PAPM: A Physics-aware Proxy Model for Process Systems

Pengwei Liu [1] **Zhongkai Hao** [2] **Xingyu Ren** [1] **Hangjie Yuan** [1] **Jiayang Ren** [3] **Dong Ni** [1]

Abstract

In the context of proxy modeling for process systems, traditional data-driven deep learning approaches frequently encounter significant challenges, such as substantial training costs induced by large amounts of data, and limited generalization capabilities. As a promising alternative, physics-aware models incorporate partial physics knowledge to ameliorate these challenges. Although demonstrating efficacy, they fall short in terms of exploration depth and universality. To address these shortcomings, we introduce a **p**hysics-**a**ware **p**roxy **m**odel (**PAPM**) that fully incorporates partial prior physics of process systems, which includes multiple input conditions and the general form of conservation relations, resulting in better out-of-sample generalization. Additionally, PAPM contains a holistic temporal-spatial stepping module for flexible adaptation across various process systems. Through systematic comparisons with state-of-the-art pure data-driven and physics-aware models across five two-dimensional benchmarks in nine generalization tasks, PAPM notably achieves an average performance improvement of 6.7%, while requiring fewer FLOPs, and just 1% of the parameters compared to the prior leading method.

1. Introduction

From molecular dynamics to turbulent flows, process systems are essential in numerous scientific and engineering domains (Cameron & Hangos, 2001). Computational modeling and simulation are crucial for understanding their complex temporal-spatial dynamics, enabling precise predictions and informed decisions across various fields. However, these valuable insights are provided by traditional numerical simulations, which are often computationally intensive, especially in scenarios necessitating frequent model queries like reverse engineering forward simulation (Dijkstra & Luijten, 2021) and optimization design (Gramacy, 2020). Recent advancements in data-driven methods have paved the way to tackle computational challenges more effectively (Lu et al., 2019; Li et al., 2020; Kochkov et al., 2021; Stachenfeld et al., 2021; Hao et al., 2023b). As shown in Fig. 1 (left), these methods input multiple conditions of process systems to output time-dependent solutions, serving as the proxy model for process systems. Through adopting a supervised learning-from-data paradigm, remarkable advancements in calculation speeds have been achieved—several orders of magnitude faster than traditional numerical simulations. This has led to significant savings in computational costs. While data-driven methods are powerful, they face two primary limitations: **1)** a dependence on extensive labeled datasets, which contrasts sharply with the high computational costs of numerical simulations, and **2)** a presumption of train-test uniformity that leads to poor generalization, especially in out-of-sample scenarios like time extrapolation. This poor generalization arises from an overemphasis on the inductive biases of network architectures based on labels, rather than a strict adherence to fundamental physical laws (Li et al., 2021; Brandstetter et al., 2022).

To ameliorate high training costs and limited generalizability, a more promising strategy, known as physics-informed deep learning (PIDL), involves prior knowledge, such as fundamental physical laws, into neural networks (NNs). This integration enhances the sample efficiency and generalizability of NNs, proving particularly vital in scenarios with limited labeled data (Li et al., 2021; Hao et al., 2022; Meng et al., 2022; Cuomo et al., 2022). One of the typical methods considers complete physical laws as the loss function of NNs to construct proxy models, such as PINNs (Raissi et al., 2019) for specific conditions, PI-DeepONet (Wang et al., 2021), and PINO (Li et al., 2021) for multiple sets of conditions. However, the real-world applicability of these methods is limited by an incomplete understanding of the underlying physics of specific process systems, making it challenging to derive complete physics laws.

Another common approach, termed as "**physics-aware**" models, offers a promising solution for this scenario by incorporating partial prior knowledge alongside a small

[1]Zhejiang University, Hangzhou, Zhejiang, China [2]Tsinghua University, Beijing, China [3]University of British Columbia, Vancouver, BC, Canada. Correspondence to: Dong Ni <dni@zju.edu.cn>.

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

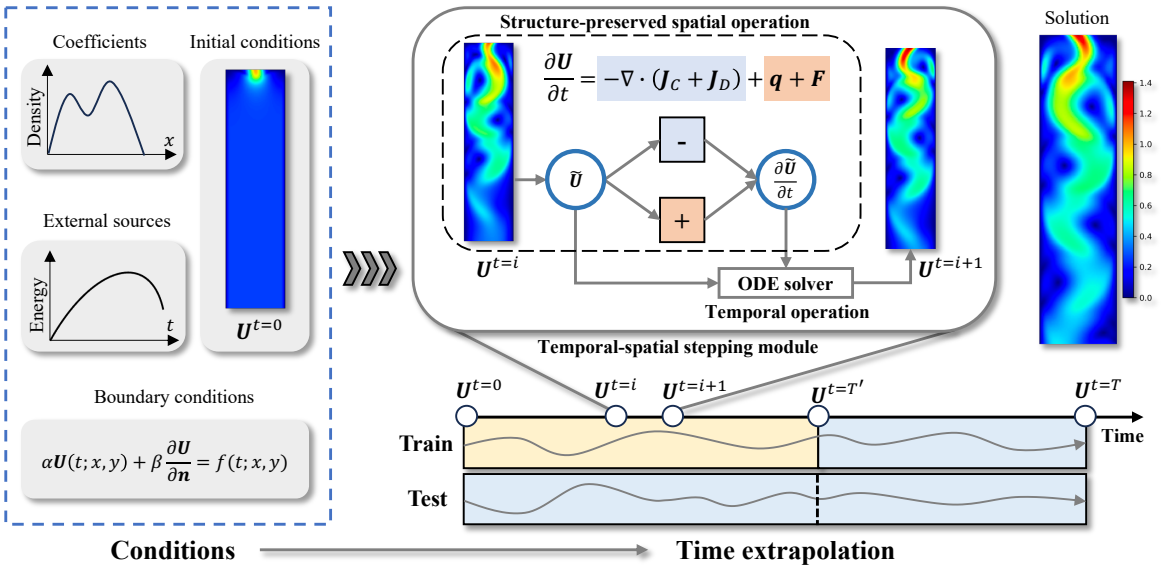

Figure 1. Overview of the PAPM's pipeline. The model takes the multiple conditions of process systems for time extrapolation and outputs solutions at an arbitrary time point. The core is the temporal-spatial stepping module (TSSM) ($\boldsymbol{U}^{t=i} \rightarrow \boldsymbol{U}^{t=i+1}$). Spatially, a structure-preserved operation aligns with the specific equation characteristics of different process systems. Temporally, it utilizes a continuous-time modeling framework through an ODE solver.

amount of labeled data. Considering the relationship between equations and their numerical schemes, the physics-aware methods convert partial prior physics laws into the corresponding numerical schemes, and embeddings them into the network structure (Long et al., 2018; Seo et al., 2020; Huang et al., 2023b; Akhare et al., 2023; Rao et al., 2023; Huang et al., 2023a; Kochkov et al., 2023; Pestourie et al., 2023; Liu et al., 2024). With these partial prior physics laws, physics-aware models only need a small number of labels to obtain excellent out-of-sample generalization performance. However, these methods focus primarily on spatial derivatives, and often neglect integral aspects such as conservation laws and constitutive relations. Consequently, they do not fully leverage prior physics knowledge, leading to unreliable solutions. Besides, these methods are generally tailored for a specific process system with limited universality.

Recognizing that process system modeling often requires the incorporation of conservation relations grounded in diffusion, convection, and source flows, this work aims to integrate this general physics law into the network architecture. By reinforcing the inductive biases in this manner, we can achieve better out-of-sample generalization. Additionally, as different process systems correspond to specific conservation or constitutive equations based on inherent system characteristics (Cameron & Hangos, 2001; Takamoto et al., 2022; Hao et al., 2023a), it is beneficial to identify both similarities and differences among process systems. Such an approach can offer a general temporal-spatial stepping module to combine various process systems flexibly.

As illustrated in Fig. 1, we propose a **p**hysics-**a**ware **p**roxy **m**odel (**PAPM**) for process systems, which incorporates multiple conditions to output solutions at arbitrary time points. PAPM fully utilizes partial prior knowledge, including multiple conditions, and the general form of conservation relations, alongside a small amount of label data, to model the dynamics of systems through the proposed temporal-spatial stepping module ($\boldsymbol{U}^{t=i} \rightarrow \boldsymbol{U}^{t=i+1}$). Notably, PAPM leverages the direction of data flow based on this general form, a distinction often overlooked by other physics-aware methods. Furthermore, PAPM focuses on out-of-sample scenarios, such as time extrapolation, aligning with the capabilities of alternative methods.

The core contributions of this work are:

- The proposal of PAPM, a versatile physics-aware architecture design that fully incorporates partial prior knowledge such as multiple conditions, and the general form of conservation relations. This design proves to be superior in both training efficiency and out-of-sample generalizability.
- The introduction of a holistic temporal-spatial stepping module (TSSM) for flexible adaptation across various process systems. It aligns with the distinct equation characteristics of different process systems by employing stepping schemes via temporal and spatial operations, whether in physical or spectral space.
- A systematic evaluation of state-of-the-art pure data-driven models alongside physics-aware models, spanning five two-dimensional non-trivial benchmarks. Notably,

PAPM achieved an average absolute performance boost of 6.7% with fewer FLOPs and only 1%-10% of the parameters compared to alternative methods.

2. Related Work

Pure Data-driven Method. There are various neural network designs, where CNNs (Yu et al., 2017; Bhatnagar et al., 2019; Stachenfeld et al., 2021) and GNNs (Sanchez-Gonzalez et al., 2020; Li & Farimani, 2022) target spatial dynamics within mesh grids, while RNN (Kochkov et al., 2021) and LSTM (Shi et al., 2015; Zhang et al., 2020) focus on temporal progression. Another line is the neural operator, excelling in mapping between temporal-spatial functional spaces, demonstrating success across various PDEs. Fourier neural operator (FNO) (Li et al., 2020) learns the operator by harnessing the spectral domain alongside the Fast Fourier Transform. DeepONet (Lu et al., 2019) approximates various nonlinear operators by leveraging branch and trunk networks for input functions and query points. Building upon this, MIONet (Jin et al., 2022) addresses the challenges of multiple input functions within the DeepONet framework. Moreover, U-FNets (Gupta & Brandstetter, 2022) and convolutional neural operators (CNO) (Raonić et al., 2023) are modified U-Net (Ronneberger et al., 2015) variants, where the former replace U-Net's layers by FNO's Fourier blocks, and the latter replace them by predefined convolutional block.

Physics-aware Method. Contrary to the method of integrating complete physics knowledge into its loss function, the physics-aware method only leverages labels while explicitly incorporating either entire or partial mechanistic knowledge into the network architecture. Inspired by the finite volume method, FINN (Karlbauer et al., 2022) innovatively employs flux and state kernels for modeling components of advection-diffusion equations in each volume, which is one class of process systems, and has an explicit form. Since FINN is conducted for each volume, FINN is computationally inefficient in the whole spatial region. PiNDiff (Akhare et al., 2023), PeRCNN (Rao et al., 2023), and PPNN (Liu et al., 2024) are inspired by the finite difference method. PiNDiff (Akhare et al., 2023) and NeuralGCM (Kochkov et al., 2023) integrate partial physics knowledge into the NN block for forecasting the systems' future states, ensuring mathematical integrity via differentiable programming. PeRCNN (Rao et al., 2023) employs convolutional operations to approximate unknown nonlinear terms for reaction-diffusion systems while incorporating known terms through difference schemes. PPNN (Liu et al., 2024) bakes prior-knowledge terms from low-resolution data, estimates unknown parts with the trainable network, and uses the Euler time-stepping difference scheme to form a regression model for updating states.

Learned correction methods. As highlighted in (Rackauckas et al., 2020; Um et al., 2020; Dresdner et al., 2022; Sun et al., 2023; McGreivy & Hakim, 2023), the core idea of learned correction methods is to approximate the known part with specific fixed modules (such as numerical methods) and approximate the unknown part with a neural network, which often yields superior results compared to fully learned approaches. However, the current learned correction methods typically rely on known equations and are optimized for specific conditions, which is somewhat different from our model's broader objective of generalizing across various conditions and conservation relations. The second difference between our work and the mentioned literature lies in the precision of the known parts. Specifically, PAPM only knows that the different terms in the equation adhere to the general form of conservation relations, without exact knowledge of each term's specific composition. In contrast, the literature deals with cases where the known parts are precise.

3. Preliminaries

This section presents the foundational description of process systems, known as the process model. Additionally, further clarification is provided on the specific problem in this work.

Process Model. Pivotal in engineering disciplines, process models represent and predict the dynamics of diverse process systems. This model's mathematical foundation relies on two essential sets of equations: conservation equations, governing the dynamic behavior of fundamental quantities, and constitutive equations, which describe the interactions among different variables. Further details are provided in Appendix B.

Eq. 1 and Eq. 2 represent the universal conservation and constitutive equations, respectively.

$$\begin{cases} \dfrac{\partial \boldsymbol{U}^t}{\partial t} = -\nabla \cdot (\boldsymbol{J}_C + \boldsymbol{J}_D) + \boldsymbol{q} + \boldsymbol{F} \\ \boldsymbol{J_C} = \boldsymbol{U}^t \cdot \boldsymbol{v}, \quad \boldsymbol{J_D} = -\boldsymbol{D} \cdot \nabla \boldsymbol{U}^t \end{cases} \tag{1}$$

$$\begin{cases} \boldsymbol{v} = v(\boldsymbol{U}^t), \quad \boldsymbol{D} = \boldsymbol{\lambda}, \\ \boldsymbol{q} = h_O(\boldsymbol{U}^t), \quad \boldsymbol{F} = h_F\left(\boldsymbol{X}_F\right) \end{cases} \tag{2}$$

where $\boldsymbol{U}$ is the physical quantity, denoting the system's state. Eq. 1 comprises four essential elements: the diffusion flows $\boldsymbol{J}_D$, convection flows $\boldsymbol{J}_C$, the internal source $\boldsymbol{q}$, and the external source $\boldsymbol{F}$. In Eq. 2, $\boldsymbol{v}$ denotes the velocity of the physical quantity being transmitted, $\boldsymbol{D}$ is the diffusion coefficient, $\boldsymbol{\lambda}$ denotes the coefficients, and $\boldsymbol{X}_F$ is the input of the external source term. Here, the corresponding linear or nonlinear mapping is the v, h_O, and h_F.

The structure of PAPM is depicted in Fig. 2 at time t ($\boldsymbol{U}^t \rightarrow \boldsymbol{U}^{t+1}$). Our goal is to use partial prior knowledge (the general form of Eq. 1) and a small amount of label data to

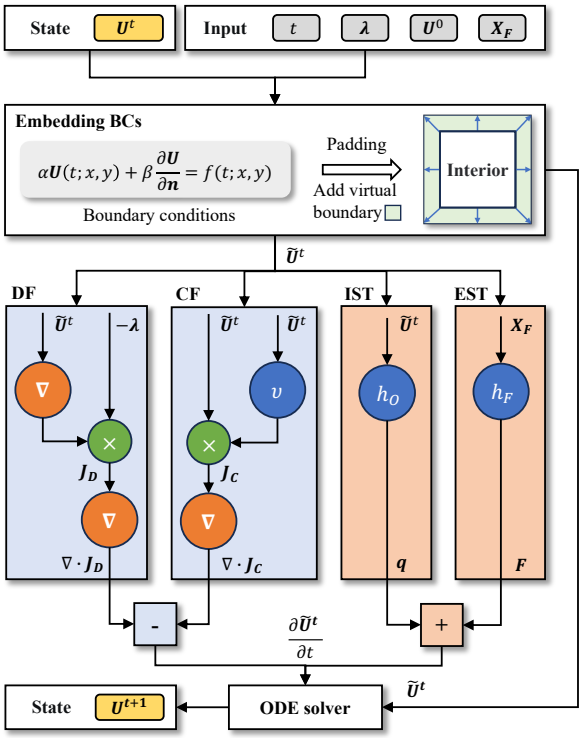

Figure 2. A detailed structure of PAPM at time t. Here, v, h_O, and h_F are the corresponding unknown mapping, and the neural networks are needed for learning. We propose a temporal-spatial stepping module (TSSM) for DF, CF, IST, and EST in section 4.2, which aligns with the distinct equation characteristics of different process systems.

establish a proxy model, which takes these various input conditions and outputs system time-dependent solutions ($\{\boldsymbol{U}^t\}_{1\le t\le T}$), as shown in Fig 1. Notably, the general form of Eq. 1 is only known, which is also the data flow, while the specific item is unknown, such as the mappings of v, h_O, and h_F, aligning with most real-world scenarios (Karlbauer et al., 2022; Huang et al., 2023a; Liu et al., 2024).

Problem Formulation. Under different initial and boundary conditions, external sources, and coefficients, the following T-step trajectory should be predicted. Moreover, due to the high cost of generating labeled data, we focused on out-of-sample scenarios, *e.g.* time extrapolation, where the training dataset only contains the subsequent T'-step trajectory, and $1 \le T' \ll T$.

Formally, the dataset $\mathcal{D} = \{(\boldsymbol{a}_k, \boldsymbol{S}_k)\}_{1\le k\le D}$, where $\boldsymbol{S}_k = \mathcal{G}(\boldsymbol{a}_k)$. Here $\boldsymbol{a}_k$ contains a set of inputs, that is, initial condition $\boldsymbol{U}_k^0$, boundary conditions, such as Robin conditions, external sources $\boldsymbol{X}_F$, and coefficients $\boldsymbol{\lambda}$. $\boldsymbol{S}_k = (\boldsymbol{U}_k^1, \cdots, \boldsymbol{U}_k^T)$ is the following trajectory, and the mapping $\mathcal{G}(\cdot)$ is our goal to learn. Each $\boldsymbol{U}_k^t = \{\boldsymbol{u}_{k,i}^t\}_{1\le i\le m}$ is a vector, which consists of $m \in \mathbb{N}^+$ physical quantities, such as velocity, vorticity, pressure, and temperature. We discretize each quantity $\boldsymbol{u}_{k,i}^t \in \mathcal{R}^N$ on the grid $\{\boldsymbol{x}_j \in \Omega\}_{1\le j\le N}$. In a nutshell, for modeling this operator $\mathcal{G}(\cdot)$, we use a parameterized neural network $\hat{\mathcal{G}}_\theta$ with parameters θ, inputs $\boldsymbol{a}_k$, and outputs $\hat{\mathcal{G}}_\theta(\boldsymbol{a}_k) = \hat{\boldsymbol{S}}_k$, where $1 \le k \le D$.

Our goal is to minimize the L_2 relative error loss between the prediction $\hat{\boldsymbol{S}}_k$ and real data $\boldsymbol{S}_k$ in the training dataset as,

$$\begin{aligned}\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \frac{1}{D_0}\sum_{k=1}^{D_0}\mathcal{L}_k(\boldsymbol{\theta}) &= \min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \frac{1}{D_0}\sum_{k=1}^{D_0}\frac{1}{T'}\sum_{t=1}^{T'}\mathcal{L}_k(t,\boldsymbol{\theta}) \\ &= \min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \frac{1}{D_0}\sum_{k=1}^{D_0}\frac{1}{T'}\sum_{t=1}^{T'}\frac{1}{m}\sum_{i=1}^{m}\frac{\| \boldsymbol{u}_{k,i}^t - \hat{\boldsymbol{u}}_{k,i}^t \|_2}{\| \boldsymbol{u}_{k,i}^t \|_2}\end{aligned} \tag{3}$$

where T' is the training time-step size, and D_0 is the size of the training dataset. $\mathcal{L}_k(t,\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\frac{\|\boldsymbol{u}_{k,i}^t - \hat{\boldsymbol{u}}_{k,i}^t\|_2}{\|\boldsymbol{u}_{k,i}^t\|_2}$ is the mean L_2 relative error loss at time t of index k. $\boldsymbol{\theta}$ is a set of the network parameters and $\boldsymbol{\Theta}$ is the parameter space.

4. Methodology

This section presents PAPM's architecture specifically tailored to conservation and constitutive relations. Then, a holistic temporal-spatial stepping module (TSSM) is proposed, adapting to the unique equation characteristics of various process systems. The Appendix D.1 provides the pseudo-code for the entire training process, offering a comprehensive understanding of our approach.

4.1. PAPM Overview

Aligning with the general form of Eq. 1 and Eq. 2, there are four elements corresponding to Diffusive Flows (DF), Convective Flows (CF), Internal Source Term (IST), and External Source Term (EST) in PAPM's structure diagram, as illustrated in Fig. 2. The versatile general structure of PAPM could enable it to work effectively across different process systems. The input contains a set of inputs, that is, coefficient $\boldsymbol{\lambda}$, initial state $\boldsymbol{U}^0$, external source input $\boldsymbol{X}_F$, and boundary conditions, which are multiple conditions of process systems. The sequence of embedding this prior knowledge unfolds as follows:

1) Embedding BCs. Using the given boundary conditions, the physical quantity $\boldsymbol{U}^t$ is updated, yielding $\tilde{\boldsymbol{U}}^t$. A padding strategy is employed to integrate four different boundary conditions in four different directions into PAPM. Further details are provided in Appendix C.

2) Diffusive Flows (DF). Using $\tilde{\boldsymbol{U}}^t$ and coefficients $\boldsymbol{\lambda}$, we represent the directionless diffusive flow. The diffusion flow and its gradient are obtained as $\boldsymbol{J}_D = -\boldsymbol{D}\cdot\nabla\tilde{\boldsymbol{U}}^t$ and $\nabla\cdot\boldsymbol{J}_D$ via a symmetric gradient operator, respectively.

3) Convective Flows (CF). The pattern v is derived from $\tilde{\boldsymbol{U}}^t$. Once $\boldsymbol{v}$ is determined, its sign indicates the direction of the flows, enabling computation of $\boldsymbol{J}_C = \tilde{\boldsymbol{U}}^t \cdot \boldsymbol{v}$ and $\nabla\cdot\boldsymbol{J}_C$

Table 1. Temporal-Spatial Stepping Module (TSSM).

Category		Localized Fig.3 (Left and Mid), Alg. 1	Spectral Fig.3 (Right), Alg. 2	Hybrid
Characteristic		Explicit	Implicit	Explicit+Implicit
Example		$-u\nabla u + \nabla^2 u$	$-u\nabla w + \nabla^2 w$	$-u\nabla u + \nabla^2 u - \nabla p$
Temporal	ODE solver	Neural ODE		
Spatial	DF CF	Pre-defined convolution	E-Conv	Pre-defined convolution
	IST/EST	ResNet block	S-Conv block	ResNet/S-Conv block

through a directional gradient operator.

4) Internal Source Term (IST) & External Source Term (EST). Generally, IST and EST present a complex interplay between physical quantities $\tilde{\boldsymbol{U}}^t$ and external inputs $\boldsymbol{X}_F$. Often, this part in real systems doesn't have a clear physics-based relation, prompting the use of NNs to capture this intricate relationship.

5) ODE solver. From DF, CF, IST, and EST, the dynamic $\partial\tilde{\boldsymbol{U}}^t/\partial t$ are derived. By doing so, the Eq. 1 can be reduced to an ordinary differential equation (ODE), and the ODE solver is used to approximate the evolving state as $\boldsymbol{U}^{t+1} = \tilde{\boldsymbol{U}}^t + \int_t^{t+1} \frac{\partial\tilde{\boldsymbol{U}}^t}{\partial t} dt$.

Fig. 2 above illustrates the data flow at time t. Then, during the training or inference phase, PAPM performs autoregressive predictions as $\boldsymbol{U}^{t+1} = \mathcal{G}_\theta(\boldsymbol{U}^t, a_k)$, where $1 \le t \le \tau$, with $\tau = T'$ during training and $\tau = T$ during inference.

In short, PAPM takes different conditions, including initial conditions, boundary conditions, external sources, and coefficients, and interactively propagates the dynamics of process systems forward using five distinct components. The purpose of such a structured design is to reinforce the inductive biases concerning strict physical laws.

4.2. Temporal-Spatial Stepping Module (TSSM)

Due to the diversity of process systems, we develop a holistic Temporal-Spatial Stepping Module (TSSM) to align with the unique characteristics of different process systems, which forms the specific network structure for each component in PAPM. As shown in Tab. 1, TSSM is categorized into three types based on structures of process systems, where each type decomposes temporal and spatial components, *i.e.*, structure-preserved localized operator, spectral operator, and hybrid operator. Notably, all three approaches can employ a common temporal operation through ODE solvers.

Temporal Operation. After obtaining the dynamic state derivative, $\partial\tilde{\boldsymbol{U}}^t/\partial t$, the subsequent state $\boldsymbol{U}^{t+1}$ can be computed through numerical integration over different time spans. Due to the numerical instability associated with first-order explicit methods like the Euler method (Gottlieb et al., 2001; Fatunla, 2014), we adopt the neural ordinary differential equation approach (Neural ODE (Chen et al., 2018)), which employs the Runge–Kutta time-stepping strategy to enhance stability. The computed state $\boldsymbol{U}^{t+1}$ is then recursively fed back into the network as the input for the subsequent time step, continuing this process until the final time step is reached.

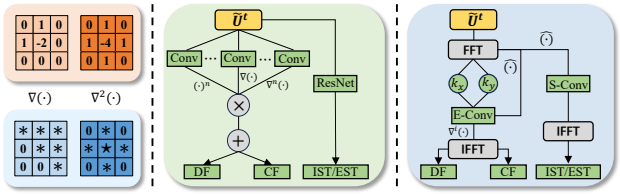

Figure 3. **Left:** Pre-defined convolutional kernels, where fixed and trainable correspond to the matrices at the top and bottom, respectively. The bottom kernels approximate the unidirectional convection (upwind scheme) and directionless diffusion (central scheme). Symbols $*$ and $\star$ indicate trainable parameters corresponding to the upper triangular and symmetric matrices. **Mid:** Structure-preserved localized operator. **Right:** Structure-preserved spatial operator.

Structure-preserved Localized Operator. For systems with explicit structures, such as the Burgers and RD equations, typified by expressions like $-u\nabla u + \nabla^2 u$, convolutional kernels in the physical space are employed to capture system dynamics. We opt for either fixed or trainable kernels, illustrated in Fig. 3 (Left), depending on our understanding of the system. Specifically, the fixed one is based on the predefined convolution kernel derived from difference schemes, and further details are provided in Appendix D.2.1. Moreover, the trainable version tailors its design to essential features of convection (upper or lower triangular) and diffusion (symmetric). Once set, the localized operator is depicted in Fig. 3 (Mid). We could represent nonlinear terms in DF and CF using predefined convolution kernels alongside partially known physics (the general form of Eq. 1 and Eq. 2). Any unknown specific terms, such as source, are then addressed through a shallow ResNet block.

Structure-preserved Spectral Operator. For systems with implicit structures, such as the Navier-Stokes Equation in vorticity form, represented like $-u\nabla w + \nabla^2 w$, we adopt a sequential process, as shown in Fig. 3(Right). Recognizing the implicit linkage between velocity and vorticity, w is initially processed to extract the flow function, subsequently leading to the velocity derivation. The spectral space dimensions (k_x and k_y) and spectral quantity (denoted as $\hat{\cdot}$), are obtained by leveraging the FFT. Associating k_x and k_y with $\hat{\cdot}$, differential operators like $\nabla^i(\cdot)$ are represented via E-Conv (*e.g.*, element-wise product), and then mapped back to the physical space using IFFT. Doing so can represent the nonlinear terms in DF and CF via simple computations such as addition and multiplication in the spectral domain. Moreover, the spectral convolution (S-Conv) fo FNO (Li et al., 2020) is introduced to learn unknown components. This process can be further detailed in Appendix D.2.2.

Structure-preserved Hybrid Operator. For systems with a hybrid structure, such as the Navier-Stokes Equation in general form (*e.g.*, $-u\nabla u + \nabla^2 u - \nabla p$), given the implicit interrelation between pressure p and velocity u, a combination of the method above is employed. Explicit constituents, such as $u\nabla u$ and $\nabla^2 u$, are addressed through the localized operator. Meanwhile, implicit relations are resolved similarly by the spectral operator. For unknown components, either of the two operators can be engaged. We generally favor the localized operator as it allows direct operations without requiring transitions between different spaces.

5. Experiments

5.1. Experimental setup and evaluation protocol

Datasets. We employ five datasets spanning diverse domains, such as fluid dynamics and heat conduction, detailed in Appendix E. These datasets are categorized based on the Temporal-Spatial Stepping Module used in PAPM to highlight distinct equation characteristics in various process systems.

- **Localized Category: Burgers2d (Huang et al., 2023a)** is a 2D benchmark PDE with periodic boundary conditions, various initial conditions, and viscosity, while the source remains unknown. **RD2d (Takamoto et al., 2022)** addresses a 2D FitzHugh-Nagumo reaction-diffusion equation with no-flow Neumann boundary condition, diverse initial conditions, and unknown source terms.
- **Spectral Category: NS2d (Li et al., 2020)** explores incompressible fluid dynamics in vorticity form with varied initial conditions and unknown sources.
- **Hybrid Category: Lid2d** focuses on incompressible lid-driven cavity flow, characterized by differing viscosity and BCs. **NSM2d** deals with incompressible fluid dynamics within an unknown magnetic field source, featuring time-varying BCs, various initial conditions, and viscosity.

Baselines. We compared our approach with eight SOTA baselines for a comprehensive evaluation. **ConvLSTM** (Shi et al., 2015) is a classical time series modeling technique that captures dynamics via CNN and LSTM. **DilResNet** (Stachenfeld et al., 2021) adopts the encoder-process-decoder process with dilated-ConvResNet for dynamic data through an autoregressive stepping manner. **time-FNO2D** (Li et al., 2020) and **MIONet** (Jin et al., 2022) are two typical neural operators in learning dynamics. **U-FNet** (Gupta & Brandstetter, 2022) and **CNO** (Raonić et al., 2023) are modified U-Net (Ronneberger et al., 2015) variants. **PeRCNN** (Rao et al., 2023) incorporates specific physical structures into a neural network, ideal for sparse data scenarios. **PPNN** (Liu et al., 2024) is a novel autoregressive framework preserving known PDEs using multi-resolution convolutional blocks.

Metrics. According to Eq. 3, we adopt the mean L_2 relative error, abbreviated as ϵ, as our evaluation metric for validation and testing datasets. ϵ is formulated as follows:

$$\epsilon = \frac{1}{D}\sum_{k=1}^{D}\frac{1}{T}\sum_{t=1}^{T}\frac{1}{m}\sum_{i=1}^{m}\frac{\| \boldsymbol{u}_{k,i}^t - \hat{\boldsymbol{u}}_{k,i}^t \|_2}{\| \boldsymbol{u}_{k,i}^t \|_2} \tag{4}$$

Evaluation Protocol. Based on the experimental setting of time extrapolation, we further conducted experiments in the following two parts: coefficient interpolation (referred to as **C Int.**) and coefficient extrapolation (referred to as **C Ext.**). More information about the evaluation protocol, the hyper-parameters of baselines, and our methods can be further detailed in Appendix F.

5.2. Main Results

Performance Comparisons. Tab. 2 and Tab. 3 present the primary experimental outcomes, the number of trainable parameters (N_P), and computational cost (FLOPs) for each baseline across datasets [1].

Here, **Bold** and Underline indicate the best and second best performance, respectively. Notably, lower results mean better performance because the metric is the mean L_2 relative error. Our observations from the results are as follows:

Firstly, PAPM exhibits the most balanced trade-off between performance, parameter count, and computational cost among all methods evaluated, from explicit structures (Burgers2d, RD2d) to implicit (NS2d) and more complex hybrid structures (Lid2d, NSM2d). Remarkably, even though PAPM requires significantly fewer FLOPs and only 1%

[1] Code is available at https://github.com/pengwei07/PAPM.

Table 2. ϵ (Eq. 4) across different datasets in time extrapolation task.

Config	Burgers2d		RD2d	NS2d			Lid2d	NSM2d	
	C Int.	C Ext.	C Int.	ν=1e-3	ν=1e-4	ν=1e-5	C Ext.	C Int.	C Ext.
ConvLSTM	0.314	0.551	0.815	0.781	0.877	0.788	1.323	0.910	1.102
Dil-ResNet	0.071	0.136	0.021	0.152	0.511	0.199	0.261	0.288	0.314
time-FNO2D	0.173	0.233	0.333	0.118	0.100	0.033	0.265	0.341	0.443
MIONet	0.181	0.212	0.247	0.139	0.114	0.051	0.221	0.268	0.440
U-FNet	0.109	0.433	0.239	0.191	0.190	0.256	0.192	0.257	0.457
CNO	0.112	0.126	0.258	0.125	0.148	**0.030**	0.218	0.197	0.355
PeRCNN	0.212	0.282	0.773	0.571	0.591	0.275	0.534	0.493	0.493
PPNN	0.047	0.132	0.030	0.365	0.357	0.046	0.163	0.206	0.264
PAPM (Our)	**0.039**	**0.101**	**0.018**	**0.110**	**0.097**	0.034	**0.160**	**0.189**	**0.245**

Table 3. Comparison of parameters and FLOPs.

Config	N_P			FLOPs/M
	Localized/M	Spectra/M	Hybrid/M	
ConvLSTM	0.175	0.139	0.211	32.75
Dil-ResNet	0.150	0.148	0.152	62.40
time-FNO2D	0.464	0.463	0.464	6.88
MIONet	0.261	0.261	0.261	10.01
U-FNet	9.853	9.851	9.854	559.89
CNO	2.606	2.600	2.612	835.37
PeRCNN	**0.001**	**0.001**	**0.001**	3.44
PPNN	1.201	1.190	1.213	348.56
PAPM	0.014	0.034	0.035	**1.23**

of the parameters employed by the prior leading method, PPNN, it still outperforms it by a large margin. In a nutshell, our model enhances the performance by an average of 6.7% over nine tasks, affirming PAPM as a versatile and efficient framework suitable for diverse process systems.

Secondly, PAPM's structured treatment of system inputs and states leads to a remarkable 9.6% performance boost in three coefficient-extrapolation tasks. This highlights its superior generalization capability in out-of-sample scenarios. Unlike models like PPNN, which directly use system-specific inputs, PAPM integrates coefficient data more intricately within conservation and constitutive relations, boosting its adaptability to varying coefficients.

Thirdly, data-driven methods are less effective than physics-aware methods like PPNN and our PAPM in time extrapolation tasks, where incorporating prior physical knowledge through structured network design enhances a model's generalization ability. Notably, PeRCNN uses 1×1 convolution to approximate nonlinear terms, but experimental results suggest limited performance. Further details are available in Appendix F.2.

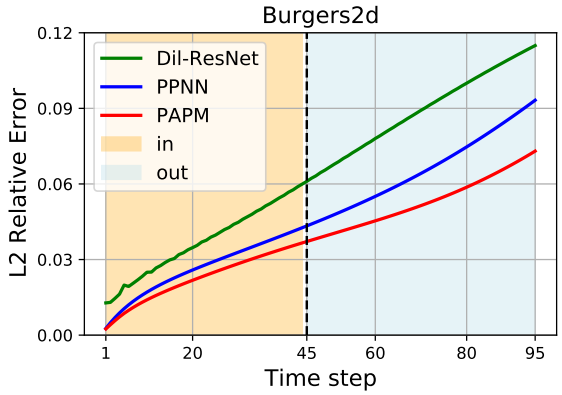

Figure 4. ϵ (Eq. 4) of predicted each time step on Burgers2d, where **in** is the same as the training, **out** is the time extrapolation.

Visualization. Fig. 4 showcases the stepwise relative error of PAPM during the extrapolation process in the test dataset, using Burgers2d's C Int. as a representative example. Compared to the two best-performing baselines, our model (depicted by the red line) exhibits superior performance throughout the extrapolation process, with the least error accumulation. Turning our attention to the more challenging NSM2d dataset, Fig. 5 presents the results across five extrapolation time slices. While FNO demonstrates commendable accuracy within the training domain ($T \leq \frac{1}{2}T_{end}$), its performance falters significantly outside of it ($\frac{1}{2}T_{end} < T \leq T_{end}$). On the other hand, physics-aware methods (PPNN), and PAPM in particular, consistently capture the evolving patterns with a greater degree of robustness. Notably, our method emerges as a leader in terms of precision. Additional visual results can be found in Appendix G.1.

5.3. Efficiency

Training and Inference Cost: Dataset generation for our work is notably resource-intensive, with inference times ranging from 10^2 to 10^4 seconds for public datasets, and up to 10^3 seconds for those datasets we generated using COMSOL Multiphysics. In stark contrast, both baselines and PAPM register inference times between 0.1 to 10 seconds (detailed in Appendix G.2), achieving an improvement of 3 to 5 orders of magnitude. Notably, PAPM's time cost rivals or even surpasses baselines across different datasets. PAPM's efficiency remains competitive with other data-driven methods.

Data Efficiency. Owing to PAPM's structured design, data utilization is significantly enhanced. To evaluate data efficiency, we conducted tests using RD2d as a representative example, with Dil-ResNet and PPNN symbolizing pure data-driven and physics-aware methods. The results, displayed in Fig. 6, depict PAPM's efficiency concerning data volume and label data step size in training.

(1) Amount of Data: With a fixed 20% reserved for the test set, the remaining 80% of the total data is allocated to the training set. We systematically varied the training data volume, ranging from initially utilizing only 5% of the training set and progressively increasing it to the entire 100%. PAPM's relative error distinctly outperforms other

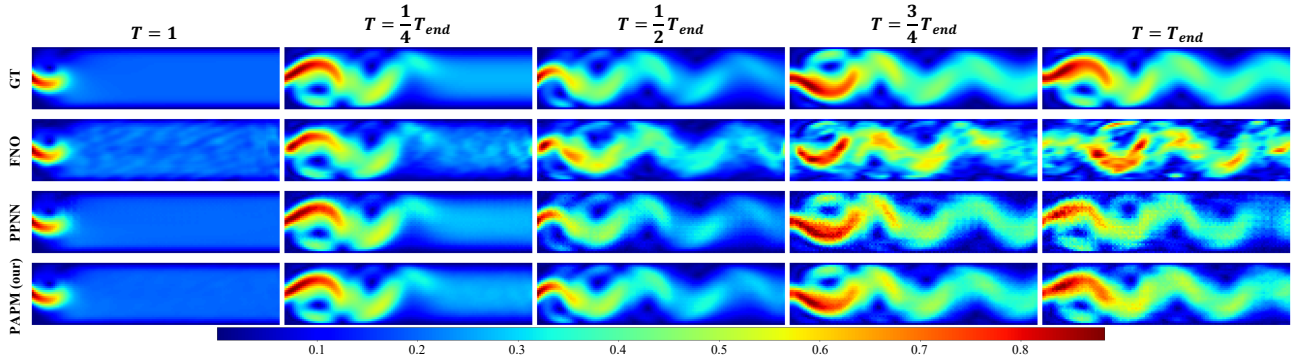

Figure 5. Predicted flow velocity ($|\boldsymbol{u}|_2$) snapshots by FNO, PPNN, and PAPM (Ours) vs. Ground Truth (GT) on NSM2d dataset in T Ext. task.

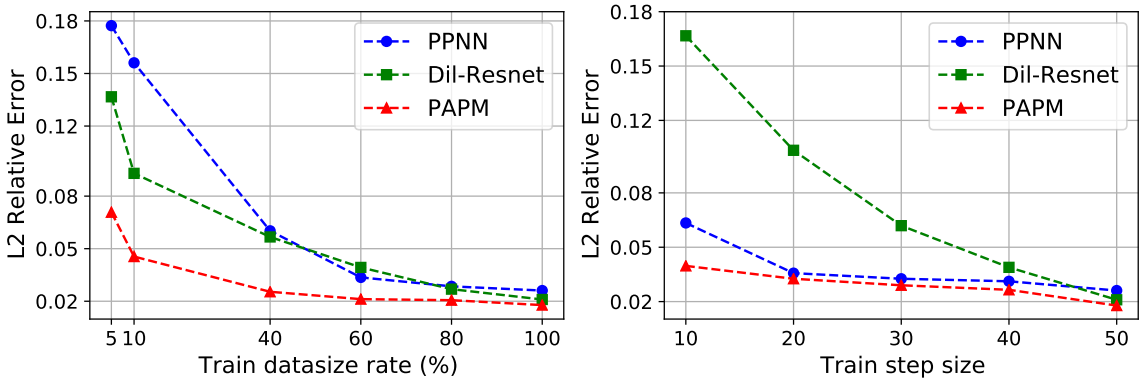

Figure 6. ϵ (Eq. 4) comparison by the leading method, PPNN, Dil-Resnet, PAPM (Ours) on the RD2d dataset. Left: varying amount of training data. Right: varying train step size.

baselines, especially with limited data (5%). As depicted in Fig. 6 (Left), PAPM's error consistently surpasses other methods, stabilizing below 2% as the training data volume increases.

(2) Time Step Size: We varied the time step size from a tenth to half of the total duration, increasing it in increments of tenths. As shown in Fig. 6 (Right), PAPM demonstrates the capability for long-range time extrapolation with fewer dynamic steps. It consistently outperforms other methods, achieving superior results even with shorter training time step sizes.

5.4. Ablation Studies

Different blocks impacts. Tab. 4 displays our selection of the Burger2d dataset for ablation studies, chosen for its representation of diffusion, convection, and source terms. We defined several configurations to assess the impact of individual components. The L_2 relative error on the boundary (BC ϵ) is introduced to highlight the importance of physics embedding further. **no_DF** excludes diffusion, whereas **no_CF** omits convection. In **no_Phy**, we retain only a structure with a residual connection, thereby eliminating both diffusion and convection. **no_BCs** setup removes the explicit embedding of boundary conditions, **no_NODE** replaces the Neural ODE with the Euler stepping scheme, and **no_All** adopts a purely data-driven approach. Additional ablation results can be found in Appendix G.3.

Key findings include the crucial roles of diffusion and convection in representing system dynamics, as evidenced in the no_DF, no_CF, and no_Phy configurations. Specifically, the no_DF configuration demonstrated the importance of integrating the viscosity coefficient with the diffusion term, with its absence leading to significant errors. The necessity of adhering to physical laws in boundary conditions was highlighted in the no_BCs, notably reducing errors on the boundary (BC ϵ). Lastly, the no_NODE results indicate that different temporal stepping schemes significantly impact the outcomes, underscoring the effectiveness of neural ODEs in continuous-time modeling.

Different blocks validations. Taking the Burgers2d and

Table 4. Ablation comparison of ϵ and ϵ on the boundary (BC ϵ).

Config	C Int.		C Ext.	
	ϵ	BC ϵ	ϵ	BC ϵ
no_DF	0.067	0.051	0.207	0.067
no_CF	0.062	0.043	0.131	0.054
no_Phy	0.149	0.051	0.210	0.144
no_BCs	0.068	0.097	0.136	0.193
no_NODE	0.053	0.041	0.150	0.045
no_All	0.162	0.195	0.216	0.250
PAPM	**0.039**	**0.037**	**0.101**	**0.043**

RD2d datasets as examples to demonstrate the fact that the convection/diffusion/source terms could actually learn those parts in the equations. We use high-fidelity FDM and FVM to compute the corresponding terms to obtain the detailed term for convection/diffusion/source terms. The L_2 relative error between ground truth and numerical results are 0.0041 and 0.0032 in all time steps for Burgers2d and RD2d, respectively. Thus, we can use the results obtained by the numerical methods as reference values to verify this. As shown in Tab. 5, the different terms of PAPM can be used to learn the equation's convection/diffusion/source parts. Additional visualization results can be found in Appendix G.1 (Fig. 9 and Fig. 10).

Table 5. Comparison of ϵ for different blocks on different datasets.

Datasets	ϵ	convection ϵ	diffusion ϵ	source ϵ
Burgers2d	**0.039**	0.037	0.041	0.069
RD2d	**0.018**	-	0.025	0.012

6. Conclusion

To address the challenges of physics-aware models falling short regarding exploration depth and universality, we have proposed PAPM. It fully incorporates partial prior physics of process systems, which includes multiple input conditions and the general form of conservation relations, resulting in better training efficiency and out-of-sample generalization. Additionally, PAPM contains a holistic temporal-spatial stepping module for flexible adaptation across various process systems. The efficacy of PAPM's structured design and holistic module was extensively validated across five datasets within distinct out-of-sample tasks. Notably, PAPM achieved an average performance boost of 6.7% with fewer FLOPs and only 1% of the parameters employed by the prior leading method. Through such analysis, PAPM exhibits the most balanced trade-off between accuracy and computational efficiency among all evaluated methods, alongside impressive out-of-sample generalization capabilities.

7. Limitation and Future Work

We aim to extend our model to more complex, industrially relevant systems, moving beyond 2D spatio-temporal dynamics to scenarios like 3D-industry-standard aerodynamics, plasma discharge, and multi-physics couplings (e.g., fluid-structure and thermal fluid-structure interactions). Despite our model's proven balance in accuracy and efficiency, we aim to challenge it further in these intricate environments. Additionally, we plan to adapt PAPM for irregular grid scenarios, typical in the industry, by integrating graph neural networks. This will enhance PAPM's versatility, allowing it to handle diverse data structures and complex dynamic processes such as convection, diffusion, and source interactions.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. High performance requires substantial resources, including computing power and time, to generate quality data. Acquiring such data can be costly or resource-intensive in real-world systems, leading to potential resource wastage. The ethical use of these resources, avoiding unnecessary environmental impact, and ensuring the cost-effectiveness of data acquisition are critical concerns. Therefore, we emphasize the importance of efficient and responsible data usage to minimize adverse societal and environmental effects.

Acknowledgment

This work was supported by the National Key Research and Development Program of China (No.2021YFF0500403).

References

Akhare, D., Luo, T., and Wang, J.-X. Physics-integrated neural differentiable (pindiff) model for composites manufacturing. *Computer Methods in Applied Mechanics and Engineering*, 406:115902, 2023.

Bhatnagar, S., Afshar, Y., Pan, S., Duraisamy, K., and Kaushik, S. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64:525–545, 2019.

Brandstetter, J., Worrall, D., and Welling, M. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

Cameron, I. T. and Hangos, K. *Process modelling and model analysis*. Elsevier, 2001.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud,

D. K. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., and Piccialli, F. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

Dijkstra, M. and Luijten, E. From predictive modelling to machine learning and reverse engineering of colloidal self-assembly. *Nature materials*, 20(6):762–773, 2021.

Dresdner, G., Kochkov, D., Norgaard, P., Zepeda-Núñez, L., Smith, J. A., Brenner, M. P., and Hoyer, S. Learning to correct spectral methods for simulating turbulent flows. *arXiv preprint arXiv:2207.00556*, 2022.

Fatunla, S. O. *Numerical methods for initial value problems in ordinary differential equations*. Academic Press, 2014.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Gottlieb, S., Shu, C.-W., and Tadmor, E. Strong stability-preserving high-order time discretization methods. *SIAM review*, 43(1):89–112, 2001.

Gramacy, R. B. *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. CRC press, 2020.

Gupta, J. K. and Brandstetter, J. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

Hao, Z., Liu, S., Zhang, Y., Ying, C., Feng, Y., Su, H., and Zhu, J. Physics-informed machine learning: A survey on problems, methods and applications. *arXiv preprint arXiv:2211.08064*, 2022.

Hao, Z., Yao, J., Su, C., Su, H., Wang, Z., Lu, F., Xia, Z., Zhang, Y., Liu, S., Lu, L., et al. Pinnacle: A comprehensive benchmark of physics-informed neural networks for solving pdes. *arXiv preprint arXiv:2306.08827*, 2023a.

Hao, Z., Ying, C., Wang, Z., Su, H., Dong, Y., Liu, S., Cheng, Z., Zhu, J., and Song, J. Gnot: A general neural operator transformer for operator learning. *arXiv preprint arXiv:2302.14376*, 2023b.

Huang, X., Li, Z., Liu, H., Wang, Z., Zhou, H., Dong, B., and Hua, B. Learning to simulate partially known spatiotemporal dynamics with trainable difference operators. *arXiv preprint arXiv:2307.14395*, 2023a.

Huang, X., Shi, W., Meng, Q., Wang, Y., Gao, X., Zhang, J., and Liu, T.-Y. Neuralstagger: accelerating physics-constrained neural pde solver with spatial-temporal decomposition. *arXiv preprint arXiv:2302.10255*, 2023b.

Jin, P., Meng, S., and Lu, L. Mionet: Learning multiple-input operators via tensor product. *SIAM Journal on Scientific Computing*, 44(6):A3490–A3514, 2022.

Karlbauer, M., Praditia, T., Otte, S., Oladyshkin, S., Nowak, W., and Butz, M. V. Composing partial differential equations with physics-aware neural networks. In *International Conference on Machine Learning*, pp. 10773–10801. PMLR, 2022.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Kochkov, D., Yuval, J., Langmore, I., Norgaard, P., Smith, J., Mooers, G., Klöwer, M., Lottes, J., Rasp, S., Düben, P., Hatfield, S., Battaglia, P., Sanchez-Gonzalez, A., Willson, M., Brenner, M. P., and Hoyer, S. Neural general circulation models for weather and climate. *arXiv preprint arXiv:2311.07222*, 2023.

Li, Z. and Farimani, A. B. Graph neural network-accelerated lagrangian fluid simulation. *Computers & Graphics*, 103: 201–211, 2022.

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*, 2020.

Li, Z., Zheng, H., Kovachki, N. B., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., and Anandkumar, A. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, 2021.

Liu, X.-Y., Zhu, M., Lu, L., Sun, H., and Wang, J.-X. Multi-resolution partial differential equations preserved learning framework for spatiotemporal dynamics. *Communications Physics*, 7(1):31, 2024.

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.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Lu, L., Jin, P., and Karniadakis, G. E. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E. Deepxde: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228, 2021.

McGreivy, N. and Hakim, A. Invariant preservation in machine learned pde solvers via error correction. *arXiv preprint arXiv:2303.16110*, 2023.

Meng, C., Seo, S., Cao, D., Griesemer, S., and Liu, Y. When physics meets machine learning: A survey of physics-informed machine learning. *arXiv preprint arXiv:2203.16797*, 2022.

Pestourie, R., Mroueh, Y., Rackauckas, C., Das, P., and Johnson, S. G. Physics-enhanced deep surrogates for partial differential equations. *Nature Machine Intelligence*, 5(12):1458–1465, 2023.

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., and Edelman, A. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. 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.

Rao, C., Ren, P., Wang, Q., Buyukozturk, O., Sun, H., and Liu, Y. Encoding physics to learn reaction–diffusion processes. *Nature Machine Intelligence*, pp. 1–15, 2023.

Raonić, B., Molinaro, R., Rohner, T., Mishra, S., and de Bezenac, E. Convolutional neural operators. *arXiv preprint arXiv:2302.01178*, 2023.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241. Springer, 2015.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pp. 8459–8468. PMLR, 2020.

Seo, S., Meng, C., Rambhatla, S., and Liu, Y. Physics-aware spatiotemporal modules with auxiliary tasks for meta-learning. *arXiv preprint arXiv:2006.08831*, 2020.

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-c. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28, 2015.

Stachenfeld, K., Fielding, D. B., Kochkov, D., Cranmer, M., Pfaff, T., Godwin, J., Cui, C., Ho, S., Battaglia, P., and Sanchez-Gonzalez, A. Learned coarse models for efficient turbulence simulation. *arXiv preprint arXiv:2112.15275*, 2021.

Sun, Z., Yang, Y., and Yoo, S. A neural pde solver with temporal stencil modeling. In *International Conference on Machine Learning*, pp. 33135–33155. PMLR, 2023.

Takamoto, M., Praditia, T., Leiteritz, R., MacKinlay, D., Alesiani, F., Pflüger, D., and Niepert, M. Pdebench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35: 1596–1611, 2022.

Um, K., Brand, R., Fei, Y. R., Holl, P., and Thuerey, N. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Advances in Neural Information Processing Systems*, 33:6111–6122, 2020.

Wang, S., Wang, H., and Perdikaris, P. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40): eabi8605, 2021.

Yu, F., Koltun, V., and Funkhouser, T. Dilated residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 472–480, 2017.

Zhang, R., Liu, Y., and Sun, H. Physics-informed multi-lstm networks for metamodeling of nonlinear structures. *Computer Methods in Applied Mechanics and Engineering*, 369:113226, 2020.

Table 6. Table of notations.

Notation	Meaning
Process model	
$\boldsymbol{U}$	the physical quantity, also as the system's state
$\boldsymbol{J}_D$	the diffusion flows in the conservation relations
$\boldsymbol{J}_C$	the convection flows in the conservation relations
$\boldsymbol{q}$	the internal source in the conservation relations
$\boldsymbol{F}$	the external source in the conservation relations
$\boldsymbol{v}$	the velocity of the physical quantity being transmitted
$\boldsymbol{D}$	the coefficients
$\boldsymbol{\lambda}$	the coefficients, such as viscosity
$\boldsymbol{X}_F$	a vector of external sources, such as voltages
$\boldsymbol{U}^0$	the initial condition
ICs	Initial conditions
BCs	Boundary conditions
Problem formulation	
$\boldsymbol{U}_k^t = \{\boldsymbol{u}_{k,i}^t\}_{1\leq i\leq m}$	a vector, which consists of $m \in \mathbb{N}^+$ physical quantities of index k at time t
$\boldsymbol{U}_k^0$	initial condition of index k at time $t = 0$
$\boldsymbol{u}_{k,i}^t \in \mathcal{R}^N$	a physical quantity, such as vorticity
$\{\boldsymbol{x}_j \in \Omega\}_{1\leq j\leq N}$	the grid (also as discretized spatial coordinates)
$\mathcal{A} = \{\boldsymbol{a}_k\}$	the input conditions space
$\boldsymbol{a}_k$	a set input conditions of index k, containing initial condition, and other conditions
$\mathcal{S} = \{\boldsymbol{S}_k\}$	the solutions space
$\boldsymbol{S}_k = (\boldsymbol{U}_k^1, \cdots, \boldsymbol{U}_k^T)$	a set output of index of index k
$\mathcal{D} = \{(\boldsymbol{a}_k, \boldsymbol{S}_k)\}_{1\leq k\leq D}$	the dataset, where $\boldsymbol{S}_k = \mathcal{G}(\boldsymbol{a}_k)$
$\mathcal{G} : \mathcal{A} \to \mathcal{S}$	the mapping of our goal to learn
$\mathcal{G}_\theta : \boldsymbol{a}_k \to \tilde{\boldsymbol{S}}_k, 1 \leq k \leq D$	a parameterized neural network with parameters $\theta \in \boldsymbol{\Theta}$
$\boldsymbol{\Theta}$	the parameter space
T	the time-step size for inference
T'	the time-step size of the training dataset, where $1 \leq T' \ll T$
Methodology	
$\tilde{\boldsymbol{U}}^t$	the updated physical quantity by using the given boundary conditions
DF	Diffusive Flows
CF	Convective Flows
IST	Internal Source Term
EST	External Source Term
TSSM	Temporal-Spatial Stepping Module
k_x, k_y	the spectral space dimensions
Experiments	
C Int.	the task of coefficient interpolation
C Ext.	the task of coefficient extrapolation
ϵ	the mean L_2 relative error
BC ϵ	the mean L_2 relative error on the boundary
N_P	the number of trainable parameters

Appendix of PAPM

In this appendix, we first summarize the notations in Tab.6 (A). Then, we describe the process model further, showing in detail the starting point of our problem (B). Secondly, we perform a further theoretical presentation on the details of embedding boundary conditions (C). The algorithm display and details of the proposed temporal and spatial stepping module (TSSM) are further elaborated (D.1) and (D.2), respectively. Moreover, the instructions of TSSM are discussed in (D.3). Subsequently, the five datasets are further described (E). The hyper-parameters settings of different baselines and PAPM are shown in detail (F). Finally, some additional experimental results are shown (G), which are data visualization (G.1), training and inference time cost-specific details (G.2), and detailed ablation studies (G.3).

A. Table of notations

A table of notations is given in Tab.6.

B. Process models

Pivotal in engineering disciplines, process models serve to represent and predict the dynamics of diverse process systems, from entire plants to single equipment pieces. These models primarily rely on the interplay between conservation and constitutive equations, ensuring an accurate depiction of system dynamics. Conservation equations dictate the model's dynamics using partial differential equations that govern primary physical quantities like mass, energy, and momentum. On the other hand, constitutive equations relate potentials to extensive variables via algebraic equations, such as flows, temperatures, pressures, concentrations, and enthalpies, enriching the model's comprehensiveness. Additionally, accounting for initial and boundary conditions ensures the model's reliability, making these four components interdependently integral to the model's solid mathematical framework.

Conservation equations. The general form in differential representation is:

$$\frac{\partial \boldsymbol{U}}{\partial t} = -\nabla \cdot (\boldsymbol{J}_C + \boldsymbol{J}_D) + \boldsymbol{q} + \boldsymbol{F} \tag{5}$$

where $\boldsymbol{J}_C = \boldsymbol{U} \cdot \boldsymbol{v}$, $\boldsymbol{J}_D = -\boldsymbol{D} \cdot \nabla \boldsymbol{U}$, and $\boldsymbol{U}(\boldsymbol{x}, t)$ represents the state of the model, which is the object of our modeling, $\boldsymbol{x} \in \Omega$ and $t \in [0, T], T \in \mathbb{R}_+$. $\boldsymbol{J}_C$ represents convective flows, and $\boldsymbol{J}_D$ represents diffusive or molecular flows. $\boldsymbol{v}$ describes the convective flow pattern into or out of the system volume. $\boldsymbol{D}$ represents the diffusion coefficient. $\boldsymbol{q}$, the internal source term, for example, the chemical reaction for component mass conservation, where species appear or are consumed due to reactions within the space of interest. Other internal source terms arise from energy dissipation, conversion, compressibility, or density changes. $\boldsymbol{F}$, the external source term, including gravitational, electrical, and magnetic fields as well as pressure fields.

Constitutive equations. For the internal source term, it usually depends on the state of the model and can be expressed as $\boldsymbol{q} = h_q(\boldsymbol{U}, \boldsymbol{x}, t)$. The external source term is usually related to the external effects imposed and can be expressed as $\boldsymbol{F} = h_F(\boldsymbol{X}_F)$, where $\boldsymbol{X}_F$ is a vector of parameters imposed externally, which may include voltages, pressures, etc. For convective flows, the velocity $\boldsymbol{v}$ may be determined by the state of the model, which can be expressed as $\boldsymbol{v} = g(\boldsymbol{U}, \boldsymbol{x}, t)$.

Initial conditions (IC). Every process or system evolves over time, but we need a reference or a starting point to predict or understand this evolution. The initial conditions provide this starting point. For example, in the context of a reactor, initial conditions might describe the concentration of various reactants at $t = 0$. Mathematically, IC can be represented as $\boldsymbol{U}(\boldsymbol{x}, 0) = \boldsymbol{U}_0(\boldsymbol{x})$.

Boundary conditions (BC). Initial conditions set the foundation at $t = 0$, while boundary conditions inform how a system evolves and interacts with its environment, for instance, by specifying heat flux at a heat exchanger's boundary or flow rate at a reactor's inlet. These boundary conditions can be categorized as Dirichlet, prescribing specific values like temperature on the boundary; Neumann, defining derivatives or fluxes such as the heat flux; and Robin, which combines aspects of both

Dirichlet and Neumann, encompassing parameters like both heat transfer rates and surface temperatures. Regardless of the type, they're mathematically expressed as $\boldsymbol{U}(\boldsymbol{x}_b, t) = \boldsymbol{f}_b(t)$, where $\boldsymbol{x}_b \in \partial\Omega$.

C. Embedding Boundary Conditions

This part covers the method of embedding four different boundary conditions (**Dirichlet**, **Neumann**, **Robin**, and **Periodic**) into neural networks via convolution padding. Let's consider a rectangular region in a $2D$ space, $\Omega = [0, a] \times [0, b]$, which can be discretized into an $M \times N$ grid, $\delta x = \frac{a}{M}$, $\delta y = \frac{b}{N}$. Each grid point can be represented as $X_{ij} = (x_i, y_j)$, where $i = 1, 2, ..., M$ and $j = 1, 2, ..., N$. Hence, we can transform the continuous space into a discrete grid of points.

Boundary Conditions on the X-axis. The direction vector is $\mathbf{n} = (1, 0)^T$, which means the boundary conditions are the same for each y value.

- **Dirichlet**: If the boundary condition is given as $\boldsymbol{U}(X, t) = f(X, t), X \in \partial\Omega$, the discrete form would be $\boldsymbol{U}_{Mj} = f_j$, and we can use a padding method in the convolution kernel $\boldsymbol{U}_{Mj} = f_j$.
- **Neumann**: If the boundary condition is given as $\frac{\partial \boldsymbol{U}(X,t)}{\partial \mathbf{n}} = f(X, t), X \in \partial\Omega$, the discrete form would be $\frac{\boldsymbol{U}_{(M+1)j} - \boldsymbol{U}_{(M-1)j}}{2\delta x} = f_j$ and we can use a padding method in the convolution kernel $\boldsymbol{U}_{(M+1)j} = \boldsymbol{U}_{(M-1)j} + (2 \times \delta x) \times f_j$.
- **Robin**: If the boundary condition is given as $\alpha\boldsymbol{U}(X, t) + \beta\frac{\partial \boldsymbol{U}(X,t)}{\partial \mathbf{n}} = f(X, t), X \in \partial\Omega$, the discrete form would be $\alpha\boldsymbol{U}_{Mj} + \beta\frac{\boldsymbol{U}_{(M+1)j} - \boldsymbol{U}_{(M-1)j}}{2\times\delta x} = f_j$. We can use a padding method in the convolution kernel $\boldsymbol{U}_{(M+1)j} = \frac{2\times\delta x}{\beta}(f_j - \alpha\boldsymbol{U}_{Mj}) + \boldsymbol{U}_{(M-1)j}$.
- **Periodic**: If the boundary condition is given as $\boldsymbol{U}(X_1, t) = \boldsymbol{U}(X_2, t), X_1 \in \partial\Omega_1, X_2 \in \partial\Omega_2$, where Ω_1 denotes the left boundary and Ω_2 the right boundary, the discrete form would be $\boldsymbol{U}_{Mj} = \boldsymbol{U}_{1j}$. We can use a padding method in the convolution kernel $\boldsymbol{U}_{Mj} = \boldsymbol{U}_{1j}, \boldsymbol{U}_{(M+1)j} = \boldsymbol{U}_{2j}$.

Boundary Conditions on the Y-axis. The direction vector is $\mathbf{n} = (0, 1)^T$. The basic handling method is similar to the x-direction case but with the grid spacing replaced with δy, and the boundary conditions applied to the upper and lower boundaries, *i.e.*, $j = 1$ and $j = N$. The corresponding y-direction expressions can be derived by replacing x with y in the x-direction expressions and swapping i with j.

Arbitrary Direction Boundary Conditions in the Rectangular Area. The direction vector $\mathbf{n} = (cos(\theta), sin(\theta))^T$. Both x and y directions need to be considered, resulting in the following expressions for each of the four boundary conditions:

- **Dirichlet**:Given the condition $\boldsymbol{U}(X, t) = f(X, t)$, its discrete form remains $\boldsymbol{U}_{ij} = f_{ij}$. The corresponding padding method in the convolution kernel is $\boldsymbol{U}_{ij} = f_{ij}$.
- **Neumann**: For the boundary condition $\frac{\partial \boldsymbol{U}(X,t)}{\partial \mathbf{n}} = f(X, t)$, the discrete form can be represented as $cos(\theta)\frac{\boldsymbol{U}_{(i+1)j} - \boldsymbol{U}_{(i-1)j}}{2\delta x} + sin(\theta)\frac{\boldsymbol{U}_{i(j+1)} - \boldsymbol{U}_{i(j-1)}}{2\delta y} = f_{ij}$. The corresponding padding method in the convolution kernel can be written as $\boldsymbol{U}_{(i+1)j} = \boldsymbol{U}_{(i-1)j} + 2cos(\theta)\delta x f_{ij}$ and $\boldsymbol{U}_{i(j+1)} = \boldsymbol{U}_{i(j-1)} + 2sin(\theta)\delta y f_{ij}$.
- **Robin**: Given the condition $\alpha\boldsymbol{U}(X, t) + \beta\frac{\partial \boldsymbol{U}(X,t)}{\partial \mathbf{n}} = f(X, t)$, the discrete form becomes $\alpha\boldsymbol{U}_{ij} + \beta cos(\theta)\frac{\boldsymbol{U}_{(i+1)j} - \boldsymbol{U}_{(i-1)j}}{2\delta x} + \beta sin(\theta)\frac{\boldsymbol{U}_{i(j+1)} - \boldsymbol{U}_{i(j-1)}}{2\delta y} = f_{ij}$. The corresponding padding method in the convolution kernel is $\boldsymbol{U}_{(i+1)j} = \frac{1}{\beta cos(\theta)}[f_{ij} - \alpha\boldsymbol{U}_{ij}] \times 2\delta x + \boldsymbol{U}_{(i-1)j}$ and $\boldsymbol{U}_{i(j+1)} = \frac{1}{\beta sin(\theta)}[f_{ij} - \alpha\boldsymbol{U}_{ij}] \times 2\delta y + \boldsymbol{U}_{i(j-1)}$.
- **Periodic**: For the condition $\boldsymbol{U}(X_1, t) = \boldsymbol{U}(X_2, t)$, the discrete form is $\boldsymbol{U}_{Mj} = \boldsymbol{U}_{1j}$ and $\boldsymbol{U}_{Ni} = \boldsymbol{U}_{1i}$. The corresponding padding method in the convolution kernel is $\boldsymbol{U}_{Mj} = \boldsymbol{U}_{1j}, \boldsymbol{U}_{(M+1)j} = \boldsymbol{U}_{2j}$ and $\boldsymbol{U}_{Ni} = \boldsymbol{U}_{1i}, \boldsymbol{U}_{i(N+1)} = \boldsymbol{U}_{i2}$.

Directionless Boundary Conditions in the Rectangular Area. The following strategies are employed for handling the Neumann and Robin boundary conditions:

- For $\frac{\partial \boldsymbol{U}}{\partial X} = f(X, t)$, the discrete form is $\frac{\boldsymbol{U}_{(i+1)j} - \boldsymbol{U}_{(i-1)j}}{2\delta x} = f_{ij}$ and $\frac{\boldsymbol{U}_{i(j+1)} - \boldsymbol{U}_{i(j-1)}}{2\delta y} = f_{ij}$. We can use a padding method in the convolution kernel where $\boldsymbol{U}_{(i+1)j} = \boldsymbol{U}_{(i-1)j} + 2\delta x f_{ij}$ and $\boldsymbol{U}_{i(j+1)} = \boldsymbol{U}_{i(j-1)} + 2\delta y f_{ij}$.

- For $\alpha \boldsymbol{U}(X,t) + \beta \frac{\partial \boldsymbol{U}(X,t)}{\partial X} = f(X,t)$, the discrete form is $\alpha \boldsymbol{U}_{ij} + \beta \frac{\boldsymbol{U}_{(i+1)j} - \boldsymbol{U}_{(i-1)j}}{2\delta x} = f_{ij}$ and $\alpha \boldsymbol{U}_{ij} + \beta \frac{\boldsymbol{U}_{i(j+1)} - \boldsymbol{U}_{i(j-1)}}{2\delta y} = f_{ij}$. We can use a padding method in the convolution kernel where $\boldsymbol{U}_{(i+1)j} = \frac{1}{\beta}[f_{ij} - \alpha \boldsymbol{U}_{ij}] \times 2\delta x + \boldsymbol{U}_{(i-1)j}$ and $\boldsymbol{U}_{i(j+1)} = \frac{1}{\beta}[f_{ij} - \alpha \boldsymbol{U}_{ij}] \times 2\delta y + \boldsymbol{U}_{i(j-1)}$.

D. Supplemental TSSM

D.1. Algorithm Display

Here, we provide the pseudo-code for the Temporal-Spatial Stepping Module (TSSM) training, offering a comprehensive understanding of our approach. As shown in Alg. 1, the structure-preserved localized operator is detailed. The latter is shown in Alg. 2, and the third one, the hybrid operator, is a combination of these two operators.

Algorithm 1 Structure-preserved localized operator.

Initialization: Fixed or pre-defined convolutional kernels K (with parameters θ), as shown in Fig. 3 (Left); Initialize other network parameters $\theta \in \Theta$
Input: A set of inputs $\boldsymbol{a}_k$ for $1 \le k \le D_0$, time interval Δt, and temporal trajectory length T'
Output: The mapping $\mathcal{G}_\theta$, where $\tilde{\boldsymbol{U}}_k \leftarrow \tilde{\mathcal{G}}_\theta(\boldsymbol{a}_k)$
for $k = 1$ *to* D_0 **do**
  $\boldsymbol{a}_k \leftarrow \boldsymbol{U}_k^{t=0}, \boldsymbol{\lambda}, X_F$, BCs
  for $t = 1$ *to* T' **do**
    $\tilde{\boldsymbol{U}}_k^t \leftarrow \boldsymbol{U}_k^t$ # Embedding BCs
    $(\cdot)^n, \nabla^n(\cdot), n = 0,1,2 \leftarrow \tilde{\boldsymbol{U}}_k^t \circledast K$
    DF, CF $\leftarrow (\cdot)^n, \nabla^n(\cdot), \boldsymbol{a}_k, n = 0,1,2$
    IST, EST $\leftarrow$ ResNet$(\tilde{\boldsymbol{U}}_k^t, \boldsymbol{a}_k)$
    $\frac{\partial \tilde{\boldsymbol{U}}_k^t}{\partial t} \leftarrow$ DF+CF+IST+EST
    Next states $\boldsymbol{U}_k^{t+1} \leftarrow$ Neural ODE$(\tilde{\boldsymbol{U}}_k^t, \frac{\partial \tilde{\boldsymbol{U}}_k^t}{\partial t}; \Delta t)$
    Update input $\boldsymbol{a}_k \leftarrow \boldsymbol{U}_k^{t+1}$
  end
  Subsequent trajectory $\tilde{\boldsymbol{U}}_k \leftarrow \tilde{\mathcal{G}}_\theta(\boldsymbol{a}_k)$
  Loss $\mathcal{L}_r(\boldsymbol{\theta}) \leftarrow$ Eq. 3
  Update weights θ by minimizing the loss $\mathcal{L}_r(\boldsymbol{\theta})$
end

Algorithm 2 Structure-Preserved Spectral Operator

Initialization: E-Conv (1×1 conv) and S-conv (spectral convolutions) with parameters θ as shown in Fig. 3 (Right)
Input: A set of inputs $\boldsymbol{a}_k$ for $1 \le k \le D_0$, time interval Δt, and temporal trajectory length T'
Output: The mapping $\mathcal{G}_\theta$ where $\tilde{\boldsymbol{U}}_k \leftarrow \tilde{\mathcal{G}}_\theta(\boldsymbol{a}_k)$
for $k = 1$ **to** D_0 **do**
  $\boldsymbol{a}_k \leftarrow \boldsymbol{U}_k^{t=0}, \boldsymbol{\lambda}, X_F$, BCs
  for $t = 1$ **to** T' **do**
    $\tilde{\boldsymbol{U}}_k^t \leftarrow \boldsymbol{U}_k^t$ # Embedding BCs
    $k_x, k_y, \hat{\cdot} \leftarrow$ FFT$(\tilde{\boldsymbol{U}}_k^t)$
    $\hat{(\cdot)}^n, \hat{\nabla}^n(\cdot), n = 0,1,2 \leftarrow$ E-Conv$(k_x, k_y, \hat{\cdot})$
    DF, CF $\leftarrow$ IFFT$((\cdot)^n, \nabla^n(\cdot), \boldsymbol{a}_k)$
    IST, EST $\leftarrow$ S-conv$(\tilde{\boldsymbol{U}}_k^t, \boldsymbol{a}_k)$
    $\frac{\partial \tilde{\boldsymbol{U}}_k^t}{\partial t} \leftarrow$ DF + CF + IST + EST
    Next states $\boldsymbol{U}_k^{t+1} \leftarrow$ Neural ODE$(\tilde{\boldsymbol{U}}_k^t, \frac{\partial \tilde{\boldsymbol{U}}_k^t}{\partial t}; \Delta t)$
    Update input $\boldsymbol{a}_k \leftarrow \boldsymbol{U}_k^{t+1}$
  end
  Subsequent trajectory $\tilde{\boldsymbol{U}}_k \leftarrow \tilde{\mathcal{G}}_\theta(\boldsymbol{a}_k)$
  Compute loss $\mathcal{L}_r(\boldsymbol{\theta})$ according to Eq. 3
  Update weights θ by minimizing the loss $\mathcal{L}_r(\boldsymbol{\theta})$
end

D.2. Details of TSSM

D.2.1. Structure-preserved localized operator

Fixed convolution operations. The differential operator can be approximated via convolution operations. For a one-dimensional function $u(x)$, we could use a convolution kernel of the form:

$$K = \frac{1}{2\Delta x}[-1, 0, 1] \tag{6}$$

where Δx represents the step size. This convolution operation, corresponding to this kernel, can approximate the first-order central difference operator as follows:

$$u'(x) \approx \frac{u(x + \Delta x) - u(x - \Delta x)}{2\Delta x} \approx u(x) \circledast K, \tag{7}$$

with $\circledast$ denoting the convolution operation. For a two-dimensional function, it can be decomposed into a convolution of two one-dimensional functions. Assuming $u(x, y)$ is a two-dimensional function, the kernel could be formed as:

$$K = \frac{1}{h^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \frac{1}{h^2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \frac{1}{h^2}\begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \tag{8}$$

where $h = \Delta x = \Delta y$ signifies the step size. The convolution operation corresponding to this kernel can approximate the second-order central difference operator, which is:

$$\begin{aligned} \nabla^2 u(x, y) &= \frac{\partial^2 u(x, y)}{\partial x^2} + \frac{\partial^2 u(x, y)}{\partial y^2} \\ &\approx \frac{u(x + h, y) - 2u(x, y) + u(x - h, y)}{h^2} + \frac{u(x, y + h) - 2u(x, y) + u(x, y - h)}{h^2} \\ &= \frac{u(x + h, y) + u(x, y + h) - 4u(x, y) + u(x - h, y) + u(x, y - h)}{h^2} \\ &= u(x, y) \circledast K. \end{aligned} \tag{9}$$

Analogously, different convolution kernels can approximate other orders' differential operators. Utilizing convolution operations to approximate differential operators can boost computational efficiency. Nevertheless, careful consideration is needed when choosing a convolution kernel, as different kernels can influence the stability and accuracy of the numerical solution.

Selection of FD kernels. FD kernels are used to approximate derivative terms in PDEs, which directly affect the computational efficiency and the reconstruction accuracy. Therefore, it is crucial to choose appropriate FD kernels for discretized-based learning frameworks. For spatio-temporal systems, we need to consider both temporal and spatial derivatives. In specific, the second-order central difference is utilized for calculating temporal derivatives, *i.e.*,

$$\frac{\partial u}{\partial t} = \frac{-u(t - \delta t, \xi, \eta) + u(t + \delta t, \xi, \eta)}{2\delta t} + \mathcal{O}\left((\delta t)^2\right), \tag{10}$$

where $\{\xi, \eta\}$ represent the spatial locations and δt is time spacing. In the network implementation, it can be organized as a convolutional kernel K_t,

$$K_t = [-1, 0, 1] \times \frac{1}{2\delta t}.$$

Likewise, we also apply the central difference to calculate the spatial derivatives for internal nodes and use forward/backward differences for boundary nodes. For instance, in this paper, the fourth-order central difference is utilized to approximate the first and second spatial derivatives. The FD kernels for 2D cases with the shape of 5×5 are given by

$$K_{s,1} = \frac{1}{12(\delta x)}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & -8 & 0 & 8 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, K_{s,2} = \frac{1}{12(\delta x)^2}\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 16 & 0 & 0 \\ -1 & 16 & -60 & 16 & -1 \\ 0 & 0 & 16 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}, \tag{11}$$

where δx denotes the grid size of HR variables; $K_{s,1}$ and $K_{s,2}$ are FD kernels for the first and second derivatives, respectively. In addition, we conduct a parametric study on the selection of FD kernels, including the second-order (3×3), the fourth-order (5×5), and the sixth-order (7×7) central difference strategies.

D.2.2. STRUCTURE-PRESERVED SPECTRAL OPERATOR

In Fourier space, the simplification of problem-solving involves converting differential operations and wave number multiplications. In 2D space, the discrete values of the set function $\Phi(x, y)$ in real space, denoted as Φ_{ij}, correspond to $\hat{\Phi}_{mn}$ in Fourier space, with k_x and k_y representing the respective wave numbers. Differential operators are transformed as follows. (1) The first-order differential operator, $\frac{\partial \Phi}{\partial x}$, becomes $ik_x\hat{\Phi}_{mn}$ in the x direction and $ik_y\hat{\Phi}_{mn}$ in the y direction. (2) The second-order differential operator, $\frac{\partial^2 \Phi}{\partial x^2}$, is represented as $-k_x^2\hat{\Phi}_{mn}$ for the x direction and $-k_y^2\hat{\Phi}_{mn}$ for the y direction. (3) The Laplacian operator $\nabla^2\Phi$ transforms into $(-k_x^2 - k_y^2)\hat{\Phi}_{mn}$ in Fourier space.

Moreover, for a 2D flow field, the relationship between the flow function ψ and the velocity fields (u, v) can be expressed by the following partial differential equation, $u = \frac{\partial \psi}{\partial y}$ and $v = -\frac{\partial \psi}{\partial x}$, where we can use k_x and k_y to obtain differential results. Moreover, here **S-Conv** is from FNO (Li et al., 2020) (the module named as "SpectralConv2d_fast"). This "SpectralConv2d_fast" class is a neural network module that performs a 2D spectral convolution by applying an FFT, a learned linear transformation in the Fourier domain, and an IFFT.

D.3. The Instructions of TSSM

Knowing whether a system exhibits **explicit**, **implicit**, or **hybrid** structures is beneficial but not strictly necessary before choosing a structure. Our method provides the flexibility to select an appropriate architecture based on the problem's characteristics, even with partial knowledge. At the same time, complete comprehension of these structural types can guide the choice more precisely. In the setting of our problem, the conservation relation of the process system is of definite form,

$$\frac{\partial \boldsymbol{U}}{\partial t} = -\nabla \cdot (\boldsymbol{J}_C + \boldsymbol{J}_D) + \boldsymbol{q} + \boldsymbol{F}$$

Under such a premise, we can easily select the appropriate structures for different spatio-temporal systems according to the following rules (**The choice of different paths** and **Impact of Mismatched Path Selection on Performance**).

D.3.1. The choice of different paths

As shown in Tab. 7, we defined three structural types: localized, spectral, and hybrid, each suited to different system characteristics. Localized path is appropriate for systems like the Burgers equation, where diffusion and convection terms are explicit and uncoupled. Spectral path is better for systems with coupled terms, like the vorticity form of the NS equation. Hybrid path suits systems combining these elements, like the general form of the NS equation, where diffusion and convection are uncoupled, but the internal source term is coupled (i.t. the pressure and target velocity fields are coupled to satisfy the Poisson equation). When the interaction between convection and diffusion terms is unknown, the spectral path is a reliable approximation for all these elements, and the hybrid path addresses the source term (Internal/External Source).

Table 7. The choice of different paths.

	Localized	**Spectral**	**Hybrid**
Characteristic	Explicit	Implicit	Explicit+Implicit
Example	$-u\nabla u + \nabla^2 u$	$-u\nabla w + \nabla^2 w$	$-u\nabla u + \nabla^2 u - \nabla p$
Diffusive Flows/Convective Flows	Pre-defined convolution	E-Conv	Pre-defined convolution
Internal Source Term/External Source Term	ResNet block	S-Conv block	ResNet/S-Conv block

Table 8. Impact of Mismatched Path Selection on Performance.

Datasets	**Category**	Localized	Spectral	Hybrid
Burgers2d	Localized	0.039	0.043	**0.037**
NS2d ($\nu = 1e$-5)	Spectral	0.061	**0.034**	0.048
NSM2d	Hybrid	0.205	0.196	**0.189**

D.3.2. Impact of Mismatched Path Selection on Performance

Using all three paths, we evaluated the performance across three datasets (Burgers2d for Localized, NS2d for Spectral, and NSM2d for Hybrid). The results are summarized as Tab. 8. For Burgers2d and NSM2d, with explicit diffusion and convection terms, physical space approximation outperforms spectral space gradient approximation. For NS2d, featuring implicit diffusion and convection terms, spectral space gradient approximation is superior to physical space approximation. Across all datasets, the Hybrid path surpasses the Localized path in performance. However, this comes at a cost: an increase

in model parameters from 14k to 35k, higher computational demands, and marginal performance gains. Consequently, the Localized path is preferred for scenarios without complex couplings, such as Burgers2d.

Table 9. The difference of five datasets.

Dataset	Category	Various Conditions			
		Initial conditions	Boundary conditions	Coefficients	External sources
Burgers2d	Localized	✓	Periodic	✓	unknown
RD2d	Localized	✓	No-flow Neumann		unknown
NS2d	Spectral	✓	Periodic		unknown
Lid2d	Hybrid		Dirichlet, Neumann	✓	
NSM2d	Hybrid	✓	Dirichlet, Neumann	✓	unknown

E. Datasets

As shown in Tab.9, we employ five datasets spanning diverse domains, such as fluid dynamics and heat conduction. Based on the TSSM scheme employed by PAPM, we categorize the aforementioned five datasets into three types: **Burgers2d** and **RD2d** fall under the **localized** category, **NS2d** is classified as **spectral**, while **Lid2d** and **NSM2d** are designated as **hybrid**. The generations of Lid2d and NSM2d are detailed via COMSOL multiphysics in E.2. We are particularly keen to make **Lid2d** and **NSM2d** publicly available, anticipating various research endeavors on these datasets by the community.

E.1. Five datasets

Burgers2d (Huang et al., 2023a). The 2D Burgers equation is a fundamental nonlinear partial differential equation. Its formulation is given by:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} = -\boldsymbol{u} \cdot \nabla \boldsymbol{u} + v\Delta \boldsymbol{u} + \boldsymbol{f}, \\ \boldsymbol{u}|_{t=0} = \boldsymbol{u}_0(x, y) \end{cases} \tag{12}$$

where $\boldsymbol{u} = (u(x, y, t), v(x, y, t))$ represents the velocity field, and the spatial domain is $\Omega = [0, 2\pi]^2$ with periodic boundary conditions. The viscosity coefficient v varies within the range $v \in [0.001, 0.1]$. The forcing term is defined as:

$$f(x, y, \boldsymbol{u}) = (\sin(v)\cos(5x + 5y), \sin(u)\cos(5x - 5y))^\top. \tag{13}$$

The initial condition, denoted as $\boldsymbol{u}_0(x, y)$, is drawn from a Gaussian random field characterized by a variance of $25(-\Delta + 25I)^{-3}$. Subsequently, it is linearly normalized to fall within the $[0.1, 1.1]$ range. A total of $N = 250$ samples are generated, each spanning $M = 3200$ time steps with a step size of $\delta t = \frac{0.01}{32}$. A high-resolution traditional numerical solver is employed to generate high-precision numerical solutions. This solver utilizes the δt value and operates on a finely discretized 256×256 grid. The resulting high-precision solutions are stored at intervals of every 32 time step, resulting in 100 time slices. Subsequently, these solutions are downsampled to a coarser 64×64 grid.

RD2d (Takamoto et al., 2022). Considering the 2D diffusion-reaction equation, the conservation of the activator u and inhibitor v can be represented as:

$$\begin{cases} \dfrac{\partial u}{\partial t} = -\nabla J_u + R_u, \dfrac{\partial v}{\partial t} = -\nabla J_v + R_v \\ J_u = -D_u \nabla u, \quad J_v = -D_v \nabla v \end{cases} \tag{14}$$

Where J_u and J_v are the flux terms for the activator and inhibitor, respectively. These represent the diffusive or molecular flows for each component. The reaction functions R_u and R_v for the activator and inhibitor, respectively, are defined by the Fitzhugh-Nagumo (FN) equation, written as $R_u = u - u^3 - k - v$ and $R_v = u - v$, where $k = 5 \times 10^{-3}$ and the diffusion coefficients for the activator and inhibitor are $D_u = 1 \times 10^{-3}$ and $D_v = 5 \times 10^{-3}$, respectively. The initial condition is characterized by a standard normal random noise, with $u(0, x, y) \sim \mathcal{N}(0, 1.0)$ for $x \in (-1, 1)$ and $y \in (-1, 1)$. The boundary conditions are defined as no-flow Neumann boundary conditions. This entails that the partial derivatives satisfy

the conditions: $D_u\partial_x u = 0$, $D_v\partial_x v = 0$, $D_u\partial_y u = 0$, and $D_v\partial_y v = 0$, all applicable for the domain $x, y \in (-1, 1)^2$. This dataset [2] is transformed into a coarser grid with dimensions of 64×64 while keeping the time step consistently constant.

NS2d (Li et al., 2020). We refer to **FNO** as the source for our exploration of the two-dimensional incompressible Navier-Stokes equation in vorticity form. This equation is defined on the unit torus and is outlined as follows:

$$\begin{cases} \partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) = \nu \Delta w(x,t) + f, \\ \nabla \cdot u(x,t) = 0, \\ w(x,0) = w_0(x), \end{cases} \tag{15}$$

where $x \in (0,1)^2$, $t \in (0,T]$, and u represents the velocity field, $w = \nabla \times u$ denotes the vorticity, w_0 stands for the initial vorticity distribution, $\nu \in \mathbb{R}_+$ signifies the viscosity coefficient and f denotes the forcing function. In this work, the viscosity coefficient is set to $\nu = 1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}$. It's worth noting that, for the purpose of maintaining a consistent evaluation framework, the resolution is standardized at 64×64 for both training and testing phases, given that the baseline methods are not inherently resolution-invariant.

Lid2d. A constant velocity across the top of the cavity creates a circulating flow inside. To simulate this, a constant velocity boundary condition is applied to the lid while the other three walls obey the no-slip condition. Different Reynolds numbers yield different results, so in this article, $Re \in [100, 1500]$ are applied. At high Reynolds numbers, secondary circulation zones are expected to form in the corners of the cavity. The system of differential equations (N-S equations) consists of two equations for the velocity components $\boldsymbol{u} = (u(x,y,t), v(x,y,t))$, and one equation for pressure $(p(x,y,t))$:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} = -\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \dfrac{1}{Re} \Delta \boldsymbol{u} - \nabla p, \\ \nabla \cdot \boldsymbol{u} = 0 \end{cases} \tag{16}$$

where $(x,y) \in (0,1)^2$. The initial condition is $(u,v,p) = \mathbf{0}$ everywhere. And the boundary conditions are: $u = 1$ at $y = 1$ (the lid), $(u,v) = \mathbf{0}$ on the other boundaries, $\partial p/\partial y = 0$ at y=0,p=0 at $y = 1$, and $\partial p/\partial x = 0$ at $x = 0, 1$. The data generation for the Lid2d is processed by COMSOL Multiphysics®, and a total of $N = 200$ samples are generated, each spanning $M = 1000$ time steps with a step size of $\delta t = \frac{0.1}{10}$. Every 10 steps, we save the data, resulting in 100 time slices. This solver utilizes the value of δt and operates on a finely discretized 128×128 grid. Subsequently, these solutions are downsampled to a coarser 64×64 grid.

NSM2d. Consider the Navier-Stokes equations with an additional magnetic field:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\nabla p + \nu \nabla^2 \boldsymbol{u} + \boldsymbol{F}, \quad t \in [0,T], \\ \nabla \cdot \boldsymbol{u} = 0, \end{cases} \tag{17}$$

where $(x,y) \in [0,4] \times [0,1]$, $\boldsymbol{u} = [u(x,y,t), v(x,y,t)] \in \mathbb{R}^2$ is the velocity vector, $p(x,y,t) \in \mathbb{R}$ is the pressure, $\nu = 1/Re$ represents the kinematic viscosity (with Re as the Reynolds number), and $\boldsymbol{F} = [F_x, F_y]$ is an external source term induced by the magnetic field. The components of $\boldsymbol{F}$ are defined as follows:

$$\begin{cases} F_x = mH \dfrac{\partial H}{\partial x}, \quad F_y = mH \dfrac{\partial H}{\partial y} \\ H(x,y) = \exp\left[-8\left((x - L/2)^2 + (y - W/2)^2\right)\right] \end{cases} \tag{18}$$

where $L = 4$, $W = 1$, $m = 0.16$ is the magnetization, and H is a time-invariant magnetic intensity. The simulation is conducted on a 2D rectangular domain $\{x, y\} \in [0,4] \times [0,1]$ with the following boundary conditions: the inflow boundary $(x = 0)$ is prescribed with a velocity distribution $\boldsymbol{u}(0,y,t)$, where y_0 represents the vertical position of the inlet jet center:

$$\boldsymbol{u}(0,y,t) = \begin{bmatrix} u(0,y,t) \\ v(0,y,t) \end{bmatrix} = \begin{bmatrix} \exp\left(-50\,(y - y_0)^2\right) \\ \sin(t) \cdot \exp\left(-50\,(y - y_0)^2\right) \end{bmatrix} \tag{19}$$

The outflow boundary $(x = 4)$ is set with a reference pressure $p(4,y,t) = 0$. The no-slip boundary condition is applied at the top and bottom walls $(y = 0, 1)$. The Reynolds number is dimensionless and ranges from 100 to 1500. The inlet jet

[2] This dataset can be downloaded at https://github.com/pdebench/PDEBench

position y_0 is varied within the domain $0.4 \leq y_0 \leq 0.6$. The data generation for the NSM2d is processed by COMSOL Multiphysics®, and a total of $N = 200$ samples are generated, each spanning $M = 1000$ time steps with a step size of $\delta t = \frac{0.2}{10}$. Every 10 steps, we save the data, resulting in 100 time slices. This solver utilizes the δt value and operates on a finely discretized 256×64 grid. Subsequently, these solutions are downsampled to a coarser 128×32 grid.

E.2. Detailed Data Generation Process

Our research employed COMSOL multiphysics software [3] for fluid dynamics simulation in a lid-driven cavity and a magnetic stirring scenario. The main text outlines the simulation parameters, utilizing grids of 128×128 and 256×64 for each case, respectively. The Time-Dependent Module, with specific time steps, was used for execution. The simulations required substantial computational resources, solving for 49,152 and 16,130 internal degrees of freedom (DOFs) in each scenario. To generate a comprehensive dataset, we varied simulation parameters, running 200 simulations for each scenario with different Reynolds numbers and, in the magnetic stirring case, the y_0 value. This data was stored in h5 format.

The computational intensity was significant: a single run in the lid-driven scenario took 91 seconds on average, while the magnetic stirring case took 226 seconds. The total computation time was approximately 10^5 seconds for all 200 cases, highlighting the time-consuming nature of such simulations. The intricacy of multi-physics coupling and the extensive computational demand in these simulations point towards the necessity of more efficient methods. This situation underscores the potential of neural networks in accelerating simulation processes. By leveraging neural networks, we aim to reduce the computational time significantly, addressing the inherent slowness of detailed simulations like those in our study. This approach could transform the feasibility and scalability of complex simulations in various scientific and engineering domains.

F. Details for experimental setup and models' hyper-parameters

F.1. Experimental setup

Based on the experimental setting of time extrapolation, we further conducted experiments in the following two parts: coefficient interpolation (referred to as **C Int.**) and coefficient extrapolation (referred to as **C Ext.**). For **C Int.**, the data is uniformly shuffled and then split into training, validation, and testing datasets in a $[7:1:2]$ ratio. However, in the case of **C Ext.**, the data splitter is determined based on the order of coefficients, with equal proportions $[7:1:2]$. For example, the viscosity coefficients are divided from largest to smallest, and the coefficients with the lowest viscosity, representing the most challenging tasks, are selected as the test set.

One crucial point to consider is that to maintain consistency with the data-driven approach (Shi et al., 2015; Stachenfeld et al., 2021; Li et al., 2020; Gupta & Brandstetter, 2022; Raonić et al., 2023), we must replace the initial time $t = 0$ with the initial step size t_0. We consistently set the initial time step size for all datasets across various tasks as $t_0 = 5$. In the test set, $T_{end} = 100$, except for the NS2d. Specifically, for NS2d with viscosity values of $\nu = 1\text{e}{-3}$ and $1\text{e}{-4}$, $T_{end} = 50$, while for $\nu = 1\text{e}{-5}$, $T_{end} = 20$. The trajectory length in the training set that can be used as label data is given by $T_{end}/2 - t_0$.

We train all models with AdamW (Loshchilov & Hutter, 2017) optimizer with the exponential decaying strategy, and epochs are set as 500. The causality parameter $\alpha_1 = 0.1$ and $\alpha_0 = 0.001$. The initial learning rate is $1e$-3, and the ReduceLROnPlateau schedule is utilized with a patience of 20 epochs and a decay factor of 0.8. For a fair comparison, the batch size is identical across all methods for the same task, and all experiments are run on $1 \sim 3$ NVIDIA Tesla P100 GPUs.

To account for potential variability due to the partitioning process, each experiment is performed three times, and the final result is derived as the average of these three independent runs. Except for the predefined parameters, the parameters of all models are initialized by Xavier (Glorot & Bengio, 2010), setting the scaling ratio $c = 0.02$.

F.2. Hyper-parameters

PAPM. In this work, PAPM designed three different temporal-spatial modeling methods according to the characteristics of five different data sets.

- **Localized Operator.** Burgers2d uses the predefined fixed convolution kernel as the convolution kernel parameter of diffusive and convective flows, while a 4-layer convolution layer characterizes the source term. Its channel is set to 16, and the GELU activation function is used. In RD2d, trainable convolutional kernels are used as the convolution kernel

[3] https://www.comsol.com/

parameter of diffusive flows, and the kernel is set to 5. For the source term, like burgers2d, a four-layer convolutional layer with channel 16 and GELU is used to characterize the source term.

- **Spectral Operator.** After FFT, k_x, k_y and $\hat{w}$ are input into a 1×1 conv for dot product, which is used to solve the partial derivatives of vorticity w and velocity field u, and then to physical space through IFFT. Simple operations such as multiplication and addition are performed according to conservation relations. As for the source term is characterized by a layer of S-Conv (*i.e.*, spectralConv2d_fast) with (width= 12, modes1= 12, modes2= 12).
- **Hybrid Operator.** According to the velocity part of the conservation equation, we set the kernel as five using trainable convolutional kernels as the convolution kernel parameters of diffusive and convective flows. We use a three-layer convolutional layer with channel 16 and GELU to represent the source term. Then, the intermediate results of the velocity field are fed into an S-Conv (width= 8, modes1= 8, modes2= 8) to map the complete velocity field and pressure field.

ConvLSTM (Shi et al., 2015). Specializing in spatial-temporal prediction, ConvLSTM blends LSTM's temporal cells with CNN spatial extraction. The setup consists of three distinct blocks: an encoding block employing a 5×5 convolution kernel with channel 32, an LSTM cell-based forecasting block, and a decoding block featuring 2 CNN layers with 5×5 kernels and 2 Res blocks. Notably, multi-step predictions are achieved through state concatenation within the forecasting block. Despite its strengths, its performance in complex process systems can be limited due to potential error accumulations.

Dil-ResNet (Stachenfeld et al., 2021). This model combines the encode-process-decode paradigm with the dilated convolutional network. The processor consists of $N = 4$ residual blocks connected in series, and each is made of dilated CNN stacks with residual connections. One stack consists of 7 dilated CNN layers with dilation rates of $(1, 2, 4, 8, 4, 2, 1)$, where a dilated rate of N indicates that each pixel is convolved with multiples of N pixels away. Each CNN layer in the processor is followed by ReLU activation. The key part of this network is a residual connection, which helps avoid vanishing gradients, and dilations allow long-range communication while preserving local structure. We found difficulties running this model on complex datasets due to computing and memory constraints.

time-FNO2D (Li et al., 2020). This model applies matrix multiplications in the spectral space with learnable complex weights for each component and linear updates and combines embedding in the spatial domain. The model consists of 2 MLP layers for encoding and decoding and 4 Fourier operation blocks (width= 12, modes1= 12, modes2= 12). Each block contains Fourier and CNN layers, followed by GELU activation. Optionally, low-pass filtering truncates high-frequency modes along each dimension in the Fourier-transformed grid.

MIONet (Jin et al., 2022). In the original paper, the Depth of MIONet is set to 2, the width is 200, and the number of parameters is 161K. Build MIONet with DeepXDE (Lu et al., 2021) [4]. In this work, we have greatly adjusted the number of parameters; the number of parameters is about 20k, and the width is set to 20. Consistent with FNO, MLP is also introduced to construct the project layer for data, and then projection is also carried out in output. Other contents are consistent with the original text.

U-FNet (Gupta & Brandstetter, 2022). This model improves U-Net architectures, replacing lower blocks both in the downsampling and in the upsampling path of U-Net architectures by Fourier blocks, where each block consists of 2 FNO layers and residual connections. Other contents are consistent with the original text. In this work, we have adjusted $n\_input\_scalar\_components$ and $n\_output\_scalar\_components$ to the number of channels of the physical field in our datasets, and both $time\_history$ and $time\_future$ are set to 5 for better fitting.

CNO (Raonić et al., 2023).This model proposes a sequence of layers with the convolutional neural operator, mapping between bandlimited functions based on U-Net architectures. The convolutional neural operator consists of 4 different blocks, i.e., the downsampling block, the upsampling block, the invariant block, and the ResNet block. In the original paper, the width and height of spatial size for the mesh grid should be identical. In this work, we relaxed this restriction in activation function $filtered\_lrelu$ to fit on non-square grids like the NSM2d dataset. Other contents are consistent with the original text.

PeRCNN (Rao et al., 2023). The network consists of two components: a fully convolutional decoder as an initial state generator and a novel recurrent block named $\prod$-block for recursively updating the state variables. Since our experiments' available measurement size is full, we have omitted this decoder. In the recurrent $\prod$-block, the state first goes through multiple parallel 1×1 Conv layers with stride 1 and output channel 32. The feature maps produced by these layers are then fused via the elementwise product operation. Then, the multi-channel goes through a conv layer with a filter size of 1 to

[4] https://github.com/lululxvi/deepxde

obtain the output of the desired number of channels. We found this method unstable when approaching nonlinear complex terms and prone to NaN values during training.

PPNN (Liu et al., 2024). This model combines known partial nonlinear functions with a trainable neural network, which is named ConvResNet. The only difference between these two models is that a trainable portion of PPNN has an extra input variable $\mathcal{F}$, provided by the PDE-preserving portion of PPNN. The state first goes through the decoder, which is made of four ConvResNet blocks, and each of them consists of a 7×7 kernel with 96 channels and a zero padding of 3. The following decoder includes a pixel shuffle with an upscale factor equal to 4 and a convolution layer with a 5×5 kernel. Due to the physics-aware design, this model shows lower relative error in the extrapolation range.

G. Additional experimental results

In this section, some additional experimental results are shown, which are data visualization (G.1), training and inference time cost-specific details (G.2), and detailed ablation studies (G.3).

G.1. Visualization

Fig. 7 and Fig. 8 showcase the results across five extrapolation time slices on Burgers2d and RD2d datasets. Both datasets clearly show that the physics-aware methods, PPNN and PAPM (our), can predict the dynamics of these two complex systems well. However, in the second half of the extrapolation ($T \geq \frac{1}{2}T_{end}$), It can be seen that our method PAPM is better than PPNN in local detail reconstruction. It is worth mentioning that our method has only 1% of the number of parameters and FLOPs of PPNN. However, the effect is better than PPNN, which further affirms the superiority of our structured design and specific spatio-temporal modeling method. Fig. 9 and Fig. 10 show the visual effects between different terms of PAPM and the numerical results, which can further prove that PAPM can learn the equation's convection/diffusion/source parts.

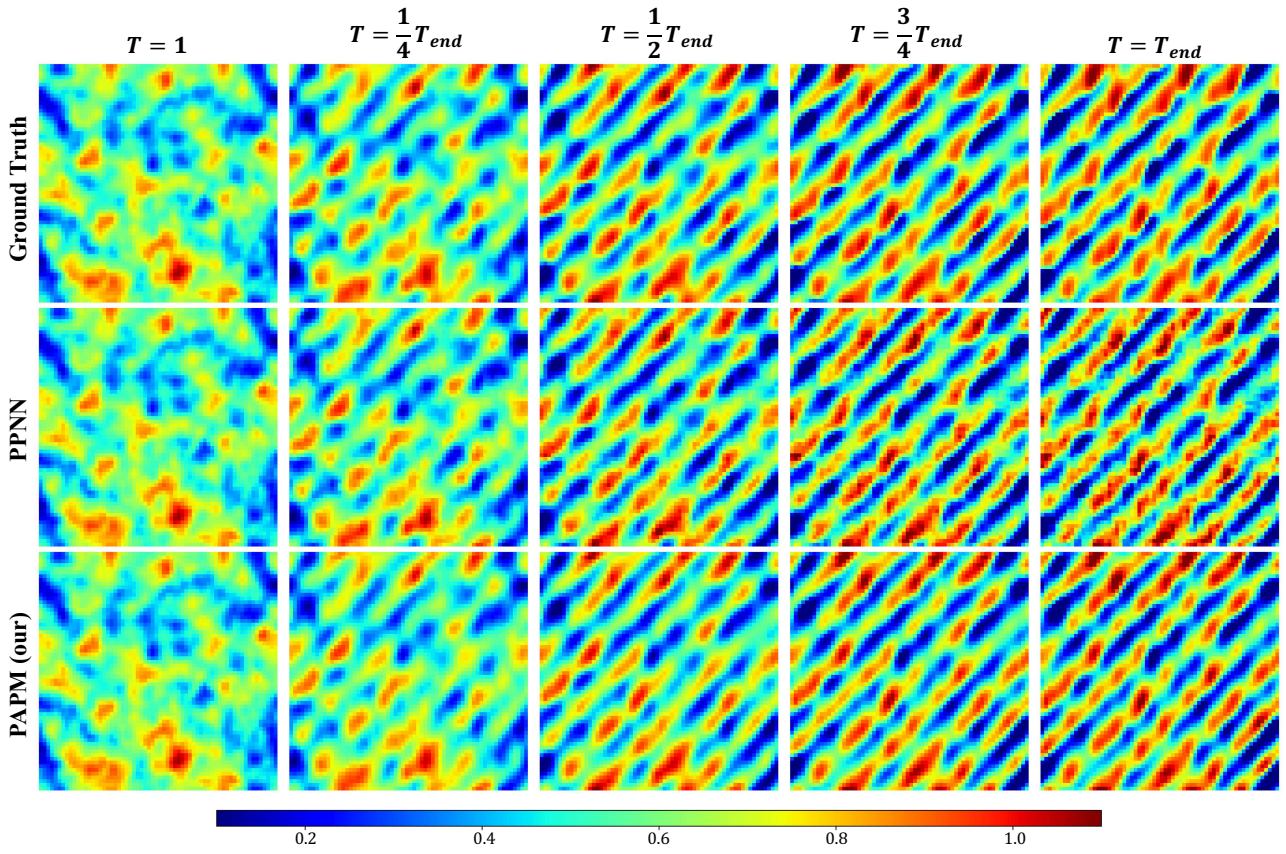

Figure 7. Predicted flow velocity ($||\boldsymbol{u}||_2$) snapshots by PPNN, and PAPM (Ours) vs. Ground Truth (GT) on Burgers2d dataset in T Ext. task.

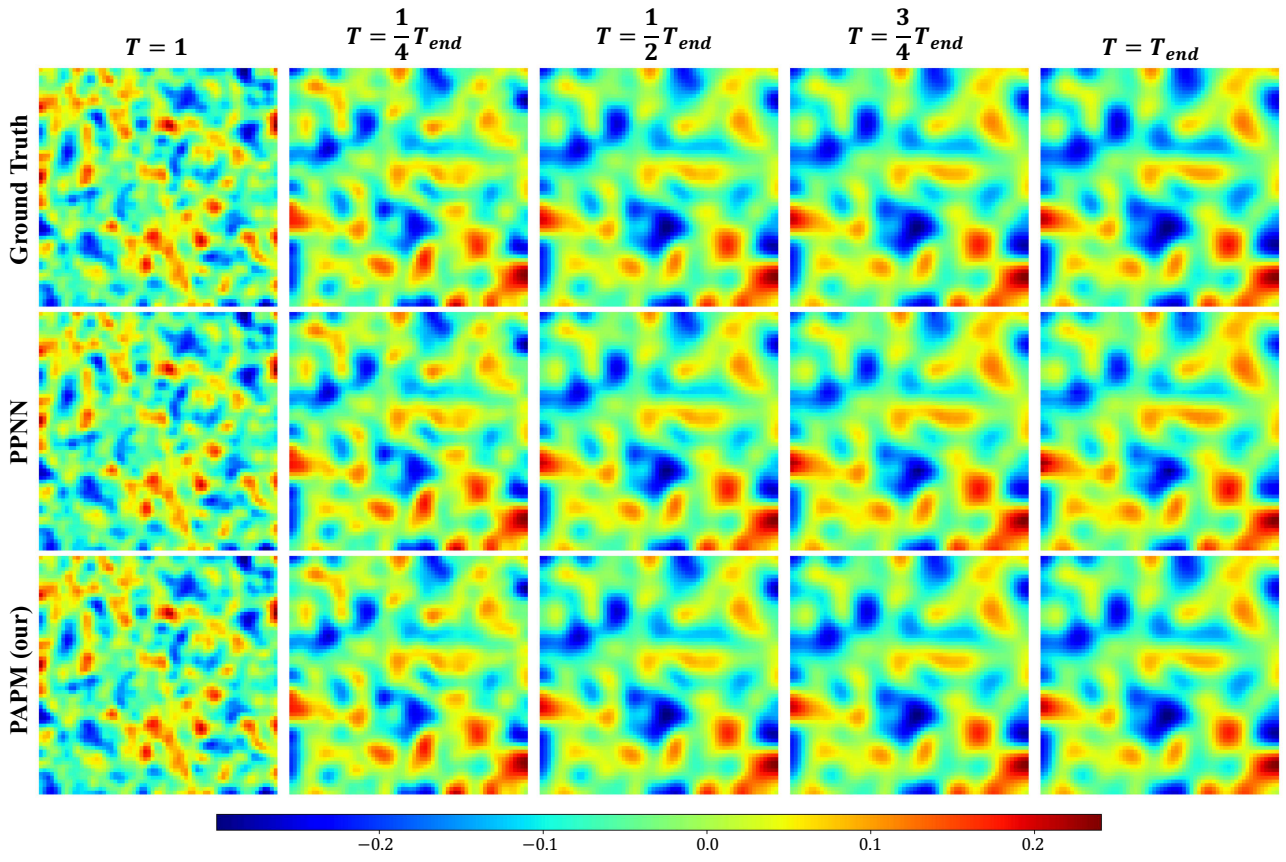

Figure 8. Predicted flow velocity ($\|\boldsymbol{u}\|_2$) snapshots by PPNN, and PAPM (Ours) vs. Ground Truth (GT) on RD2d dataset in T Ext. task.

G.2. Training and inference time cost

Dataset generation for our work is notably resource-intensive, with inference costs ranging from $10^2 \sim 10^4$ s for public datasets and up to 10^3 s for those we generated using COMSOL Multiphysics®. As detailed in Tab. 10, in stark contrast, both baselines and PAPM register inference times between $0.1 \sim 10$ s, achieving an improvement of 3 to 5 orders of magnitude. Notably, PAPM's time cost rivals or even surpasses baselines across different datasets. Unlike traditional numerical algorithms, this indicates that introducing rigorous physical mechanisms doesn't necessarily bloat time costs. PAPM's efficiency remains competitive with other data-driven methods.

G.3. Supplemental Ablation studies

G.3.1. The comparison between fixed and trainable convolutional kernels

In PAPM, the pre-defined convolutional kernels contain the fixed and trainable ones. For the fixed ones, we can directly use the difference scheme as the parameter of the convolution kernel without participating in the subsequent training optimization. For the trainable one, the trainable parameters in kernels are constrained as the triangular and symmetric matrices, which correspond to unidirectional convection and directionless diffusion. The use of trainable ones is due to spatio-temporal discretization, where the original difference scheme is insufficient in characterizing the corresponding gradient information. Thus, we make the kernel in the difference scheme in Diffusive Flows (DF) and Convective Flows (CF) learnable for better capturing spatial gradients.

Taking the Burgers2d and RD2d datasets in the C Int. setting as examples to show the different performance for fixed and trainable convolutional kernels. Tab. 11 indicates that when the kernel becomes a learnable term, the model's performance greatly improves, where the metric is the mean L_2 relative error (Eq. 4), lower values indicate better results, and **Bold** indicates the best performance.

Table 10. Training and inference time cost (epoch/second) of different baselines.

Config	Burgers2d		RD2d		NS2d		Lid2d		NSM2d	
	Train	Infer	Train	Infer	Train	Infer	Train	Infer	Train	Infer
ConvLSTM	5.41	1.12	21.41	3.94	7.05	0.86	**8.68**	3.22	4.39	0.92
Dil-ResNet	6.64	1.73	27.06	4.06	9.96	1.19	10.90	3.66	6.34	1.03
time-FNO2D	4.87	1.46	8.94	1.95	5.16	0.79	10.41	2.11	**3.35**	0.69
MIONet	5.69	1.58	8.69	2.03	5.05	0.89	10.54	3.02	4.03	0.76
U-FNet	3.64	**0.52**	14.56	1.96	6.67	0.51	10.42	1.14	6.96	0.82
CNO	4.02	0.60	15.72	2.28	4.92	**0.44**	11.08	**1.12**	5.90	0.68
PeRCNN	5.02	1.72	**5.73**	1.47	6.53	0.84	17.44	4.08	4.24	0.82
PPNN	5.07	0.96	8.88	**1.19**	4.87	0.91	15.58	3.44	8.08	**0.64**
PAPM	**3.44**	0.93	8.62	2.07	**3.70**	1.27	8.91	2.94	5.13	0.88

Table 11. Performance comparison for different configurations.

config	Burgers2d	RD2d
fixed	0.082	0.049
trainable	**0.039**	**0.018**

G.3.2. Ablation Studies of Different Terms

Here, we present a series of ablation experiments conducted on the Burgers2d, RD2d, and NSM2d datasets within the C Int. setting. Each experiment investigates the impact of excluding specific terms: **no_DF**, which omits diffusion; **no_CF**, which excludes convection; **no_Phy**, which removes both diffusion and convection; **no_IST**, which excludes internal sources; **no_EST**, which excludes external sources; and **no_BCs**, which eliminates the explicit embedding of boundary conditions.

We can get the following insights as shown in Tab. 12, where the metric is the mean L_2 relative error (Eq. 4), lower values indicate better results, and **Bold** indicates the best performance. **First**, for Burgers2d and NSM2d, the no_DF configuration demonstrated the importance of integrating the viscosity coefficient with the diffusion term, with its absence leading to significant errors. **Second**, for Burgers2d and RD2d, internal sources primarily drive the state update process. In NSM2d, both the internal source (gradient of pressure, ∇p) and the external source (time-invariant magnetic intensity) play crucial roles. **Third**, the necessity of adhering to physical laws in boundary conditions was highlighted in the no_BCs, notably reducing errors.

Table 12. Performance metrics across different datasets with various modifications.

Datasets	ϵ	no_DF	no_CF	no_Phy	no_IST	no_EST	no_BCs
Burgers2d	**0.039**	0.067	0.062	0.149	0.174	-	0.068
RD2d	**0.018**	0.102	-	0.102	0.281	-	0.083
NSM2d	**0.189**	0.273	0.212	0.299	0.392	0.311	0.201

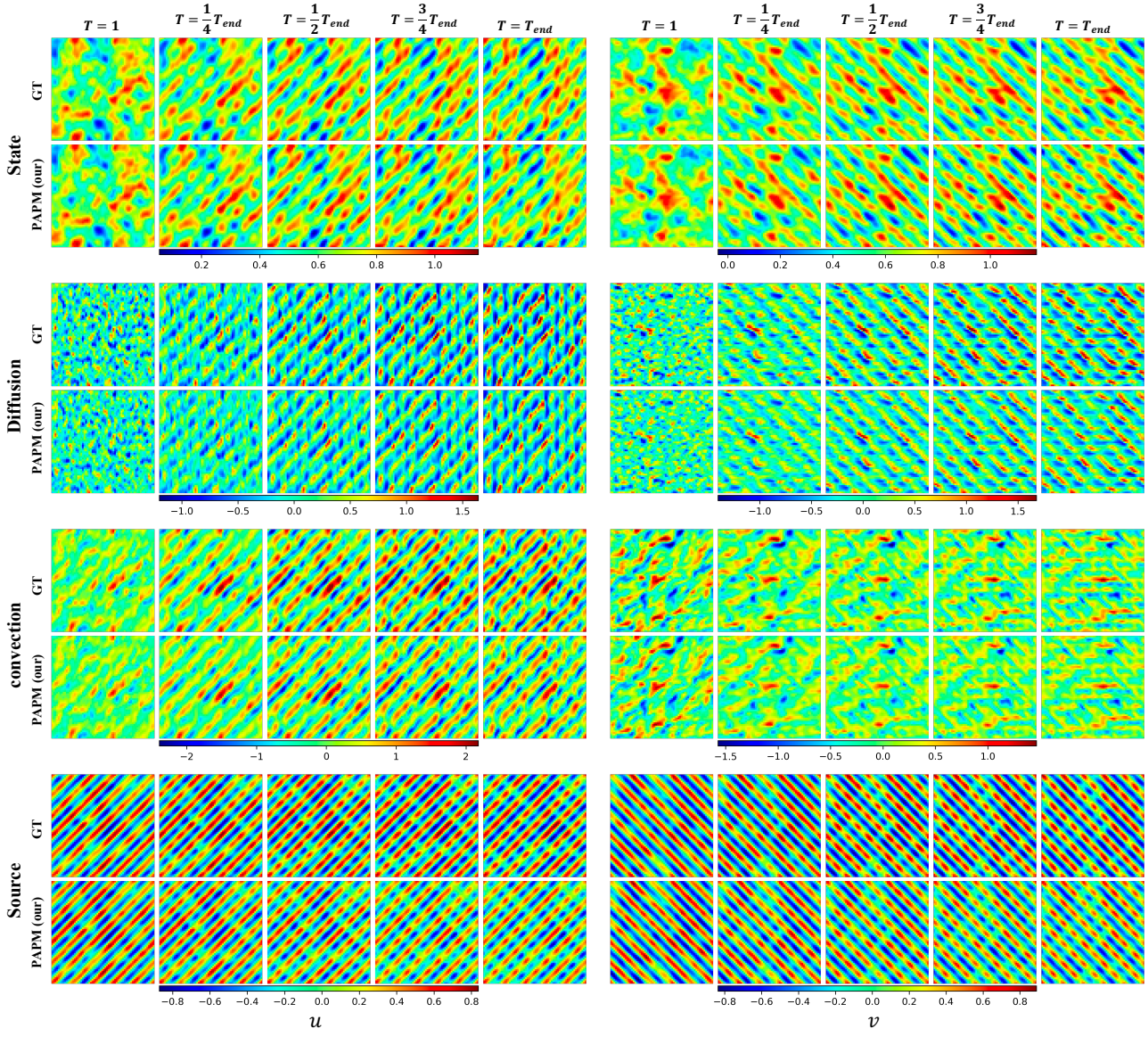

Figure 9. Different terms on the Burgers2d dataset in T Ext. task.

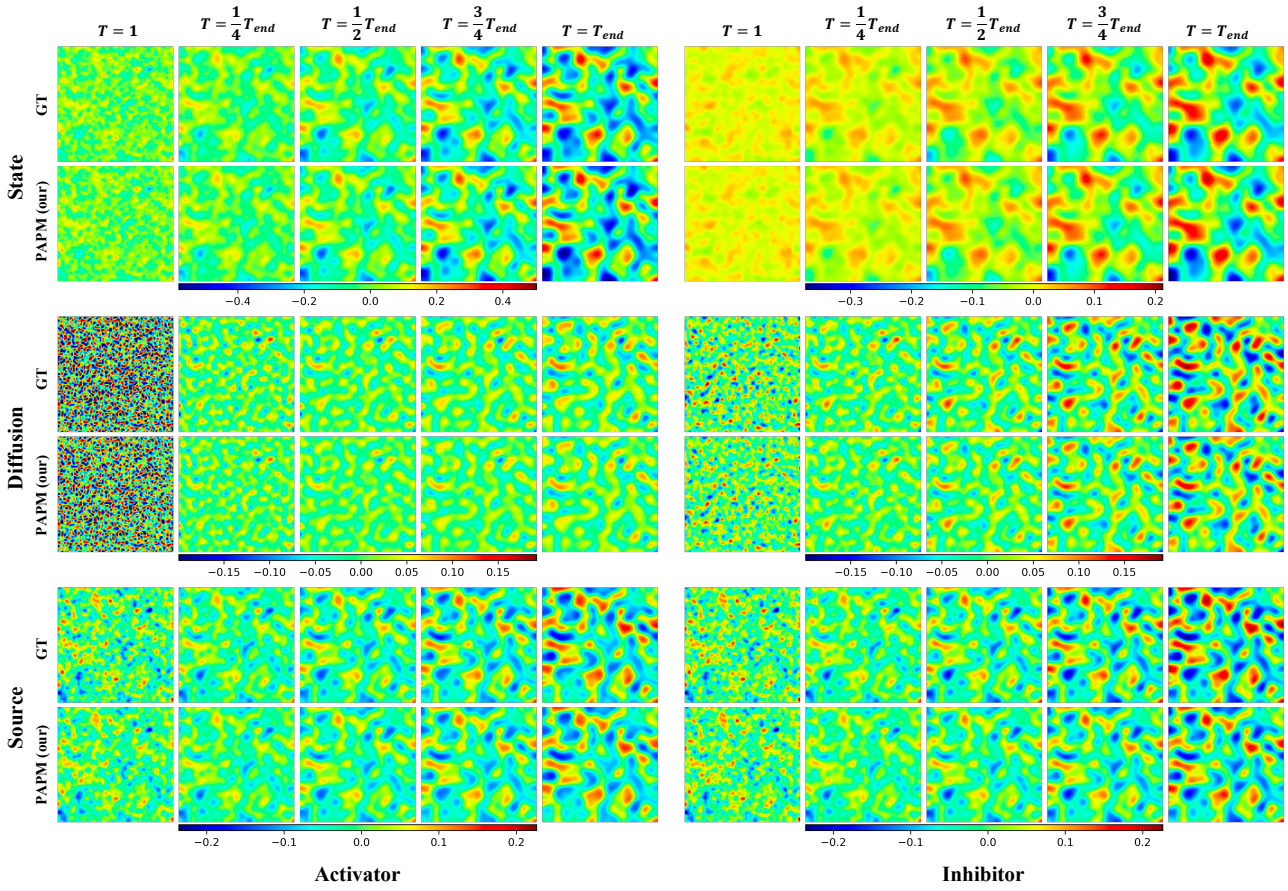

Figure 10. Different terms on the RD2d dataset in T Ext. task.