COMPOSING PARTIAL DIFFERENTIAL EQUATIONS WITH PHYSICS-AWARE NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce a compositional physics-aware 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 (Burger's, diffusion-sorption, diffusion-reaction) demonstrate FINN's superior process 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 as universal function approximators (Cybenko, 1989). Their effective learning ability, however, greatly depends on domain and task-specific pre-structuring 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 can 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 huge 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) model—a physics-aware neural network structure adhering to the idea of spatial and temporal discretization in numerical simulation. FINN consists of multiple neural network modules that are designed to account for specific parts of advection-diffusion equations, a class of partial differential equations (PDEs). This modularization allows 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 neural network model, explicitly designed to generalize over initial and boundary conditions, demonstrating 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 benefits 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.

2 RELATED WORK

Non-physics-aware ANN architectures 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 enormous 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 ANN architectures When designed to satisfy physical equations, ANNs are reported to have greater robustness to out-of-distribution data. 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 can actually not be applied to other different initial and boundary conditions, which limits its applicability in real-world scenarios.

Other methods, as proposed by Long et al. (PDENet, 2018), Guen & Thome (PhyDNet, 2020), or Sitzmann et al. (SIREN, 2020), implement an implicit satisfaction of the first n derivatives in order to achieve a physically plausible nonlinear behavior, but lack the option to include physical equations. The same limitation applies to Li et al. (2020a;b) when learning operators to approximate PDEs, or Seo et al. (2019)'s physics-aware graph neural network. Guen et al. (APHYNITY, 2020) propose 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.

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 In this work we focus on modeling spatiotemporal physical processes. Specifically, we consider systems that are governed by the advection-diffusion type equation[1] (Smolarkiewicz, 1983):

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

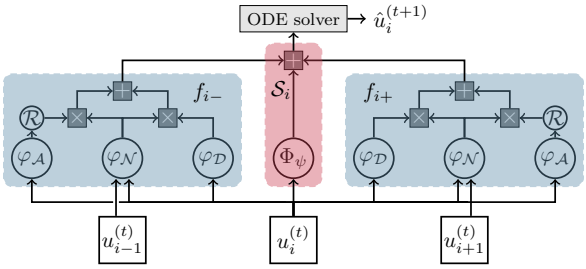
where u is the quantity of interest, t is time, x is the spatial coordinate, D is the diffusion coefficient, v is the advection velocity, and q is the source/sink term. Equation 1 can be partitioned into three parts: the storage term, the flux terms, and the source/sink term. The storage term $\frac{\partial u}{\partial t}$ 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 positions. The source/sink term $q(u)$ describes the generation or elimination of the quantity u.

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, and the state kernel, modeling the source/sink term as well as the storage term. The overall FINN architecture is illustrated 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:

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

[1]For better clarity, the explanation of FINN's structure is written for the 1D advection-diffusion problem. It is also extendable for 2D cases, as shown in subsection A.3 of the appendix.



$$\frac{\partial u_i}{\partial t} = \overbrace{D\left(u_i\right)}^{\varphi_\mathcal{D}} \overbrace{\frac{\partial^2 u_i}{\partial x^2}}^{\varphi_\mathcal{N}} - \overbrace{v\left(u_i\right)}^{\varphi_\mathcal{A}} \overbrace{\frac{\partial u_i}{\partial x}}^{\varphi_\mathcal{N}} + \overbrace{q\left(u_i\right)}^{\Phi_\psi}$$

$$\underbrace{\underbrace{\qquad\qquad}_{\mathcal{F}_i = \{f_{i-}, f_{i+}\}} \qquad\qquad}_{\mathcal{S}_i}$$

Figure 1: Illustration of the flux and state kernels in the FINN in one-dimensional case (left) and a detailed assignment of the individual modules with their contribution to Equation 1 (right).

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 (which are realized as 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) take the inputs u_i and its neighbor u_{i-1} and u_{i+1}. The inputs then enter the module $\varphi_\mathcal{N}$, which is a linear layer with the purpose to learn the numerical FVM stencil and outputs the approximation of the spatial derivative $\frac{\partial u}{\partial x}$. This signifies that the weight of $\varphi_\mathcal{N}$ should amount to $[-1, 1]$, which corresponds to $[u_i, u_{i-1}]$ and $[u_i, u_{i+1}]$ (i.e. simple difference between neighboring control volumes) in ideal one-dimensional problems.

The flux kernel additionally contains two modules denoted as $\varphi_\mathcal{A}$ and $\varphi_\mathcal{D}$ that are responsible for advective and diffusive fluxes, respectively. Moreover, the module $\mathcal{R}$ applies an upwind differencing scheme, which prevents numerical instability in the first order spatial derivative calculation (Versteeg & Malalasekera, 1995). It ensures that calculation of the advective fluxes using $\varphi_\mathcal{A}$ is performed only on one control volume surface, whereas calculation of the diffusive fluxes using $\varphi_\mathcal{D}$ is performed on both surfaces. This means, for the advective flux calculation, the summation of the numerical stencil from both f_{i-} and f_{i+} will only lead to $[-1, 1]$, which corresponds to either $[u_i, u_{i-1}]$ 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, the summation of the numerical stencil will lead to the classical one-dimensional numerical Laplacian with $[1, -2, 1]$ corresponding to $[u_{i-1}, u_i, u_{i+1}]$ (i.e. second order spatial derivative) because the calculation of $\varphi_\mathcal{D}$ is performed on both surfaces. Accordingly, module $\mathcal{R}$ generates the inductive bias to make $\varphi_\mathcal{A}$ only approximate the advective flux, and $\varphi_\mathcal{D}$ the diffusive flux.

For the advective flux calculation, if the velocity v is a function of u, the module $\varphi_\mathcal{A}$ approximates this dependence, i.e. $\varphi_\mathcal{A}(u) \approx v(u)$. Otherwise, $\varphi_\mathcal{A}$ is a scalar value $\varphi_\mathcal{A} \equiv v$, which can be also set as a learnable parameter. The output of $\varphi_\mathcal{A}$ is activated by the following case-sensitive ReLU $\mathcal{R}$:

$$\mathcal{R}(\varphi_\mathcal{A}) = \begin{cases} \mathrm{ReLU}(\varphi_\mathcal{A}), & \text{on } f_{i-}, \\ -\mathrm{ReLU}(-\varphi_\mathcal{A}), & \text{on } f_{i+}, \end{cases} \tag{3}$$

and then multiplied with the output of $\varphi_\mathcal{N}$ (i.e. $\frac{\partial u}{\partial x}$) to obtain the corresponding advective flux at the surface. Similarly for the diffusive fluxes, if the diffusion coefficient D depends on u, the module $\varphi_\mathcal{D}$ takes u as an input to learn the function $\varphi_\mathcal{D}(u) \approx D(u)$. Otherwise, $\varphi_\mathcal{D}$ is a scalar value $\varphi_\mathcal{D} \equiv D$, which can also be set as a learnable parameter. The output of $\varphi_\mathcal{D}$ is not activated by the module $\mathcal{R}$, but is directly multiplied with the output of $\varphi_\mathcal{N}$ to obtain the corresponding diffusive flux at the surface. Consequently, integration of subkernels f_j in Equation 2 and substitution of Equation 3 leads to the following representation:

$$\mathcal{F}_i = \begin{cases} \varphi_\mathcal{N}(u_i, u_{i-1})[\varphi_\mathcal{D} - \varphi_\mathcal{A}]_{f_{i-}} + \varphi_\mathcal{N}(u_i, u_{i+1})[\varphi_\mathcal{D}]_{f_{i+}}, & \text{if } \varphi_\mathcal{A} > 0, \\ \varphi_\mathcal{N}(u_i, u_{i-1})[\varphi_\mathcal{D}]_{f_{i-}} + \varphi_\mathcal{N}(u_i, u_{i+1})[\varphi_\mathcal{D} - \varphi_\mathcal{A}]_{f_{i+}}, & \text{otherwise.} \end{cases} \tag{4}$$

Boundary conditions A means of applying boundary conditions in the model is essential when solving PDEs. Currently available models mostly adopt a convolutional structure for modelling spatiotemporal processes. However, a convolutional structure only allows a constant value to be padded at the domain boundaries (e.g. zero-padding or mirror-padding). This corresponds to a Dirichlet boundary condition, which is defined using a constant value of the quantity of interest at the

boundary. Other types of frequent boundary conditions are Neumann and Cauchy. They are defined as a derivative of the quantity of interest, and hence cannot be easily implemented in these models. As a consequence of performing flux calculation at the control volume surfaces, flux kernels allow straightforward implementation of different types of numerical boundary conditions. With Dirichlet boundary condition, a constant value $u = u_b$ is set as u_{i-1} or u_{i+1} at the corresponding boundary. With Neumann boundary condition ν, the output of f_j 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 Equation 1. The source/sink (if required) is learned using the module $\Phi_\psi \approx q(u)$, which takes u as input. The storage, $\frac{\partial u}{\partial t}$, is then calculated using the output of the flux kernel and module Φ_ψ of the state kernel:

$$\mathcal{S}_i = \mathcal{F}_i + \Phi_\psi(u_i) \approx \frac{\partial u_i}{\partial t}. \tag{5}$$

By doing so, the PDE in Equation 1 is now reduced to a system of coupled ordinary differential equations (ODEs), which is a function of u_i, u_{i-1}, u_{i+1}, and t. Thus, the solutions of the coupled ODE system can be found using a 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 adaptive time-stepping strategies. When using the Neural ODE method, the predicted values of u at time t are fed back into the network as direct inputs to predict u at time $t + 1$. Therefore, the training is performed in a closed-loop setting, improving stability and accuracy of the prediction compared to networks trained with teacher forcing, i.e. one-step-ahead prediction (Praditia et al., 2020). In a bigger picture, FINN takes only the initial condition u at time $t = 0$ and propagates the dynamics forward.

4 Experiments, Results & Discussion

4.1 Synthetic Dataset

To demonstrate FINN's performance in comparison to other neural network models, three different equations are considered as applications. First, *Burger's equation* (Basdevant et al., 1986) is chosen as a particularly challenging function, as it is a non-linear PDE with $v(u) = u$ that could lead to a shock in the solution $u(x, t)$. Second, the *diffusion-sorption equation* (Nowak & Guthke, 2016) is selected with an unknown (non-)linear retardation factor $R(u)$ as coefficient for the storage term. Third, the two-dimensional Fitzhugh-Nagumo equation (Klaasen & Troy, 1984) as candidate for a *diffusion-reaction equation* (Turing, 1952) is selected. This equation is particularly challenging because it consists of two non-linearly coupled PDEs to solve two main unknowns: the activator u_1 and the inhibitor u_2. Details on all three equations, data generation and architecture designs can be found in subsection A.1, subsection A.2, and subsection A.3 of the appendix, respectively.

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 in terms of 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.

These data are used to train and test FINN, compared with spatiotemporal deep learning models such as TCN, ConvLSTM, DISTANA and with physics-aware neural network models such as PINN and PhyDNet. All models are trained with ten different random seeds using PyTorch's default weight initializations, and the mean and standard deviation values of the prediction errors are summarized in Table 1 for *train*, *in-dis-test* and *out-dis-test*. The details of each run can be found in the appendix. It is also noteworthy that PINN cannot be tested on the *out-dis-test* dataset, because 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.

Table 1: Comparison of MSE and according standard deviation scores across ten repetitions between different deep learning (above dashed line) and physics-aware neural network (below dashed line) methods on the different equations. Best results are reported in bold.

Eqn.	Model	Params	Dataset *Train*	Dataset *In-dis-test*	Dataset *Out-dis-test*
Burger	TCN	38 500	$(1.6 \pm 3.4) \times 10^{-1}$	$(1.8 \pm 3.1) \times 10^{-1}$	$(1.6 \pm 3.3) \times 10^{-1}$
	ConvLSTM	13 200	$(6.8 \pm 9.9) \times 10^{-2}$	$(1.2 \pm 1.1) \times 10^{-1}$	$(7.3 \pm 9.5) \times 10^{-2}$
	DISTANA	25 126	$(1.8 \pm 1.0) \times 10^{-4}$	$(4.0 \pm 3.1) \times 10^{-3}$	$(1.5 \pm 1.6) \times 10^{-3}$
	PINN	3 021	$(5.1 \pm 0.5) \times 10^{-4}$	$(5.0 \pm 8.4) \times 10^{-3}$	-
	PhyDNet	37 718	$(7.2 \pm 2.2) \times 10^{-5}$	$(1.8 \pm 1.6) \times 10^{-1}$	$(4.5 \pm 2.5) \times 10^{-2}$
	FINN	**421**	$\mathbf{(2.8 \pm 2.9) \times 10^{-6}}$	$\mathbf{(2.5 \pm 3.1) \times 10^{-6}}$	$\mathbf{(2.8 \pm 2.9) \times 10^{-6}}$
Diffusion-sorption	TCN	3 834	$(9.7 \pm 13.5) \times 10^{-2}$	$(1.2 \pm 1.7) \times 10^{-1}$	$(1.1 \pm 1.4) \times 10^{-1}$
	ConvLSTM	3 960	$(3.2 \pm 2.9) \times 10^{-2}$	$(3.0 \pm 2.3) \times 10^{-2}$	$(5.8 \pm 4.0) \times 10^{-2}$
	DISTANA	3 739	$(4.6 \pm 2.5) \times 10^{-5}$	$(2.4 \pm 2.5) \times 10^{-3}$	$(4.6 \pm 4.6) \times 10^{-3}$
	PINN	3 042	$(4.7 \pm 8.4) \times 10^{-5}$	$(4.1 \pm 8.7) \times 10^{-3}$	-
	PhyDNet	37 815	$\mathbf{(3.5 \pm 1.7) \times 10^{-5}}$	$(9.1 \pm 15.4) \times 10^{-3}$	$(1.7 \pm 0.9) \times 10^{-2}$
	FINN	**528**	$(4.7 \pm 4.9) \times 10^{-5}$	$\mathbf{(1.3 \pm 1.3) \times 10^{-4}}$	$\mathbf{(4.1 \pm 4.0) \times 10^{-5}}$
Diffusion-reaction	TCN	31 734	$(1.4 \pm 0.9) \times 10^{-2}$	$(4.7 \pm 2.1) \times 10^{-1}$	$(1.5 \pm 0.8) \times 10^{-1}$
	ConvLSTM	24 440	$(8.7 \pm 21.3) \times 10^{-3}$	$(9.3 \pm 4.9) \times 10^{-2}$	$(1.5 \pm 1.3) \times 10^{-2}$
	DISTANA	75 629	$(4.0 \pm 3.4) \times 10^{-3}$	$(1.8 \pm 0.6) \times 10^{-1}$	$(1.3 \pm 0.9) \times 10^{-2}$
	PINN	3 062	$(2.7 \pm 1.8) \times 10^{-4}$	$(7.0 \pm 6.3) \times 10^{-2}$	-
	PhyDNet	185 589	$\mathbf{(7.5 \pm 0.9) \times 10^{-5}}$	$(7.8 \pm 1.8) \times 10^{-2}$	$(3.5 \pm 1.3) \times 10^{-2}$
	FINN	**882**	$(1.3 \pm 0.3) \times 10^{-4}$	$\mathbf{(2.1 \pm 0.5) \times 10^{-3}}$	$\mathbf{(6.1 \pm 0.3) \times 10^{-3}}$

Burger's 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 A.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 8 (left) and verifies that it successfully learned the advective velocity to be described by an identity function.

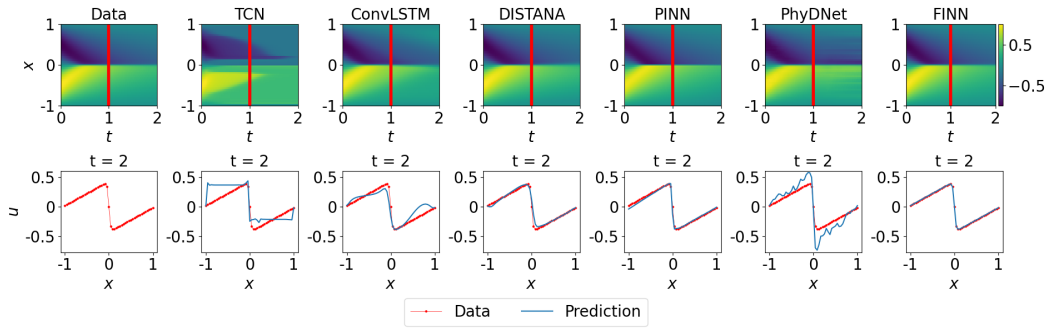


Figure 2: Plots of Burger's data and *in-dis-test* prediction using different models. The plots in the first row show the solution over x and t (the red lines mark the transition from *train* to *in-dis-test*), the second row visualizes the solution distributed in x at $t = 2$.

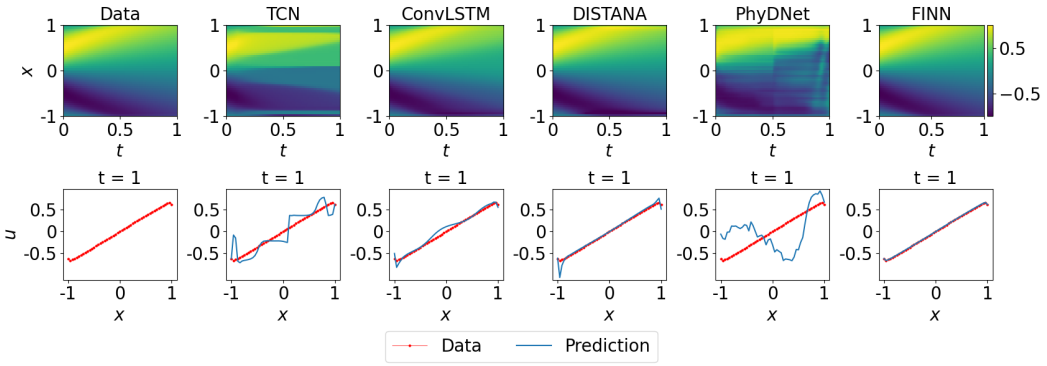


Figure 3: Plots of Burger's data and prediction of *out-dis-test* data using different models. The plots in the first row show the solution over x and t, the second row visualizes the solution over x at $t = 1$.

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 4 and Figure 5. 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 the train data (as detailed in subsection A.2). The retardation factor $R(u)$ learned by FINN's $\varphi_{\mathcal{D}}$ module is shown in Figure 8 (second from left). 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 6 and Figure 7. 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_ψ module are shown in Figure 8. The plots show that the module successfully learned the Fitzhugh-Nagumo reaction function, both for the activator and inhibitor.

Discussion 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,

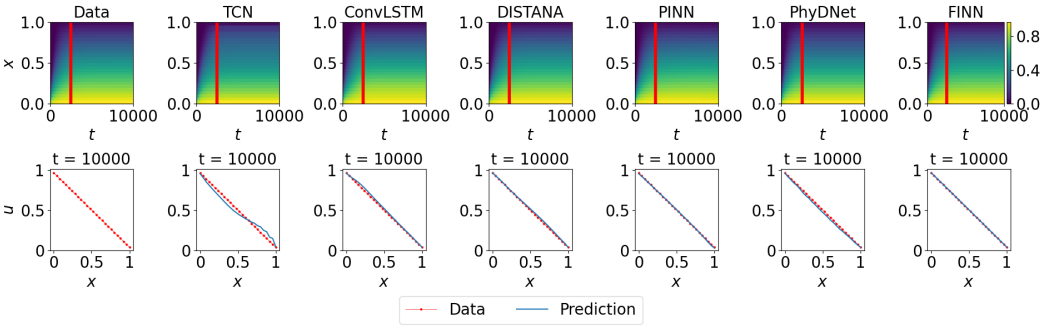


Figure 4: Plots of diffusion-sorption's dissolved concentration and *in-dis-test* prediction using different models. The plots in the first row show the solution over x and t (red lines mark the transition from *train* to *in-dis-test*), the second row visualizes the solution distributed in x at $t = 10\,000$.

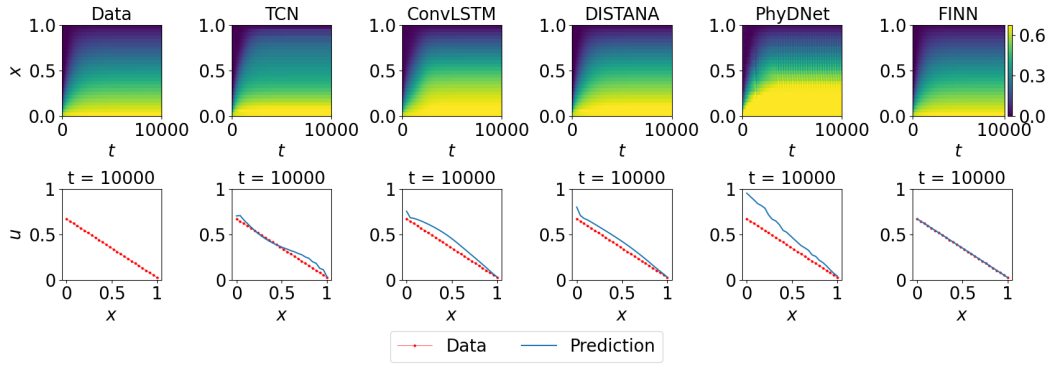


Figure 5: Plots of diffusion-sorption's dissolved concentration data and prediction of *out-dis-test* data using different models. The plots in the first row show the solution over x and t, the second row visualizes the solution distributed in x at $t = 10\,000$.

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 the Burger's equation, the retardation factor in the diffusion-sorption equation, and the reaction function in the diffusion-reaction equation 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 train 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 the middle, compromising between putting physical knowledge into the model and leaving room for learning from data. As a consequence, FINN shows excellent generalization ability. It significantly outperforms the 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 interpretability of the model.

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 5. Here, the test data is generated with a Dirichlet boundary condition $u(0, t) = 0.7$,

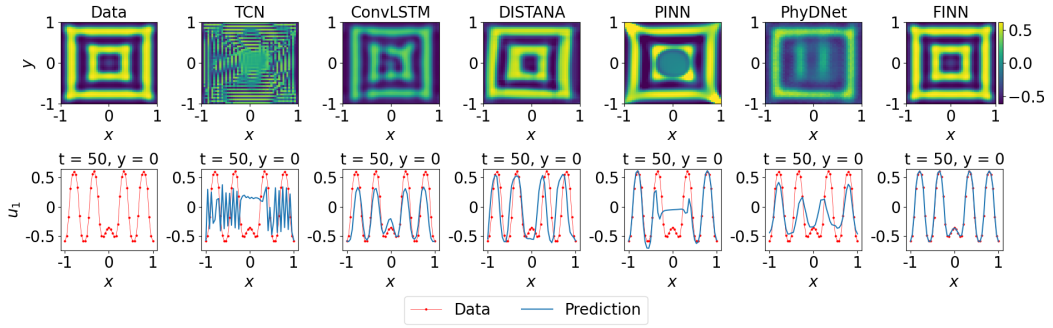


Figure 6: Plots of diffusion-reaction's activator data u_1 and extrapolated prediction using different models. The plots in the first row show the solution distributed over x and y at $t = 50$, and the plots in the second row show the solution distributed in x at $y = 0$ and $t = 50$.

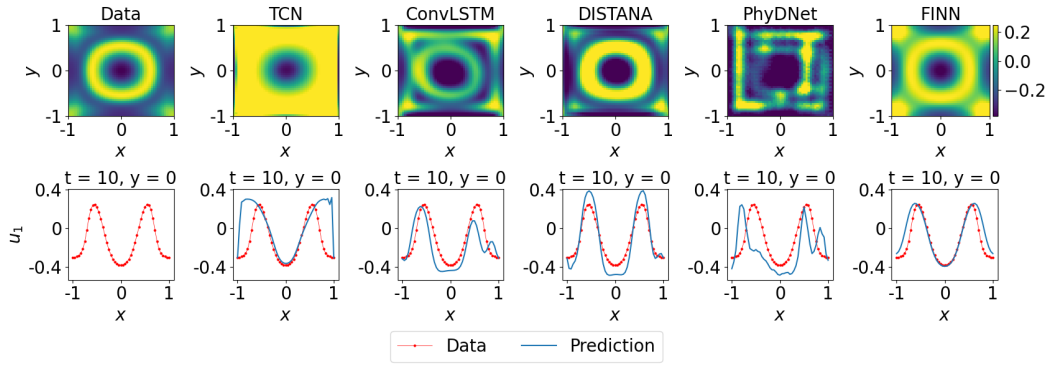


Figure 7: Plots of diffusion-reaction's activator data and prediction of unseen dataset using different models. The plots in the first row show the solution distributed over x and y at $t = 10$, and the plots in the second row show the solution distributed in x at $y = 0$ and $t = 10$.

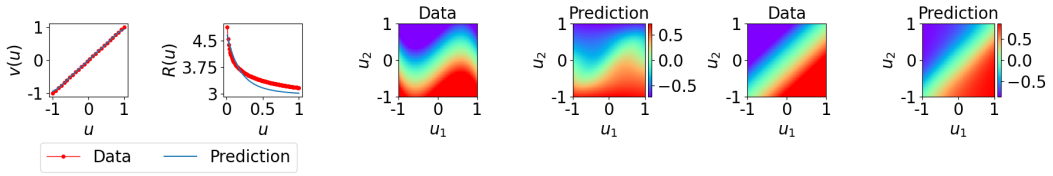


Figure 8: Plots of the learned u as a function of x compared to the data for Burger's (left) and diffusion-sorption (second from left). The learned activator u_1 and inhibitor u_2 functions (right plots of the pairs in center and right) in the diffusion-reaction equation are contrasted to the corresponding ground truth (left plots of the pairs).

which is lower than the value used in the train data, $u(0, t) = 1.0$. However, 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 performance for the three test cases. Nevertheless, we conducted an experiment showing that PINN performs slightly better and more consistently when trained on higher resolution data (see appendix, subsection A.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 the next section, 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.

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 8. 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.

4.2 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

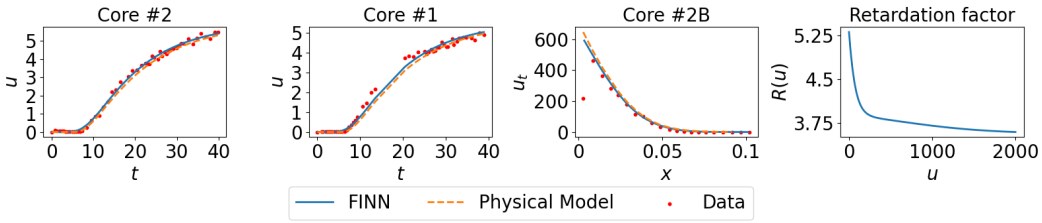


Figure 9: Breakthrough curve prediction of FINN (blue line) during training using data from core sample #2 (left), during testing using data from core sample #1 (second from left) and total concentration profile prediction using data from core sample #2B (second from right). The predictions are compared with the experimental data (red circles) and the results obtained using the physical model (orange dashed line). The learned retardation factor $R(u)$ is shown in the right-most plot.

from three different core samples that are taken from the same geographical area: #1,#2, and #2B (see subsection A.6 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).

FINN reaches a higher accuracy for the training with core sample #2, with $\text{MSE} = 4.84 \times 10^{-4}$ compared to a calibrated physical model from Nowak & Guthke (2016) with $\text{MSE} = 1.06 \times 10^{-3}$, 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 9 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 1.37×10^{-3} compared to the physical model that underestimates the breakthough curve (i.e. concentration profile) with $\text{MSE} = 2.50 \times 10^{-3}$. 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 1.16×10^{-3}, whereas the physical model overestimates the concentration with $\text{MSE} = 2.73 \times 10^{-3}$. 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 applying to a different boundary condition type.

5 Conclusion

Spatiotemporal dynamics often can be described by means of advection-diffusion type equations, such as Burger's, 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 can be seen as regularization, yielding robust predictions beyond the training data distribution.

With FINN, we have introduced a modular, physics-aware neural network 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. Another promising future work is the application of FINN to real-world weather data.

AUTHOR CONTRIBUTIONS

Hidden for anonymization reasons.

ACKNOWLEDGMENTS

Hidden for anonymization reasons

REPRODUCIBILITY STATEMENT

Code and data that are used for this paper can be found in the repository ***** *[hidden for anonymization reasons. Instead, the supplementary material of this submission contains an anonymous version of the repository's* `README.md` *file along with the according data and model scripts.]*

REFERENCES

C. Basdevant, M. Deville, P. Haldenwang, J.M. Lacroix, J. Ouazzani, R. Peyret, P. Orlandi, and A.T. Patera. 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.

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

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

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

R. Courant, K. Friedrichs, and H. Lewy. 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.

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

V.L. Guen and N. Thome. 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.

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

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

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

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

G.A. Klaasen and W.C. Troy. 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.

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

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

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

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

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

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

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

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. doi: 10.1016/j.jcp.2018.10.045.

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

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

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

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

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

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

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

A APPENDIX

This appendix is structured as follows: additional details on the equations used as well as model specifications and in-depth results are presented in subsection A.1 (Burger's), subsection A.2 (diffusion-sorption), and subsection A.3 (diffusion-reaction). Results of training PINN with high-resolution data are reported in subsection A.4, whereas results of the original PhyDNet architecture are outlined in subsection A.5. The soil parameters and simulation domain used in the diffusion-sorption laboratory experiment are presented in subsection A.6.

A.1 BURGER

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

Data The 1D generalized Burger's equation is written as

$$\frac{\partial u}{\partial t} = -v(u)\frac{\partial u}{\partial x} + D\frac{\partial^2 u}{\partial x^2}, \tag{6}$$

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 $x = [-1, 1]$, $t = [0, 1]$ 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 $N_t = 401$. 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 $u(x, 0) = \sin(\pi x)$.

Model architectures Both **TCN** and **ConvLSTM** are designed to have one input neuron, one hidden layer of size 32, 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 32 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** 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), as reported in the original work by Raissi et al. (2019). **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 $\varphi_{\mathcal{N}}$, $\varphi_{\mathcal{D}}$, $\mathcal{R}$ 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 of the ten different training runs as well as visualizations for the *train* (Table 2), *in-dis-test* (Table 3 and Figure 10), and *out-dis-test* (Table 4 and Figure 11) datasets.

Table 2: Closed loop MSE on the *train* data from ten different training runs for each model for the Burger's equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	4.1×10^{-2}	7.9×10^{-2}	1.2×10^{-4}	6.1×10^{-4}	8.0×10^{-5}	1.9×10^{-6}
2	5.1×10^{-2}	1.7×10^{-1}	1.8×10^{-4}	4.1×10^{-4}	4.7×10^{-5}	1.5×10^{-6}
3	3.6×10^{-2}	1.3×10^{-2}	6.4×10^{-5}	4.9×10^{-4}	6.8×10^{-5}	2.2×10^{-6}
4	4.3×10^{-2}	2.7×10^{-4}	2.9×10^{-4}	5.6×10^{-4}	1.2×10^{-4}	2.5×10^{-6}
5	5.4×10^{-2}	2.1×10^{-4}	2.2×10^{-4}	5.1×10^{-4}	6.7×10^{-5}	1.6×10^{-6}
6	3.9×10^{-2}	1.5×10^{-3}	1.3×10^{-4}	5.2×10^{-4}	5.5×10^{-5}	9.9×10^{-7}
7	6.6×10^{-2}	4.1×10^{-4}	5.4×10^{-5}	5.2×10^{-4}	5.1×10^{-5}	2.2×10^{-6}
8	1.2×10^{0}	3.6×10^{-2}	2.7×10^{-4}	4.8×10^{-4}	5.6×10^{-5}	3.0×10^{-6}
9	3.4×10^{-2}	5.5×10^{-2}	3.9×10^{-4}	5.5×10^{-4}	9.9×10^{-5}	1.1×10^{-5}
10	6.8×10^{-2}	3.2×10^{-1}	1.1×10^{-4}	5.0×10^{-4}	8.1×10^{-5}	4.5×10^{-7}

Table 3: Closed loop MSE on *in-dis-test* data from ten different training runs for each model for the Burger's equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	3.0×10^{-2}	1.2×10^{-1}	2.6×10^{-3}	4.1×10^{-3}	1.1×10^{-1}	1.6×10^{-6}
2	7.0×10^{-2}	3.7×10^{-1}	1.1×10^{-3}	1.3×10^{-3}	2.5×10^{-1}	1.3×10^{-6}
3	2.4×10^{-2}	2.0×10^{-1}	8.9×10^{-4}	2.9×10^{-3}	3.4×10^{-2}	1.9×10^{-6}
4	5.1×10^{-2}	5.6×10^{-3}	6.2×10^{-3}	3.7×10^{-3}	4.7×10^{-1}	2.1×10^{-6}
5	5.4×10^{-2}	2.6×10^{-3}	2.6×10^{-3}	2.4×10^{-3}	1.1×10^{-2}	1.3×10^{-6}
6	3.2×10^{-2}	1.0×10^{-2}	1.2×10^{-2}	1.9×10^{-3}	3.2×10^{-1}	6.8×10^{-7}
7	8.8×10^{-2}	5.5×10^{-3}	1.1×10^{-3}	2.1×10^{-3}	3.3×10^{-1}	1.8×10^{-6}
8	1.1×10^{0}	1.6×10^{-1}	4.0×10^{-3}	4.0×10^{-4}	2.7×10^{-1}	2.4×10^{-6}
9	3.3×10^{-2}	1.9×10^{-1}	4.0×10^{-3}	3.0×10^{-2}	2.6×10^{-2}	1.2×10^{-5}
10	3.6×10^{-1}	9.8×10^{-2}	6.0×10^{-3}	1.5×10^{-3}	8.0×10^{-3}	2.8×10^{-7}

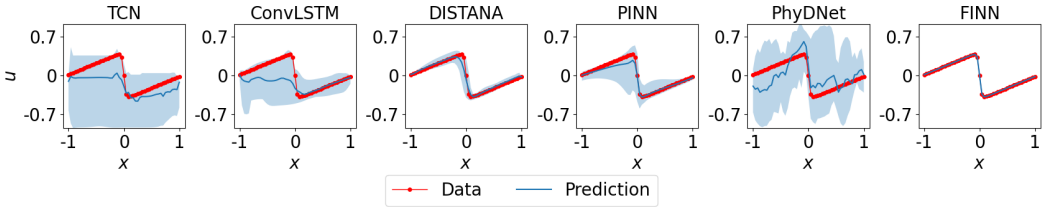


Figure 10: Prediction mean over ten different trained models (with 95% confidence interval) of the Burger's equation at $t = 2$ for the *in-dis-test* dataset.

Table 4: Closed loop MSE on *out-dis-test* data from ten different training runs for each model for the Burger's equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	3.6×10^{-2}	7.2×10^{-2}	5.9×10^{-3}	-	5.3×10^{-2}	1.9×10^{-6}
2	3.4×10^{-2}	1.9×10^{-1}	4.4×10^{-4}	-	2.1×10^{-2}	1.5×10^{-6}
3	3.3×10^{-2}	2.9×10^{-3}	2.9×10^{-3}	-	4.5×10^{-2}	2.2×10^{-6}
4	3.7×10^{-2}	8.9×10^{-4}	1.4×10^{-3}	-	2.0×10^{-2}	2.5×10^{-6}
5	4.2×10^{-2}	1.2×10^{-3}	1.0×10^{-3}	-	3.1×10^{-2}	1.6×10^{-6}
6	2.7×10^{-2}	1.4×10^{-3}	3.0×10^{-4}	-	1.1×10^{-1}	9.9×10^{-7}
7	6.7×10^{-2}	3.3×10^{-2}	6.2×10^{-4}	-	2.8×10^{-2}	2.2×10^{-6}
8	1.1×10^{0}	4.0×10^{-2}	7.0×10^{-4}	-	4.7×10^{-2}	3.0×10^{-6}
9	3.0×10^{-2}	8.5×10^{-2}	7.4×10^{-4}	-	4.5×10^{-2}	1.1×10^{-5}
10	1.2×10^{-1}	3.1×10^{-1}	7.0×10^{-4}	-	4.5×10^{-2}	4.5×10^{-7}

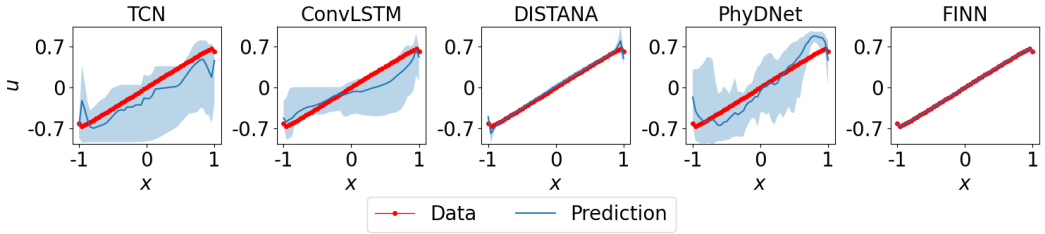


Figure 11: Prediction mean over ten different trained models (with 95% confidence interval) of the Burger's equation at $t = 1$ for the *out-dis-test* data.

A.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 1D diffusion-sorption equation is written as the following coupled system of equations:

$$\frac{\partial u}{\partial t} = \frac{D}{R(u)} \frac{\partial^2 u}{\partial x^2}, \tag{7}$$

$$\frac{\partial u_t}{\partial t} = D\phi \frac{\partial^2 u}{\partial x^2}, \tag{8}$$

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:

$$R(u) = 1 + \frac{1-\phi}{\phi} \rho_s k n_f u^{n_f - 1}, \tag{9}$$

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 $N_t = 2\,001$. 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 $c(0, t) = 0.7$.

Model architectures **TCN** is designed to have two input neurons, four hidden layers of size [4, 8, 8, 8], and two output neurons. **ConvLSTM** has two input- and output neurons and one hidden layer with 16 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 12 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 during teacher forcing. **PINN** was

Table 5: Closed loop MSE 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	3.6×10^{-2}	4.8×10^{-5}	2.1×10^{-5}	2.2×10^{-4}	3.1×10^{-5}	1.1×10^{-4}
2	2.0×10^{-1}	2.3×10^{-2}	1.6×10^{-5}	2.3×10^{-6}	2.5×10^{-5}	1.4×10^{-5}
3	8.1×10^{-2}	4.4×10^{-2}	2.7×10^{-5}	1.2×10^{-6}	3.2×10^{-5}	1.1×10^{-4}
4	2.0×10^{-1}	1.0×10^{-2}	9.3×10^{-5}	2.1×10^{-4}	2.9×10^{-5}	4.1×10^{-5}
5	1.8×10^{-3}	8.7×10^{-2}	6.2×10^{-5}	2.5×10^{-6}	7.7×10^{-5}	5.3×10^{-6}
6	2.7×10^{-4}	5.9×10^{-2}	4.0×10^{-5}	1.7×10^{-6}	4.4×10^{-5}	1.9×10^{-5}
7	3.2×10^{-4}	6.4×10^{-2}	7.7×10^{-5}	8.6×10^{-6}	1.8×10^{-5}	1.5×10^{-5}
8	1.7×10^{-2}	1.4×10^{-4}	2.5×10^{-5}	2.0×10^{-6}	2.4×10^{-5}	1.2×10^{-4}
9	4.3×10^{-1}	2.2×10^{-3}	5.9×10^{-5}	9.0×10^{-6}	2.1×10^{-5}	1.2×10^{-5}
10	6.6×10^{-4}	2.7×10^{-2}	3.9×10^{-5}	1.1×10^{-5}	5.2×10^{-5}	1.7×10^{-5}

Table 6: Closed loop MSE 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	1.9×10^{-2}	7.7×10^{-3}	6.9×10^{-4}	3.0×10^{-2}	3.8×10^{-3}	3.2×10^{-4}
2	2.4×10^{-1}	1.0×10^{-2}	8.5×10^{-5}	8.6×10^{-4}	8.4×10^{-3}	3.5×10^{-5}
3	4.7×10^{-2}	2.1×10^{-2}	1.7×10^{-3}	9.3×10^{-5}	2.9×10^{-4}	3.2×10^{-4}
4	2.4×10^{-1}	2.8×10^{-2}	3.5×10^{-3}	5.2×10^{-3}	3.6×10^{-4}	1.1×10^{-4}
5	3.5×10^{-2}	7.9×10^{-2}	7.3×10^{-3}	7.9×10^{-6}	2.8×10^{-2}	9.8×10^{-6}
6	2.4×10^{-2}	5.4×10^{-2}	3.1×10^{-4}	1.4×10^{-5}	4.8×10^{-2}	4.9×10^{-5}
7	1.4×10^{-3}	4.2×10^{-2}	6.5×10^{-3}	1.8×10^{-4}	1.0×10^{-3}	3.7×10^{-5}
8	3.9×10^{-2}	2.2×10^{-4}	9.4×10^{-4}	6.9×10^{-4}	2.5×10^{-4}	3.4×10^{-4}
9	5.6×10^{-1}	1.4×10^{-2}	1.8×10^{-3}	4.5×10^{-3}	2.4×10^{-4}	2.8×10^{-5}
10	3.4×10^{-2}	4.2×10^{-2}	9.9×10^{-4}	3.5×10^{-4}	2.8×10^{-4}	4.1×10^{-5}

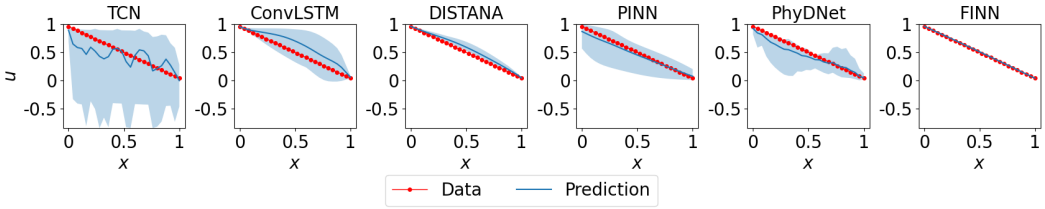


Figure 12: Prediction mean over ten different trained models (with 95% confidence interval) of the dissolved concentration in the diffusion-sorption equation at $t = 10\,000$ for the *in-dis-test* dataset.

Table 7: Closed loop MSE 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	4.7×10^{-2}	2.3×10^{-2}	2.9×10^{-3}	-	1.3×10^{-2}	9.7×10^{-5}
2	2.1×10^{-1}	4.3×10^{-2}	1.6×10^{-3}	-	1.1×10^{-2}	1.5×10^{-5}
3	9.2×10^{-2}	5.5×10^{-2}	3.4×10^{-3}	-	1.4×10^{-2}	9.5×10^{-5}
4	2.0×10^{-1}	1.9×10^{-2}	2.0×10^{-3}	-	1.1×10^{-2}	3.4×10^{-5}
5	4.4×10^{-2}	1.3×10^{-1}	4.6×10^{-4}	-	2.1×10^{-2}	4.9×10^{-6}
6	3.9×10^{-4}	1.0×10^{-1}	8.7×10^{-3}	-	4.4×10^{-2}	1.9×10^{-5}
7	9.3×10^{-4}	9.1×10^{-2}	1.6×10^{-2}	-	1.5×10^{-2}	1.5×10^{-5}
8	5.0×10^{-3}	2.6×10^{-3}	9.5×10^{-4}	-	1.2×10^{-2}	1.0×10^{-4}
9	4.6×10^{-1}	3.4×10^{-2}	7.1×10^{-3}	-	1.3×10^{-2}	1.2×10^{-5}
10	4.1×10^{-2}	8.0×10^{-2}	2.6×10^{-3}	-	1.3×10^{-2}	1.6×10^{-5}

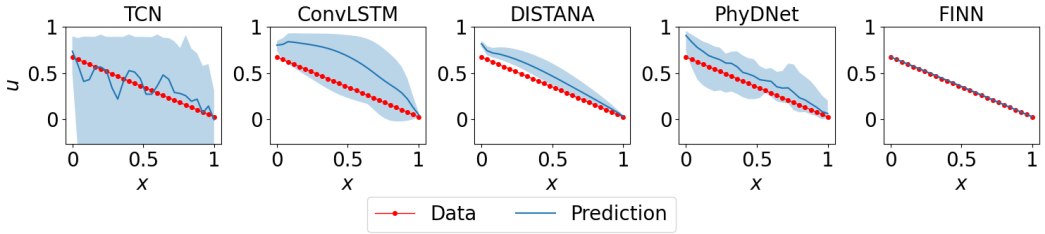


Figure 13: Prediction mean over ten different trained models (with 95% confidence interval) of the dissolved concentration in the diffusion-sorption equation at $t = 10\,000$ for the *out-dis-test* dataset.

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 of the ten different training runs as well as visualizations for the *train* (Table 5), *in-dis-test* (Table 6 and Figure 12), and *out-dis-test* (Table 7 and Figure 13) datasets. Results for the total concentration u_t were omitted due to high similarity to the concentration of the reported contamination solution u.

A.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 2D diffusion-reaction for that class of problems:

$$\frac{\partial u_1}{\partial t} = R_1(u_1, u_2) + D_1 \left(\frac{\partial^2 u_1}{\partial x^2} + \frac{\partial^2 u_1}{\partial y^2} \right), \tag{10}$$

$$\frac{\partial u_2}{\partial t} = R_2(u_1, u_2) + D_2 \left(\frac{\partial^2 u_2}{\partial x^2} + \frac{\partial^2 u_2}{\partial y^2} \right). \tag{11}$$

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:

$$R_1(u_1, u_2) = u_1 - u_1^3 - k - u_2, \tag{12}$$

$$R_2(u_1, u_2) = u_1 - u_2, \tag{13}$$

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 $\frac{\partial u_1}{\partial x}(-1, t) = 0$, $\frac{\partial u_1}{\partial x}(1, t) = 0$, $\frac{\partial u_2}{\partial x}(-1, t) = 0$, and $\frac{\partial u_2}{\partial x}(1, t) = 0$.

Table 8: Closed loop MSE on *train* data from ten different training runs for each model for the diffusion-reaction equation.

Run	TCN	ConvLSTM	DISTANA	PINN	PhyDNet	FINN
1	8.7×10^{-3}	1.7×10^{-3}	1.7×10^{-3}	2.7×10^{-4}	8.5×10^{-5}	1.0×10^{-4}
2	4.4×10^{-3}	1.6×10^{-3}	4.2×10^{-3}	5.6×10^{-4}	7.1×10^{-5}	1.6×10^{-4}
3	3.2×10^{-2}	1.4×10^{-3}	9.1×10^{-3}	2.1×10^{-4}	5.8×10^{-5}	1.4×10^{-4}
4	8.7×10^{-3}	7.8×10^{-4}	6.5×10^{-4}	2.8×10^{-4}	6.8×10^{-5}	1.2×10^{-4}
5	2.5×10^{-2}	1.5×10^{-3}	4.7×10^{-4}	2.8×10^{-4}	7.6×10^{-5}	1.0×10^{-4}
6	8.0×10^{-3}	4.8×10^{-4}	2.1×10^{-4}	2.5×10^{-4}	6.5×10^{-5}	1.7×10^{-4}
7	2.4×10^{-2}	3.7×10^{-3}	8.6×10^{-3}	6.1×10^{-5}	8.2×10^{-5}	1.6×10^{-4}
8	1.1×10^{-2}	1.0×10^{-3}	2.4×10^{-3}	1.0×10^{-4}	7.8×10^{-5}	1.0×10^{-4}
9	1.3×10^{-2}	2.2×10^{-3}	3.8×10^{-3}	3.2×10^{-5}	8.8×10^{-5}	1.5×10^{-4}
10	5.6×10^{-3}	7.3×10^{-2}	8.6×10^{-3}	6.2×10^{-4}	7.4×10^{-5}	1.1×10^{-4}

Table 9: Closed loop MSE 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	1.8×10^{-1}	8.0×10^{-2}	1.6×10^{-1}	7.7×10^{-2}	6.2×10^{-2}	2.9×10^{-3}
2	5.9×10^{-1}	5.7×10^{-2}	1.9×10^{-1}	2.3×10^{-1}	8.8×10^{-2}	1.6×10^{-3}
3	6.5×10^{-1}	9.2×10^{-2}	3.0×10^{-1}	3.5×10^{-2}	7.8×10^{-2}	1.7×10^{-3}
4	3.1×10^{-1}	6.7×10^{-2}	2.0×10^{-1}	2.6×10^{-2}	6.8×10^{-2}	2.9×10^{-3}
5	7.3×10^{-1}	4.9×10^{-2}	1.9×10^{-1}	1.3×10^{-1}	5.3×10^{-2}	1.7×10^{-3}
6	1.9×10^{-1}	5.4×10^{-2}	2.8×10^{-2}	8.9×10^{-2}	9.9×10^{-2}	2.0×10^{-3}
7	6.0×10^{-1}	1.0×10^{-1}	2.2×10^{-1}	5.4×10^{-2}	6.7×10^{-2}	1.5×10^{-3}
8	7.1×10^{-1}	8.6×10^{-2}	1.9×10^{-1}	1.3×10^{-2}	9.1×10^{-2}	2.3×10^{-3}
9	5.5×10^{-1}	2.3×10^{-1}	1.6×10^{-1}	2.3×10^{-2}	6.7×10^{-2}	1.8×10^{-3}
10	2.2×10^{-1}	1.2×10^{-1}	1.7×10^{-1}	2.6×10^{-2}	1.1×10^{-1}	2.1×10^{-3}

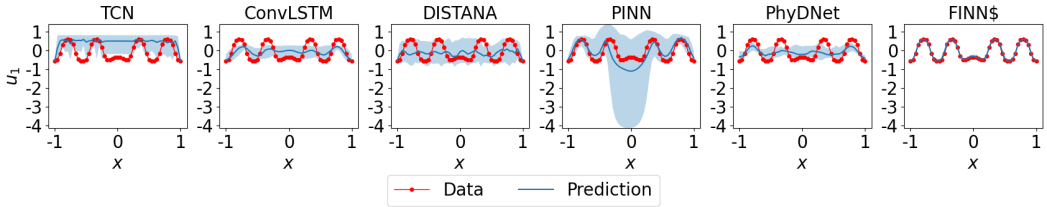


Figure 14: Prediction mean over ten different trained models (with 95% confidence interval) of the activator in the diffusion-reaction equation at $t = 50$ for the *in-dis-test* dataset.

Table 10: Closed loop MSE 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	2.3×10^{-2}	3.2×10^{-3}	9.4×10^{-3}	-	3.9×10^{-2}	6.5×10^{-3}
2	1.7×10^{-1}	1.3×10^{-2}	1.3×10^{-2}	-	5.4×10^{-2}	5.5×10^{-3}
3	2.5×10^{-1}	1.3×10^{-2}	2.1×10^{-2}	-	2.5×10^{-2}	5.9×10^{-3}
4	7.3×10^{-2}	7.2×10^{-3}	7.9×10^{-3}	-	4.2×10^{-2}	6.5×10^{-3}
5	2.6×10^{-1}	5.9×10^{-3}	2.6×10^{-3}	-	2.8×10^{-2}	6.1×10^{-3}
6	6.4×10^{-2}	1.5×10^{-2}	2.1×10^{-3}	-	2.9×10^{-2}	6.0×10^{-3}
7	2.2×10^{-1}	1.0×10^{-2}	1.9×10^{-2}	-	2.8×10^{-2}	5.7×10^{-3}
8	2.0×10^{-1}	3.9×10^{-3}	1.2×10^{-2}	-	2.1×10^{-2}	6.1×10^{-3}
9	2.3×10^{-1}	4.3×10^{-2}	1.3×10^{-2}	-	2.1×10^{-2}	6.0×10^{-3}
10	3.6×10^{-2}	3.7×10^{-2}	3.3×10^{-2}	-	5.9×10^{-2}	6.2×10^{-3}

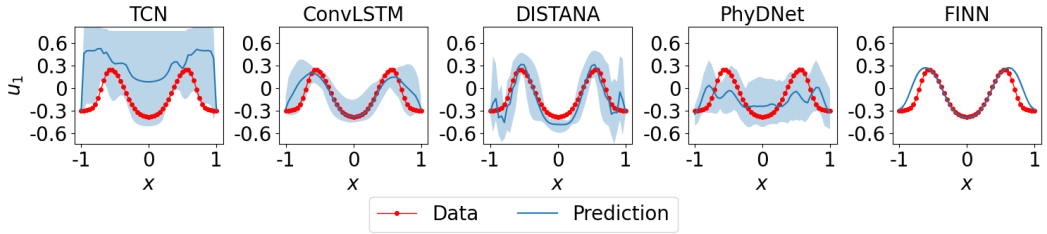


Figure 15: Prediction mean over ten different trained models (with 95% confidence interval) of the activator in the diffusion-reaction equation at $t = 10$ for the *out-dis-test* dataset.

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 $N_t = 501$. 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 32 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_ψ are used, with $\varphi_{\mathcal{D}}$ set as two learnable scalars that learn the diffusion coefficients D_1 and D_2, and Φ_ψ 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 of the ten different training runs as well as visualizations for the *train* (Table 8), *in-dis-test* (Table 9 and Figure 14), and *out-dis-test* (Table 10 and Figure 15) datasets. Results for the total inhibitor u_2 were omitted due to high similarity to the reported activator u_1.

A.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 at Burger, diffusion-sorption, and diffusion-reaction, respectively), another experiment was conducted per target equation where the spatial resolution was increased to $N_x = 999$, $N_x = 251$, and $N_x = 99$, $N_y = 99$, respectively. As reported in Table 11, Figure 16, and Figure 17, the performance increased slightly but by far did not reach FINN's accuracy. Identical results were achieved in the Burger's and diffusion-sorption equations but are omitted due to high conceptual similarity.

Table 11: MSE of PINN trained using data with finer resolution from ten different training runs for the diffusion-reaction equation.

Run	*Train*	*In-dis-test*	*Out-dis-test*
1	1.4×10^{-4}	7.8×10^{-2}	-
2	1.8×10^{-4}	5.6×10^{-2}	-
3	8.6×10^{-5}	1.5×10^{-2}	-
4	1.0×10^{-3}	1.5×10^{-1}	-
5	5.2×10^{-5}	2.7×10^{-2}	-
6	2.9×10^{-4}	8.1×10^{-1}	-
7	1.3×10^{-4}	3.4×10^{-2}	-
8	8.9×10^{-5}	2.9×10^{-2}	-
9	3.9×10^{-5}	1.3×10^{-2}	-
10	3.3×10^{-5}	1.7×10^{-2}	-

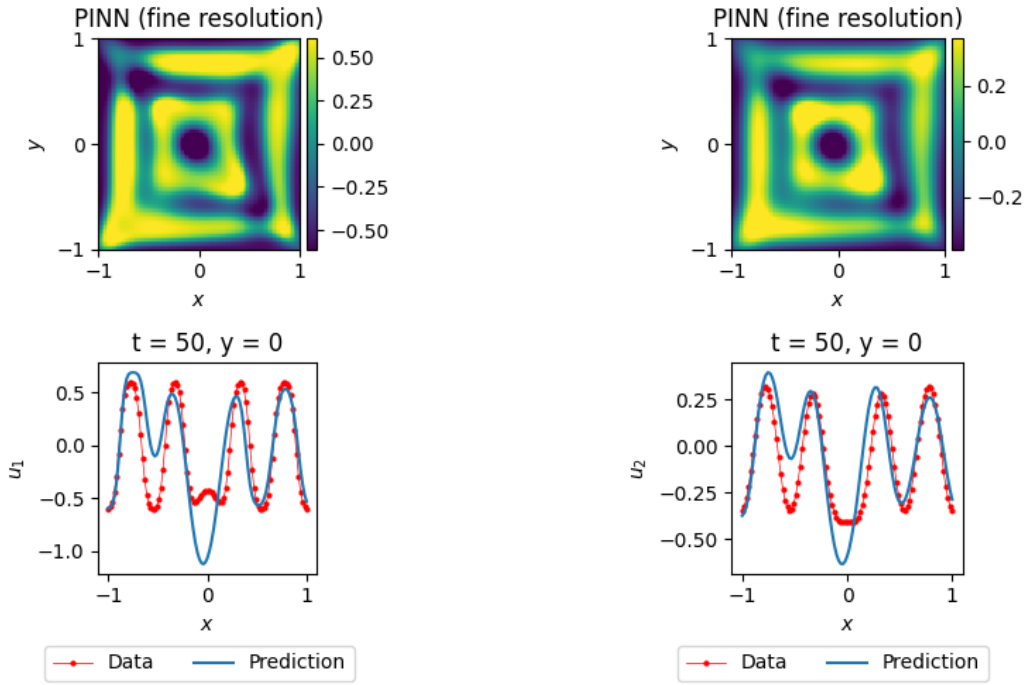


Figure 16: Plots of the activator (left) and inhibitor (right) extrapolated prediction in the diffusion-reaction equation using PINN trained with finer resolution dataset. The plots in the first row show the solution over x and t, and the plots in the second row show the solution distributed in x at $t = 50$.

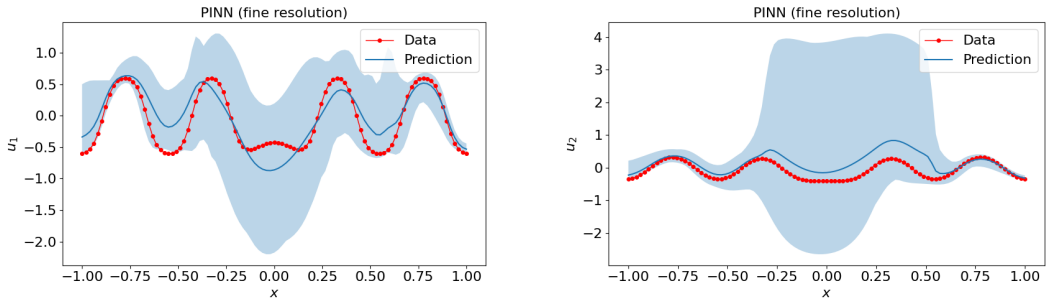


Figure 17: Prediction mean (with 95% confidence interval) of the activator (left) and inhibitor (right) in the diffusion-reaction equation at $t = 50$ compared with the *in-dis-test* data obtained using PINN trained with finer resolution dataset.

A.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 for the three equations of interest using the original PhyDNet architecture as proposed in Guen & Thome (2020). However, our results indicate no significant changes in performance, as reported in Table 12, Figure 18, and Figure 19. Again, results for the inhibitor u_2 as well as for the Burger's and diffusion-reaction equations were omitted due to high conceptual similarity.

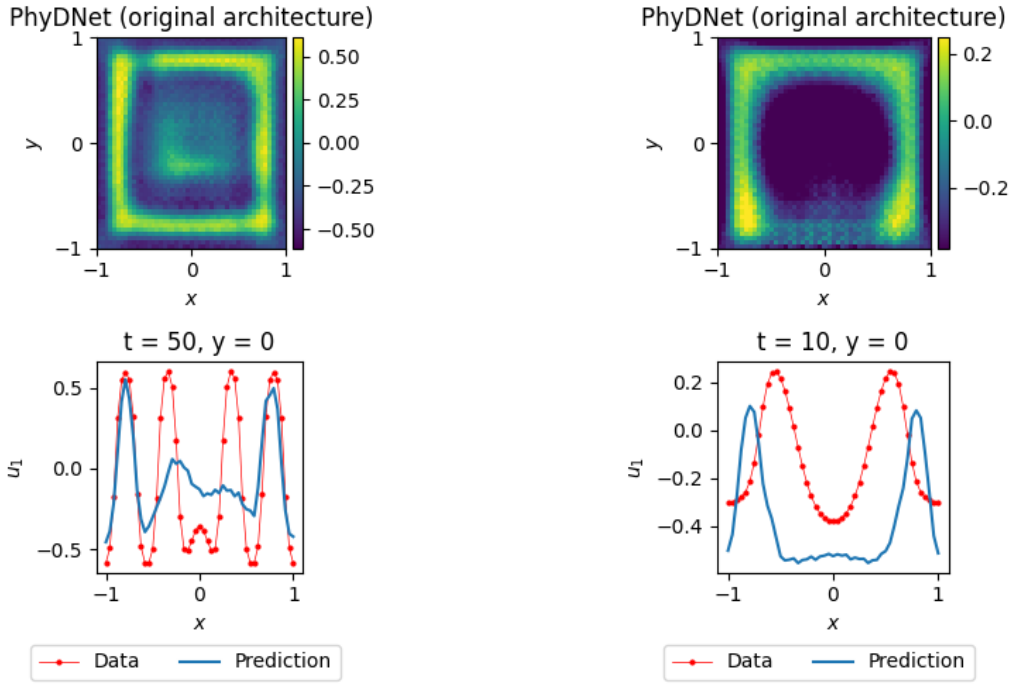


Figure 18: Plots of the activator *in-dis-test* prediction (left) and *out-dis-test* prediction (right) in the diffusion-reaction equation using PhyDNet with the original network size. The plots in the first row show the solution over x and t, and the plots in the second row show the solution distributed in x at $t = 50$ (for the *in-dis-test*) and $t = 10$ (for the *out-dis-test* dataset).

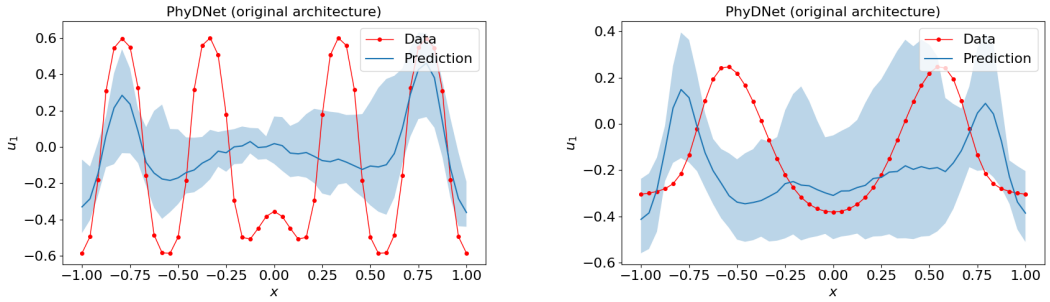


Figure 19: Prediction mean (with 95% confidence interval) of the activator in the diffusion-reaction equation at $t = 50$ (left, *in-dis-test*) and $t = 10$ (right, *out-dis-test* dataset) compared with the data obtained using PhyDNet with the original network size.

Table 12: MSE 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	9.3×10^{-5}	1.8×10^{-1}	7.6×10^{-2}
2	2.9×10^{-5}	6.5×10^{-2}	3.5×10^{-2}
3	2.2×10^{-5}	6.7×10^{-2}	4.0×10^{-2}
4	4.6×10^{-5}	7.2×10^{-2}	3.9×10^{-2}
5	3.5×10^{-5}	6.5×10^{-2}	8.8×10^{-2}
6	5.7×10^{-5}	1.2×10^{-1}	1.0×10^{-1}
7	3.5×10^{-5}	5.7×10^{-2}	9.0×10^{-2}
8	3.5×10^{-5}	5.9×10^{-2}	2.6×10^{-2}
9	2.3×10^{-5}	6.3×10^{-2}	4.1×10^{-2}
10	3.5×10^{-5}	6.1×10^{-2}	8.1×10^{-2}

A.6 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 13 for core samples #1, #2, and #2B.

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 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.

Table 13: 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				
Parameter	Unit	Core #1	Core #2	Core #2B
D	m^2/day	2.00×10^{-5}	2.00×10^{-5}	2.78×10^{-5}
ϕ	-	0.288	0.288	0.288
ρ_s	kg/m^3	1957	1957	1957
Simulation domain				
Parameter	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	1.01×10^{-4}	1.04×10^{-4}	N/A
u_s	kg/m^3	1.4	1.6	1.4