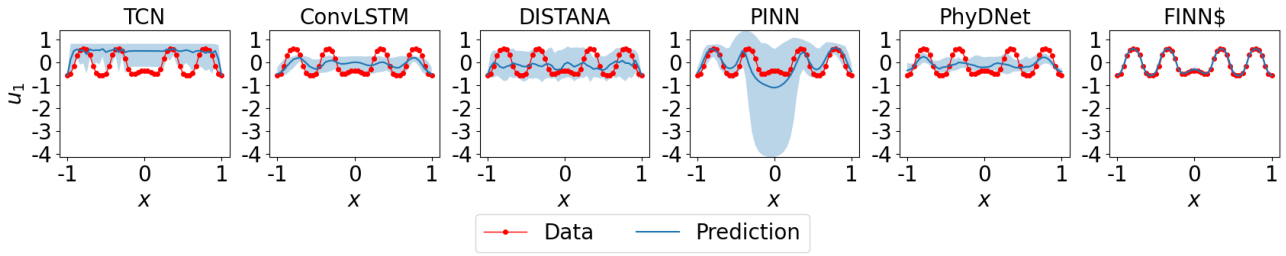


Figure 13: Prediction mean over ten different trained models (with 95% confidence interval) of the activator in the diffusion-reaction equation at $t = 50$ for the *in-dis-test* dataset.

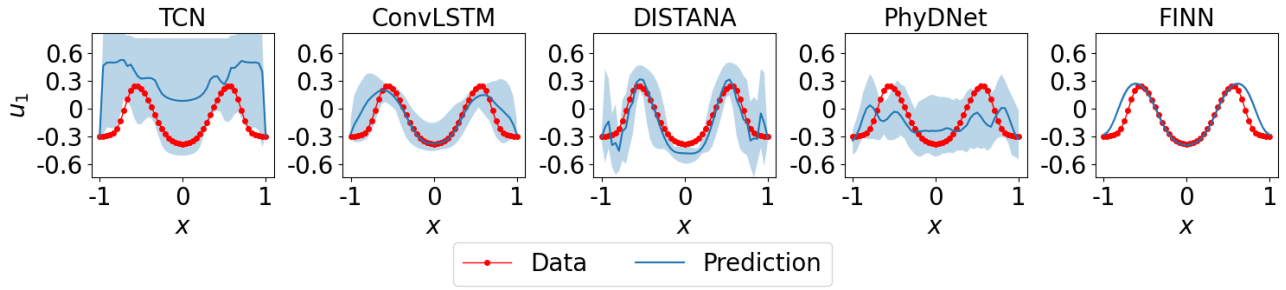


Figure 14: Prediction mean over ten different trained models (with 95% confidence interval) of the activator in the diffusion-reaction equation at $t = 10$ for the *out-dis-test* dataset.

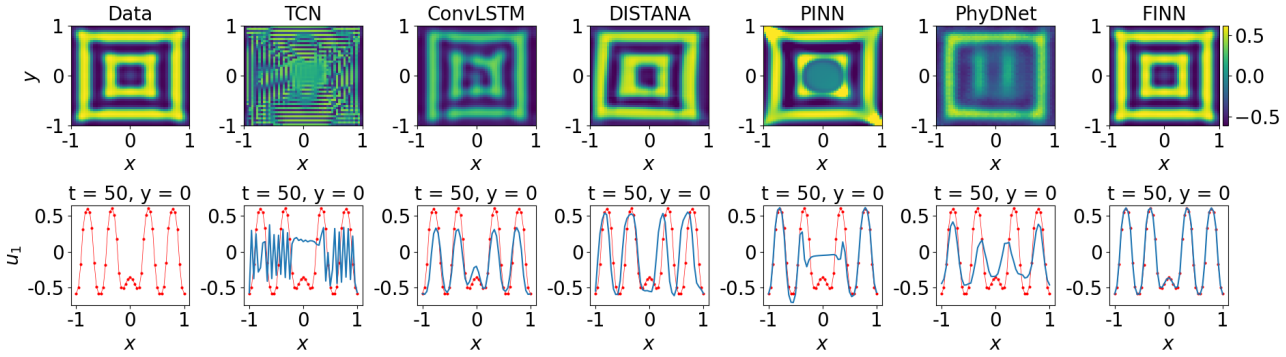


Figure 15: Plots of diffusion-reaction's activator data u_1 (red) and extrapolated prediction (blue) using different models. The plots in the first row show the solution distributed over x and y at $t = 50$, and the plots in the second row show the solution distributed in x at $y = 0$ and $t = 50$.

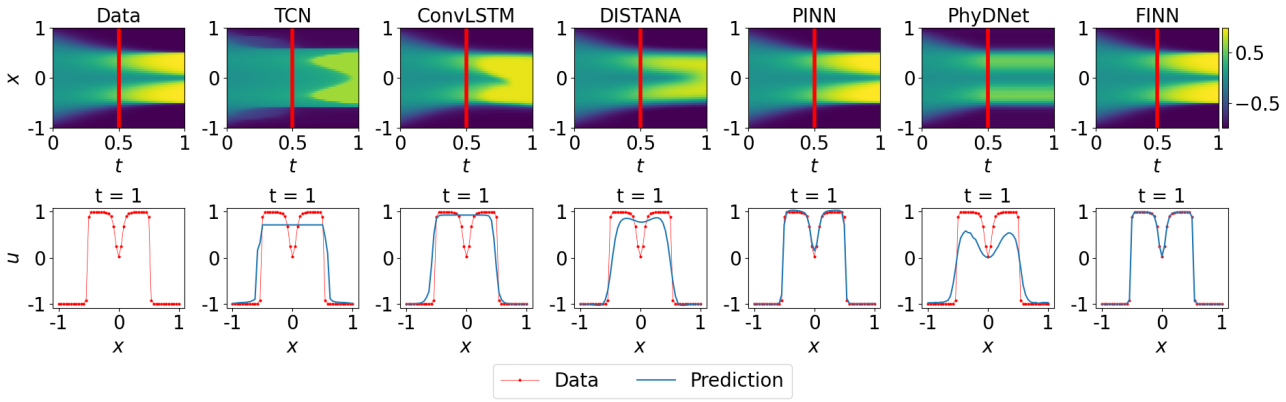


Figure 16: Plots of Allen–Cahn's data and prediction of *in-dis-test* data using different models. The plots in the first row show the solution over x and t (the red lines mark the transition from *train* to *in-dis-test*), the second row visualizes the best model's solution distributed in x at $t = 1$.

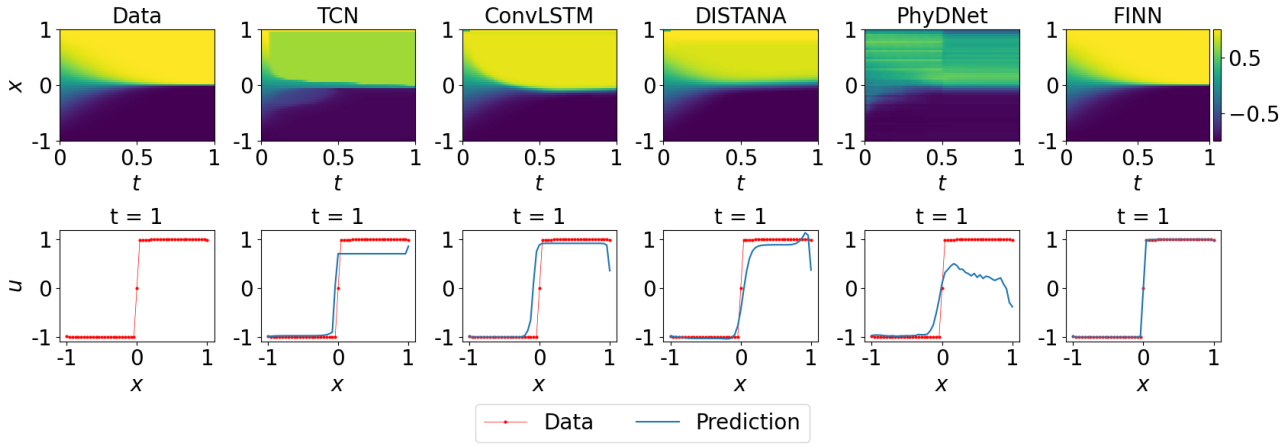


Figure 17: Plots of Allen–Cahn's data and prediction of *out-dis-test* data using different models. The plots in the first row show the solution over x and t, the second row visualizes the solution distributed in x at $t = 1$.

Data The one-dimensional Allen–Cahn equation is written as

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + R(u), \tag{23}$$

where the main unknown is u, the reaction term is denoted as $R(u)$ which is a function of u and the diffusion coefficient is $D = 10^{-4}$. In the current work, the reaction term is defined as:

$$R(u) = 5u - 5u^3, \tag{24}$$

to reproduce the experiment conducted in the PINN paper (Raissi et al., 2019). The simulation domain for the ***train* data** is defined with $x = [-1, 1]$, $t = [0, 0.5]$ and is discretized with $N_x = 49$ spatial locations, and $N_t = 201$ simulation steps. The initial condition is defined as $u(x, 0) = x^2 \cos(\pi x)$, and periodic boundary condition is used, i.e. $u(-1, t) = u(1, t)$.

***In-dis-test* data** is simulated with $x = [-1, 1]$, a time span of $t = [0.5, 1]$ and $N_t = 201$. Initial condition is taken from the train data at $t = 0.5$ and boundary conditions are also similar to the train data.

The simulation domain for the ***out-dis-test* data** is identical with the train data, except for the initial condition that is defined as $u(x, 0) = \sin(\pi x/2)$.

Model Architectures The **TCN** and **ConvLSTM** are designed to have one input neuron, one hidden layer of size 32 and 24, respectively, and one output neuron. The lateral and dynamic input- and output sizes of the **DISTANA** model are set to one and a hidden layer of size 16 is used. The pure ML models were trained on the first 150 time steps and validated on the remaining 50 time steps of the train data (applying early stopping). Also, to prevent the pure ML models from diverging too much in closed loop, the boundary data are fed into the models as done dur-

Table 12: Closed loop rMSE on the *train* data from ten different training runs for each model for the Allen–Cahn equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	5.0×10^{-2}	3.2×10^{-1}	1.3×10^{-3}	7.5×10^{-5}	6.6×10^{-4}	8.0×10^{-5}
2	2.0×10^{-1}	1.4×10^{-1}	2.1×10^{-3}	1.9×10^{-4}	2.9×10^{-4}	3.7×10^{-5}
3	1.3×10^{-1}	5.5×10^{-2}	3.6×10^{-3}	1.9×10^{-4}	1.1×10^{-3}	2.5×10^{-5}
4	4.0×10^{-1}	3.0×10^{-2}	2.2×10^{-3}	7.1×10^{-6}	2.2×10^{-4}	2.1×10^{-6}
5	6.2×10^{-2}	3.6×10^{-1}	1.2×10^{-3}	6.0×10^{-5}	2.3×10^{-4}	1.1×10^{-4}
6	4.8×10^{-2}	2.1×10^{-2}	2.6×10^{-3}	2.9×10^{-5}	2.2×10^{-4}	1.1×10^{-5}
7	4.4×10^{-2}	3.2×10^{-2}	1.9×10^{-3}	6.4×10^{-5}	7.9×10^{-4}	5.2×10^{-5}
8	1.7×10^{-1}	2.4×10^{-3}	3.9×10^{-3}	9.5×10^{-5}	1.7×10^{-4}	1.3×10^{-6}
9	3.1×10^{0}	1.8×10^{0}	3.5×10^{-3}	9.0×10^{-5}	4.5×10^{-4}	1.4×10^{-6}
10	1.3×10^{-1}	8.1×10^{-4}	6.8×10^{-4}	1.7×10^{-4}	4.5×10^{-4}	1.3×10^{-6}

Table 13: Closed loop rMSE on *in-dis-test* data from ten different training runs for each model for the Allen–Cahn equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	1.5×10^{-1}	1.7×10^{0}	7.0×10^{-2}	1.0×10^{-2}	2.0×10^{-1}	8.2×10^{-5}
2	4.2×10^{-1}	1.3×10^{0}	1.3×10^{-1}	2.7×10^{-1}	1.2×10^{-1}	3.6×10^{-5}
3	3.4×10^{-1}	5.3×10^{-1}	1.2×10^{-1}	6.2×10^{-1}	1.5×10^{-1}	2.3×10^{-5}
4	6.8×10^{-1}	4.9×10^{-1}	1.1×10^{-1}	6.7×10^{-3}	1.4×10^{-1}	3.0×10^{-6}
5	2.8×10^{-1}	8.0×10^{-1}	1.1×10^{-1}	3.3×10^{-3}	1.2×10^{-1}	1.4×10^{-4}
6	1.8×10^{-1}	4.6×10^{-1}	1.5×10^{-1}	3.0×10^{-3}	1.8×10^{-1}	1.6×10^{-5}
7	1.8×10^{-1}	7.8×10^{-1}	1.8×10^{-1}	7.3×10^{-4}	3.3×10^{-1}	4.9×10^{-5}
8	4.4×10^{-1}	8.2×10^{-1}	1.4×10^{-1}	1.9×10^{-1}	1.5×10^{-1}	1.9×10^{-6}
9	2.1×10^{0}	1.9×10^{0}	2.4×10^{-1}	1.8×10^{-2}	1.5×10^{-1}	1.9×10^{-6}
10	3.4×10^{-1}	8.0×10^{-2}	6.8×10^{-2}	4.8×10^{-2}	1.4×10^{-1}	1.9×10^{-6}

Table 14: Closed loop rMSE on *out-dis-test* data from ten different training runs for each model for the Allen–Cahn equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	4.8×10^{-2}	6.1×10^{-2}	1.6×10^{-2}	-	7.2×10^{-1}	8.6×10^{-5}
2	2.3×10^{-1}	4.4×10^{-1}	4.3×10^{-2}	-	2.8×10^{-1}	3.4×10^{-5}
3	1.5×10^{-1}	1.7×10^{-1}	7.3×10^{-2}	-	7.3×10^{-1}	2.1×10^{-5}
4	2.3×10^{-1}	3.4×10^{-1}	1.2×10^{-2}	-	8.5×10^{-1}	3.5×10^{-6}
5	2.0×10^{-1}	9.7×10^{-2}	4.7×10^{-2}	-	7.9×10^{-1}	1.5×10^{-4}
6	1.0×10^{-1}	3.4×10^{-1}	3.6×10^{-2}	-	9.1×10^{-1}	1.7×10^{-5}
7	1.2×10^{-1}	1.8×10^{-1}	6.9×10^{-2}	-	1.2×10^{0}	5.0×10^{-5}
8	1.7×10^{-1}	6.2×10^{-1}	1.1×10^{-1}	-	6.3×10^{-1}	2.3×10^{-6}
9	1.4×10^{0}	1.6×10^{0}	1.2×10^{-1}	-	7.2×10^{-1}	2.1×10^{-6}
10	1.1×10^{-1}	8.8×10^{-2}	1.8×10^{-2}	-	6.9×10^{-1}	2.2×10^{-6}

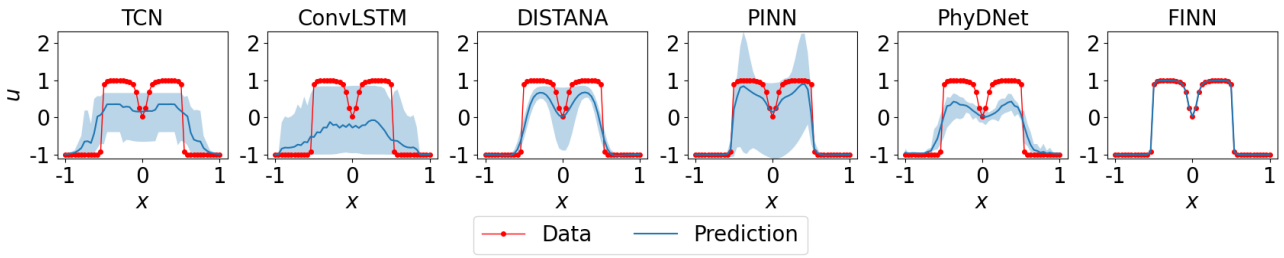


Figure 18: Prediction mean over ten different trained models (with 95% confidence interval) of the Allen–Cahn equation at $t = 1$ for the *in-dis-test* dataset.

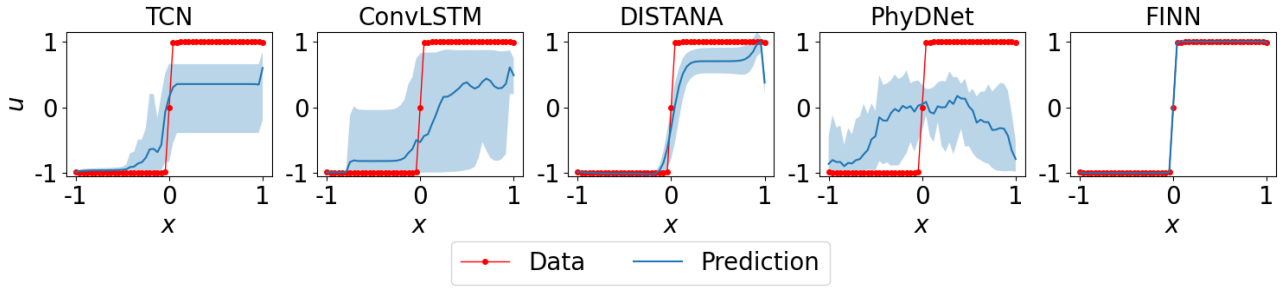


Figure 19: Prediction mean over ten different trained models (with 95% confidence interval) of the Allen–Cahn equation at $t = 1$ for the *out-dis-test* data.

ing teacher forcing. **PINN** was defined as a feedforward network with the size of [2, 20, 20, 20, 20, 20, 20, 20, 20, 1] (8 hidden layers, each contains 20 hidden neurons). **PhyDNet** was defined with the PhyCell containing 32 input dimensions, 7 hidden dimensions, 1 hidden layer, and the ConvLSTM containing 32 input dimensions, 32 hidden dimensions, 1 hidden layer.

For **FINN**, the modules $\varphi_{\mathcal{N}}$, $\varphi_{\mathcal{D}}$, and Φ were used, with $\varphi_{\mathcal{D}}$ defined as a learnable scalar that learns the diffusion coefficient D, and Φ defined as a feedforward network with the size of [1, 10, 20, 10, 1] that takes u as an input and outputs the reaction function $R(u)$. All models are trained until convergence using the L-BFGS optimizer, except for PhyDNet, which is trained with the Adam optimizer and a learning rate of 1×10^{-3} due to stability issues when training with the L-BFGS optimizer.

Additional Results Individual errors are reported for the ten different training runs and visualizations are generated for the *train* (Table 12), *in-dis-test* (Table 13, Figure 16 and Figure 18), and *out-dis-test* (Table 14, Figure 17 and Figure 19) datasets.

C.5. Soil Parameters and Simulation Domains for the Experimental Dataset

Identical to Praditia et al. (2021), the soil parameters and simulation (and experimental) domain used in the real-world diffusion-sorption experiment are summarized in Table 15 for core samples #1, #2, and #2B.

Table 15: Soil and experimental parameters of core samples #1, #2, and #2B. D is the diffusion coefficient, ϕ is the porosity, ρ_s is the bulk density, L and r are the length and radius of the sample, t_{end} is the simulation time, Q is the flow rate in the bottom reservoir and u_s is the total concentration of trichloroethylene in the sample.

Soil parameters				
Param.	Unit	Core #1	Core #2	Core #2B
D	m^2/day	$2.00 \cdot 10^{-5}$	$2.00 \cdot 10^{-5}$	$2.78 \cdot 10^{-5}$
ϕ	-	0.288	0.288	0.288
ρ_s	kg/m^3	1957	1957	1957
Simulation domain				
Param.	Unit	Core #1	Core #2	Core #2B
L	m	0.0254	0.02604	0.105
r	m	0.02375	0.02375	N/A
t_{end}	days	38.81	39.82	48.88
Q	m^3/day	$1.01 \cdot 10^{-4}$	$1.04 \cdot 10^{-4}$	N/A
u_s	kg/m^3	1.4	1.6	1.4

For all experiments, the core samples are subjected to a constant contaminant concentration at the top u_s, which can be treated as a Dirichlet boundary condition numerically. Notice that, for core sample #2, we set u_s to be

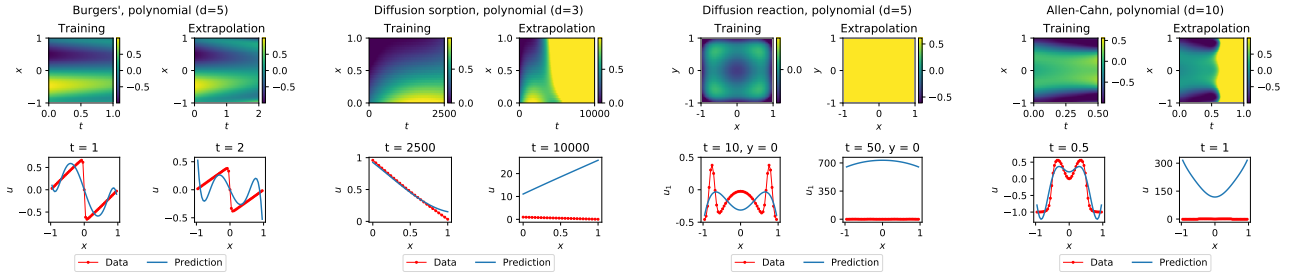


Figure 20: Prediction pairs for *train* and *in-dis-test* (left and right columns of the pairs) of the Burgers' (left, polynomial order 5), diffusion-sorption (second left, polynomial order 3), diffusion-reaction (second right, polynomial order 5 since higher orders did not converge), and Allen–Cahn (right, polynomial order 10) equations using polynomial regression.

slightly higher to compensate for the fact that there might be fractures at the top of core sample #2, so that the contaminant can break through the core sample faster.

For core samples #1 and #2, Q is the flow rate of clean water at the bottom reservoir that determines the Cauchy boundary condition at the bottom of the core samples. For core sample #2B, note that the sample length is significantly longer than the other samples, and by the end of the experiment, no contaminant has broken through the core sample. Therefore, we assume the bottom boundary condition to be a no-flow Neumann boundary condition; see (Praditia et al., 2021) for details.

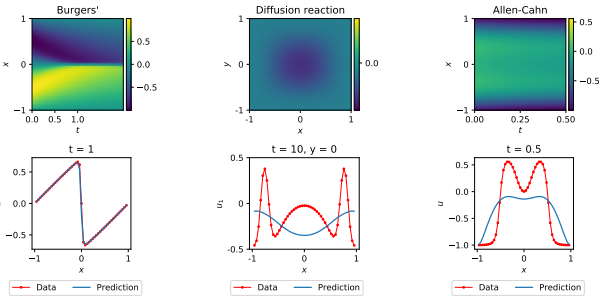
D. Ablations

We conduct the ablations with a one-dimensional instead of the two-dimensional domain for Burgers' equation. The diffusion coefficient is $D = 0.01/\pi$, and the simulation domain for the ***train* data** is defined with $x = [-1, 1]$, $t = [0, 1]$ and is discretized with $N_x = 49$ spatial locations, and $N_t = 201$ simulation steps. The initial condition is defined as $u(x, 0) = -\sin(\pi x)$, and the boundary condition is defined as $u(-1, t) = u(1, t) = 0$. ***In-dis-test* data** is simulated with $x = [-1, 1]$ and a time span of $t = [1, 2]$ and $N_t = 201$. Initial condition is taken from the *train* data at $t = 1$ and boundary conditions are also similar to the *train* data. The simulation domain for the ***out-dis-test* data** is identical with the *train* data, except for the initial condition that is defined as $u(x, 0) = \sin(\pi x)$.

D.1. Baseline Assessment with Polynomial Regression

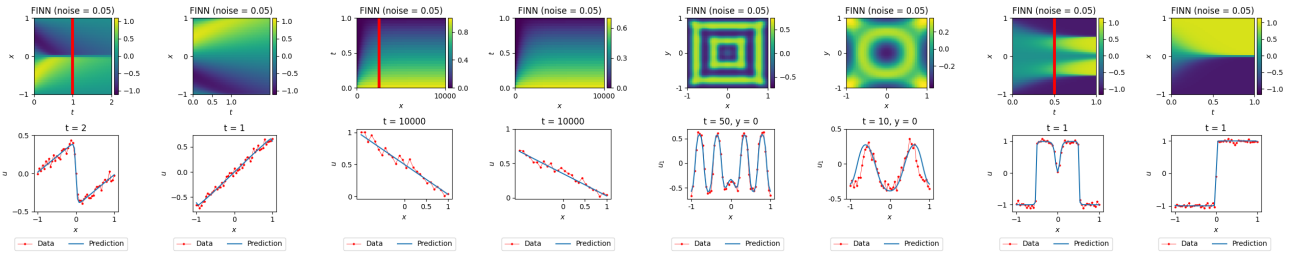
In this section, we apply polynomial regression to show that the example problems chosen in this work (i.e. Burgers', diffusion-sorption, diffusion-reaction, and Allen–Cahn) are not easy to solve. First, we use polynomial regression to fit the unknown variable $u = f(x, t)$, similar to PINN. Figure 20 shows the prediction of u for each example, obtained using the fitted polynomial coefficients. For the Burgers' equation, the *train* and *in-dis-test* predictions have rMSE values of 1.4×10^{-1} and 3.3×10^{-1}, respectively. For the diffusion-sorption equation, the *train* and *in-dis-test* predictions have rMSE values of 3.4×10^{-2} and 4.1×10^{1}, respectively. For the diffusion-reaction equation, the *train* and *in-dis-test* predictions have rMSE values of 2.2×10^{-1} and 7.1×10^{2}, respectively. For the Allen–Cahn equation, the *train* and *in-dis-test* predictions have rMSE values of 4.0×10^{-2} and 2.6×10^{5}, respectively. The simple polynomial fitting fails to obtain accurate predictions of the solution for all example problems. The results also show that the polynomials overfit the data, evidenced by the significant deterioration of performance during extrapolation (prediction of *in-dis-test* data). The diffusion-reaction and the Allen–Cahn equations are particularly the most difficult to fit, because they require higher order polynomials to obtain reasonable accuracy. With the high order, they still fail to even fit the *train* data well.

Next, we also consider using polynomial fitting in lieu of ANNs (namely the modules $\varphi_{\mathcal{A}}$, $\varphi_{\mathcal{D}}$, and Φ) in FINN. In-



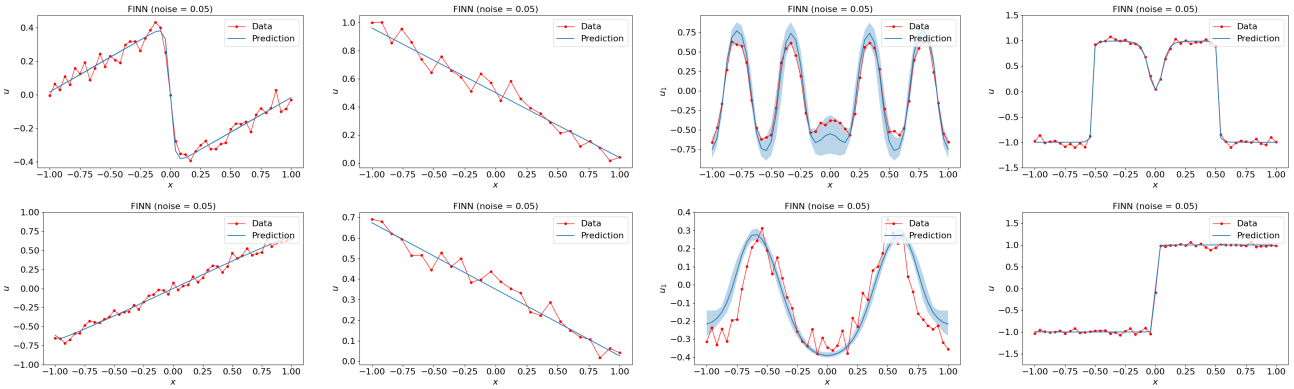
(a) (b) (c)

Figure 21: Plots of the *train* prediction in the Burgers' (left), diffusion-reaction (center), and Allen–Cahn equations (right) using FINN with polynomial fitting. Due to instability issues, the diffusion-sorption equation could not be solved with the polynomial FINN. The plots in the first row show the solution over x and t, and the plots in the second row show the solution distributed in x.



(a) Burgers' (b) Diffusion-sorption (c) Diffusion-reaction (d) Allen–Cahn

Figure 22: Paired plots of the *in-dis-test* and *out-dis-test* prediction (left and right of the pairs, respectively) after training FINN with noisy data. The plots in the first row of the pairs show the solution over x and t (red line marks the transition from *train* to *in-dis-test*), and the plots in the second row of the pairs show the best model's solution distributed in x.



(a) Burgers' (b) Diffusion-sorption (c) Diffusion-reaction (d) Allen–Cahn

Figure 23: Prediction mean over ten different trained FINN (with 95% confidence interval) of the (from left to right) one-dimensional Burgers', diffusion-sorption, diffusion-reaction, and Allen–Cahn equations obtained by training FINN with noisy data for the *in-dis-test* and *out-dis-test* (top and bottom, accordingly) prediction.

deed, with this method, the model successfully predicts Burgers' equation (Figure 21a). The rMSE values are 5.4×10^{-4}, 1.9×10^{-2}, and 3.6×10^{-4} for *train*, *in-dis-test*, and *out-dis-test* data, respectively. However, the model fails to complete the training for the diffusion-sorption equation due to major instabilities (the polynomials can produce negative outputs, leading to negative diffusion coefficients that cause numerical instability). Moreover, the model also fails to sufficiently learn the diffusion-reaction (Figure 21b) and the Allen–Cahn (Figure 21c) equations. For the diffusion-reaction equation, the rMSE values are 3.3×10^{-1}, 1.6×10^{0}, and 1.4×10^{0} for *train*, *in-dis-test*, and *out-dis-test* data, respectively. For the Allen–Cahn equation, the rMSE values are 2.6×10^{-1}, 6.6×10^{-1}, and 5.0×10^{-1} for *train*, *in-dis-test*, and *out-dis-test* data, respectively. Even though the unknown equations do not seem too complicated, they are still difficult to solve together with the PDE as a whole. The results show that for these particular problems, ANNs serve better because they allow better control during training in form of constraints, and they produce more regularized outputs than high order polynomials. However, while ANNs are not unique in their selection, they are most convenient for our implementation.

D.2. Noise Robustness Test of FINN

In this section, we test the robustness of FINN when trained using noisy data. All the synthetic data is generated with the same parameters, only added with noise from the distribution $\mathcal{N}(0.0, 0.05)$. For the Burgers' equation (Figure 22a and Figure 23a), the average rMSE values are $6.9\times10^{-3}\pm1.1\times10^{-5}$, $2.5\times10^{-2}\pm6.6\times10^{-5}$, and $7.1\times10^{-3}\pm1.1\times10^{-5}$ for the *train*, *in-dis-test*, and *out-dis-test* prediction, respectively. For the diffusion-sorption equation (Figure 22b and Figure 23b), the average rMSE values are $3.9\times10^{-2}\pm6.9\times10^{-5}$, $3.6\times10^{-2}\pm5.3\times10^{-5}$, and $7.1\times10^{-2}\pm1.0\times10^{-4}$ for the *train*, *in-dis-test*, and *out-dis-test* prediction, respectively. For the diffusion-reaction equation (Figure 22c and Figure 23c), the average rMSE values are $4.1\times10^{-2}\pm6.8\times10^{-3}$, $1.3\times10^{-1}\pm6.9\times10^{-2}$, and $2.2 \times 10^{-1} \pm 1.3 \times 10^{-2}$ for the *train*, *in-dis-test*, and *out-dis-test* prediction, respectively. For the Allen–Cahn

Table 16: Comparison of rMSE on the *Train* data for individual runs with and without Neural ODE (NODE) on the different equations. Average of models without (w/o) NODE is calculated over all individual runs. Average scores of models with (w/) NODE are taken from Table 1. Left and right columns for the different equations show the MSE and epoch from which on `NaN` values occured, respectively. FINN was trained for 100 epochs, overall, i.e. 100 in the second columns corresponds to a training without any `NaN` values encountered.

Run	Burgers' (1D)		Diffusion-sorption		Diffusion-reaction		Allen–Cahn	
	rMSE	Ep.	rMSE	Ep.	rMSE	Ep.	rMSE	Ep.
1	1.7×10^{-5}	100	-	0	3.3×10^{-1}	2	3.0×10^{-5}	100
2	2.2×10^{-5}	100	-	0	1.7×10^{-2}	14	2.8×10^{-4}	13
3	2.8×10^{-3}	100	-	0	8.4×10^{-3}	27	5.0×10^{-5}	100
4	4.0×10^{-5}	100	-	0	5.2×10^{-2}	78	2.1×10^{-6}	100
5	3.7×10^{-3}	2	-	0	5.7×10^{-2}	100	3.8×10^{-4}	100
6	1.7×10^{-5}	100	-	0	2.4×10^{-1}	83	1.2×10^{-6}	100
7	1.3×10^{-5}	100	-	0	1.1×10^{-2}	31	7.3×10^{-7}	100
8	3.4×10^{-3}	3	-	0	1.3×10^{-2}	15	2.5×10^{-6}	100
9	4.0×10^{-5}	13	-	0	3.1×10^{-3}	100	2.5×10^{-6}	100
10	1.4×10^{-5}	100	-	0	2.4×10^{-3}	45	3.3×10^{-5}	39
Avg w/o NODE	$(1.0 \pm 1.5) \times 10^{-3}$	72	-	0	$(7.4 \pm 11.0) \times 10^{-2}$	50	$(7.8 \pm 12.9) \times 10^{-5}$	85
Avg w/ NODE	$(7.8 \pm 8.2) \times 10^{-6}$	100	(omitted)	100	$(1.7 \pm 0.4) \times 10^{-3}$	100	$(3.2 \pm 3.5) \times 10^{-5}$	100

equation (Figure 22d and Figure 23d), the average rMSE values are $1.2 \times 10^{-2} \pm 4.8 \times 10^{-6}$, $3.6 \times 10^{-3} \pm 1.3 \times 10^{-5}$, and $2.9 \times 10^{-3} \pm 7.3 \times 10^{-6}$ for the *train*, *in-dis-test*, and *out-dis-test* prediction, respectively. These results show that even though FINN is trained with noisy data, it is still able to capture the essence of the equation and generalize well to different initial and boundary conditions. Additionally, the prediction is consistent, shown by the low values of the rMSE standard deviation, as well as the very narrow confidence interval in the plots.

D.3. Neural ODE Ablation and Runtime Analysis

Here, we investigate whether FINN or Neural ODE (NODE) alone can account for the high accuracy. We find that neither FINN nor NODE alone reach satisfactory results, and it is thus the combination of FINN's modular structure with NODE's adaptive time stepping mechanism that leads to the reported performance.

NODE without FINN The relevance of FINN is demonstrated by an experiment where NODE is applied on top of a conventional CNN stem to take the role of FINN. Results are reported in Table 2 and demonstrate the limitations of a CNN-NODE black box model.

FINN without NODE In order to stress the role of Neural ODE in FINN, we conduct an ablation by removing the Neural ODE part and directly feeding the output of FINN as input in the next time. This amounts to traditional closed loop training or an Euler ODE integration scheme, where the model directly predicts and outputs the delta from $u^{(t)}$ to $u^{(t+1)}$ (also referred to as residual training). Results are summarized in Table 16 and unveil two major effects of Neural ODE.

First, it helps stabilizing the training reasonably: We observe FINN having immense difficulties without Neural ODE at learning the diffusion-sorption equation (not even completing a single epoch without producing `NaN` values). Similar but less dramatic failures were observed on the Burgers', diffusion-reaction, and Allen–Cahn equations, where FINN without Neural ODE on average only managed 72, 50, and 85 epochs, respectively, without producing `NaN` values. This consolidates the supportive aspect of the Neural ODE component, which we explain is due to its adaptive time-stepping feature. More importantly, the adaptive time-stepping of Neural ODE makes it possible to apply FINN to experimental data in the first place, where the time deltas between successive measurements are not constant.

Second and as to be expected, the runtime when adding Neural ODE in the computations increases significantly. We compare the runtime for each model, run on a CPU with i9-9900K core, a clock speed of 3.60 GHz, and 32 GB RAM. Additionally, we also perform the comparison of GPU runtime on a GTX 1060 (i.e. with 6 GB VRAM). The results are summarized in Table 17. Note that the purpose of this comparison is not an optimized benchmark, but only to show that the runtime of FINN is comparable with the other models, especially when run in CPU. When run on GPU, however, FINN runs slightly slower. This is caused by the fact that FINN's implementation is not yet optimized for GPU. More importantly, the Neural ODE package we use benefits only from a larger batch size. As shown in

Table 17: Forward pass runtimes (in seconds) of a single sequence for pure ML and physics-aware neural network methods on the different equations. CPU: Intel i9-9900K, 3.60 GHz processor, GPU: Nvidia GeForce GTX 1060. Note that the datasets (equations) have different sequence lengths. Moreover, the data for PINN on the diffusion-reaction equation did not fit on the GPU and due to instability issues, APHYNITY could not be trained on the diffusion-sorption equation.

		Pure ML models					Physics-aware neural networks			
Equation	Unit	TCN	CNN-NODE	ConvLSTM	DISTANA	FNO	PINN	PhyDNet	APHYNITY	FINN
Burgers' (1D)	CPU	0.45	0.19	0.03	0.04	0.10	0.03	0.11	0.24	0.07
	GPU	0.13	0.33	0.06	0.08	0.20	0.01	0.20	0.45	0.17
Diffusion-sorption (1D)	CPU	5.75	1.34	0.32	0.41	1.03	0.28	1.06	N/A	2.45
	GPU	1.58	2.35	0.58	0.68	2.00	0.01	1.94	N/A	6.10
Diffusion-reaction (2D)	CPU	14.00	1.05	0.18	0.14	0.27	2.47	0.16	0.37	0.31
	GPU	1.01	0.56	0.03	0.04	0.14	N/A	0.12	0.26	0.37
Allen-Cahn (1D)	CPU	1.71	0.10	0.07	0.08	0.21	0.06	0.22	0.45	0.05
	GPU	0.27	0.20	0.13	0.16	0.41	0.01	0.40	0.78	0.14

Figure 24, GPU is faster for batch size larger than 20 000, whereas the maximum size that we use in the example is 2 401. With smaller batch size, CPU usage is faster.

In general, only the PINN model has a benefit when computed on the GPU, since the function is only called once on all batches. This is different for all other models that have to unroll a prediction of the sequence into the future recurrently (except for TCN which is a convolution approach that is faster on GPU). Accordingly, The overhead of copying the tensors to GPU outweighs the GPU's parallelism benefit, compared to directly processing the sequence on the CPU iteratively. On the two-dimensional diffusion-reaction benchmark, the GPU's speed-up comes into play, since in here, the simulation domain is discretized into $49 \times 49 = 2401$ volumes; compared to 49 for the Burgers' (1D) and Allen–Cahn, and 26 for the diffusion-sorption equations. Note that we have observed significantly varying runtimes on different GPUs (i.e. up to three seconds for FINN on Burgers' on an RTX 3090), which might be caused by lack of support of certain packages for a particular hardware, but further investigation is required.

Additionally, we want to emphasize that the higher runtime of FINN on GPU is not caused by the time step adaptivity. In fact, employing the adaptive time stepping strategy is cheaper than choosing all time steps to be small enough (to guarantee numerical stability). As we learn a PDE, we have no exact knowledge to derive a dedicated time integration scheme, but the adaptive Runge–Kutta method is one of the best generic choices. As our PDE and its characteristics change during training, time step adaptivity is a real asset, as, for example, the Courant–Friedrichs–Lewy (CFL) condition (Courant et al., 1967) would consistently change throughout the training. Therefore, the time step adaptivity is not a bottleneck, but rather a solution for a more efficient computation.

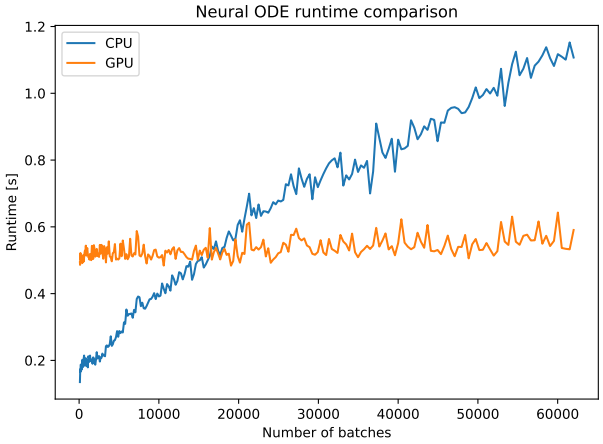


Figure 24: Runtime comparison of Neural ODE with a single hidden layer consisting of 50 hidden nodes run on CPU and GPU for 1 000 time steps. The benefit of a GPU unfolds when larger batch sizes ($> 20\,000$) are used.

In relation to FINN's limitation (see subsection B.2), topics like numerical stabilization schemes for larger time steps, adaptive spatial grid, optimized implementation in a High Performance Computing (HPC) setting, parallelization, etc. hold large potential for future work.

D.4. Training PINN With Finer Spatial Resolution

In order to determine whether the reduced accuracy of PINN in our experiments was caused by a coarse spatial resolution—we only used 49, 26, and 49×49 spatial positions for Burgers' (1D), diffusion-sorption, and diffusion-reaction—another experiment was conducted where the spatial resolutions were increased to $N_x = 999$, $N_x = 251$, and $N_x = 99$, $N_y = 99$, respectively. As reported in Table 18, Figure 25, and Figure 27 (top), the performance

Table 18: rMSE of PINN trained on data with finer resolution from ten different training runs for the diffusion-reaction equation.

Run	*Train*	*In-dis-test*	*Out-dis-test*
1	1.9×10^{-3}	6.3×10^{-1}	-
2	2.4×10^{-3}	4.6×10^{-1}	-
3	1.2×10^{-3}	1.2×10^{-1}	-
4	1.3×10^{-2}	1.2×10^{0}	-
5	6.9×10^{-4}	2.2×10^{-1}	-
6	3.9×10^{-3}	6.5×10^{0}	-
7	1.8×10^{-3}	2.8×10^{-1}	-
8	1.2×10^{-3}	2.3×10^{-1}	-
9	5.3×10^{-4}	1.0×10^{-1}	-
10	4.5×10^{-4}	1.4×10^{-1}	-

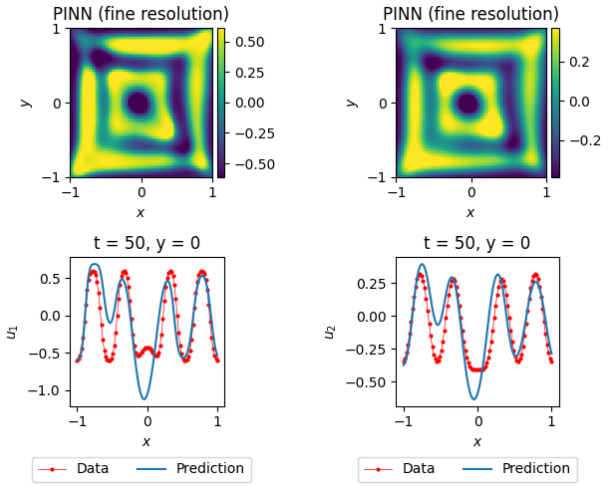


Figure 25: Plots of the diffusion-reaction equation's activator u using PINN trained with finer resolution dataset. *In-dis-test* prediction (left) and *out-dis-test* (right)

on diffusion-reaction increased slightly, however without reaching FINN's accuracy. Identical results were achieved in the Burgers' and diffusion-sorption equations but are omitted due to high conceptual similarity.

D.5. PhyDNet With Original Amount of Parameters

To verify whether our reduction of parameters and the removal of the encoder and decoder layers caused PhyDNet to perform worse, we repeated the experiments using the original PhyDNet architecture as proposed in (Guen & Thome, 2020). However, our results indicate no significant changes in performance, as reported in Table 19, Figure 26, and Figure 27 (bottom). Again, results for the inhibitor u_2 as well as for the Burgers' and diffusion-sorption equations were omitted due to high conceptual similarity.

Table 19: rMSE of PhyDNet using the original network size from ten different training runs for the diffusion-reaction equation.

Run	*Train*	*In-dis-test*	*Out-dis-test*
1	1.2×10^{-3}	1.4×10^{0}	2.2×10^{0}
2	3.9×10^{-4}	5.2×10^{-1}	1.0×10^{0}
3	2.9×10^{-4}	5.4×10^{-1}	1.2×10^{0}
4	6.1×10^{-4}	5.8×10^{-1}	1.1×10^{0}
5	4.6×10^{-4}	5.2×10^{-1}	2.5×10^{0}
6	7.6×10^{-4}	9.8×10^{-1}	3.0×10^{0}
7	4.7×10^{-4}	4.6×10^{-1}	2.6×10^{0}
8	4.7×10^{-4}	4.7×10^{-1}	7.5×10^{-1}
9	3.1×10^{-4}	5.0×10^{-1}	1.2×10^{0}
10	4.7×10^{-4}	4.9×10^{-1}	2.3×10^{0}

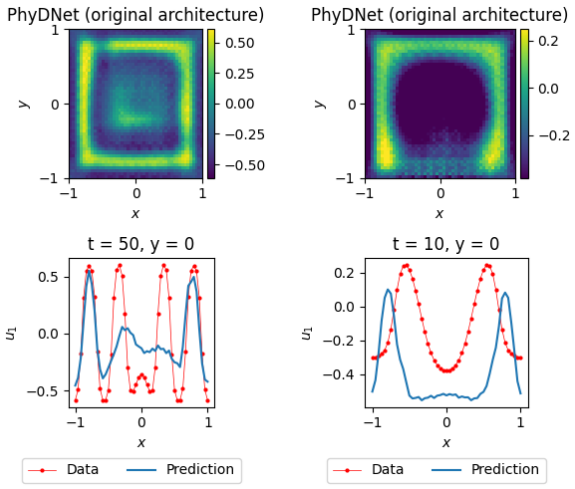


Figure 26: Plots of the diffusion-reaction equation's activator u using PhyDNet with the original network size. *In-dis-test* prediction (left) and *out-dis-test* (right).

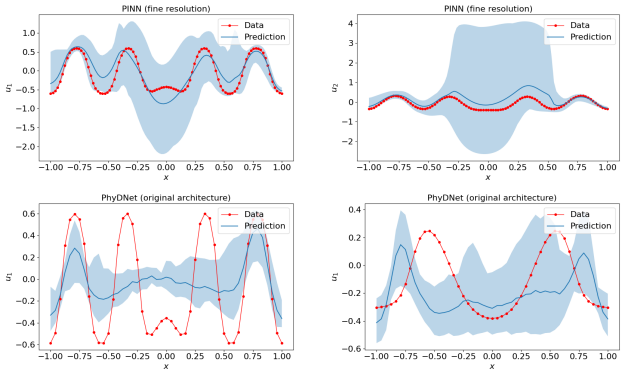


Figure 27: Prediction mean (with 95% confidence interval) of the activator (left) and inhibitor (right) in the diffusion-reaction equation at $t = 50$ compared with the *in-dis-test* data. Top: PINN fine resolution. Bottom: PhyDNet original architecture.