Graph Neural PDE Solvers with Conservation and Similarity-Equivariance

Masanobu Horie [1]   Naoto Mitsume [2]

Abstract

Utilizing machine learning to address partial differential equations (PDEs) presents significant challenges due to the diversity of spatial domains and their corresponding state configurations, which complicates the task of encompassing all potential scenarios through data-driven methodologies alone. Moreover, there are legitimate concerns regarding the generalization and reliability of such approaches, as they often overlook inherent physical constraints. In response to these challenges, this study introduces a novel machine-learning architecture that is highly generalizable and adheres to conservation laws and physical symmetries, thereby ensuring greater reliability. The foundation of this architecture is graph neural networks (GNNs), which are adept at accommodating a variety of shapes and forms. Additionally, we explore the parallels between GNNs and traditional numerical solvers, facilitating a seamless integration of conservative principles and symmetries into machine learning models. Our findings from experiments demonstrate that the model's inclusion of physical laws significantly enhances its generalizability, i.e., no significant accuracy degradation for unseen spatial domains while other models degrade. The code is available at https://github.com/yellowshippo/fluxgnn-icml2024.

1. Introduction

Predicting physical phenomena, often described by partial differential equations (PDEs), has become a focal point in research due to its relevance across various fields such as product design, disaster prevention, and environmental sciences. The integration of machine learning in this domain has been the subject of active investigation recently. This is
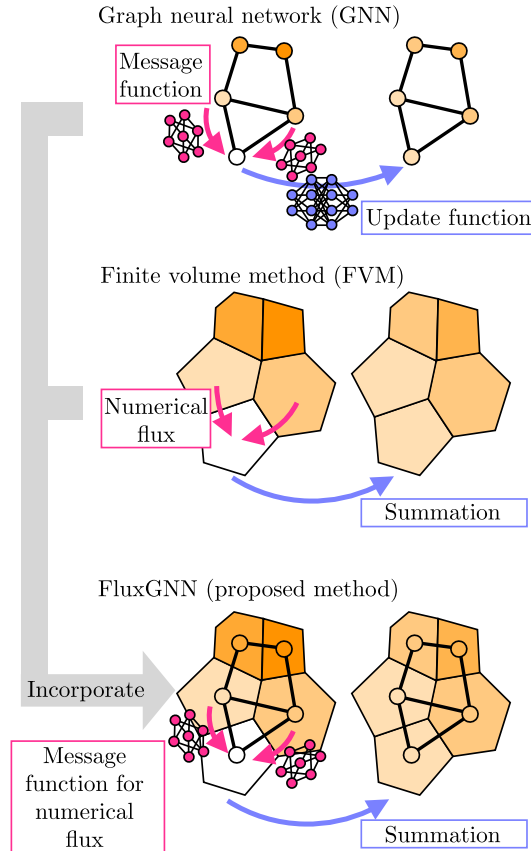

Figure 1. Overview of a typical graph neural network (GNN), finite volume method (FVM), and proposed model, FluxGNN. Our method combines GNN and FVM, realizing high expressibility from GNN and generalizability from FVM.

largely because machine learning has the potential to expedite prediction processes by effectively leveraging existing datasets (Ladický et al., 2015; Kochkov et al., 2021; Pichi et al., 2024). Moreover, when trained with actual measured data, these models can achieve enhanced prediction accuracy (Lam et al., 2023). Additionally, such models are increasingly being utilized for optimization and solving inverse problems through gradient-based optimization techniques (Chang & Cheng, 2020; Miao & Chen, 2023), facilitated by the efficiency of modern deep learning frameworks in automatic differentiation.

Despite these advancements, challenges in generalizability and reliability persist in machine learning applications

[1]RICOS Co. Ltd., Tokyo, Japan [2]Graduate School of Science and Technology, University of Tsukuba, Ibaraki, Japan. Correspondence to: Masanobu Horie <horie@ricos.co.jp>.

within physics. The diversity of spatial analysis domains and the myriad of potential states within these domains pose significant difficulties in encompassing all scenarios using solely data-driven approaches. Out-of-domain generalization is challenging in machine learning, leading to reduced generalizability in data-driven models (Bonfanti et al., 2023). These models often struggle as surrogate models for classical solvers. Moreover, machine learning methods frequently contravene critical physical laws such as conservation and symmetry, resulting in problematic decision-making processes and less reliable predictions. For instance, in predicting the movement of greenhouse gases on Earth, it is imperative to account for the conservation law of gases, which dictates that the total amount of gas should remain constant barring any generation or absorption.

Incorporating physical laws into machine learning models is a potent strategy to overcome challenges related to generalizability and reliability (Ling et al., 2016; Wang et al., 2021; Huang et al., 2022; Li et al., 2023). Physical systems exhibit symmetries; they remain invariant under transformations such as rotation and scaling. Leveraging these symmetries can enhance a model's generalizability, as an unseen input may be regarded as a familiar one when transformed. Moreover, embedding physical laws as inductive biases in a model can improve the reliability of its predictions, ensuring adherence to these laws.

In this study, we introduce the concept of flux-passing graph neural networks (*FluxGNNs*), which are designed to integrate conservation laws and physical symmetries pertaining to similarity transformations—including translation, rotation, reflection, and scaling—as inductive biases. This approach is grounded in the parallels between graph neural networks (GNNs), which are adaptable to arbitrary graphs, and the finite volume method (FVM), a classical numerical analysis method that inherently respects conservation and symmetries (Figure 1). This integration results in a model that not only generalizes effectively across spatial domains but also rigorously upholds conservation laws and symmetries regarding similarity transformations. The FluxGNNs represent a natural extension of the FVM, essentially encompassing a pure FVM framework enhanced with automatic differentiation capabilities, thanks to its implementation in a deep learning framework. Additionally, we delineate the mathematical conditions for typical GNNs to serve as conservative models.

Our method is designed to seamlessly integrate with existing machine learning techniques. It can accommodate methods such as scaling invariance (Wang et al., 2021), temporal bundling (Brandstetter et al., 2022), and boundary encoder and neural nonlinear solver (Horie & Mitsume, 2022), requiring minimal or no modifications to ensure adherence to conservation laws and symmetries. Additionally, our ap-

proach capitalizes on the geometric attributes of the input discretized shapes, such as meshes, by utilizing elements such as surface areas and normal vectors. This integration strengthens the connection between scaling symmetry and fundamental concepts of numerical analysis such as the Courant number. Our experimental results indicate that the proposed method shows high generalizability and improves the speed–accuracy tradeoff of prediction.

The main contributions of our study are summarized below.

- We mathematically reveal the conditions for typical GNNs to achieve conservations, resulting in the construction of locally conservative GNNs that can be applied to arbitrary graphs.

- We construct FluxGNNs based on the locally conservative GNNs. The proposed method can be applied to arbitrary analysis domains and satisfy conservation laws rigorously, in addition to achieving similarity-equivariance while focusing on a close relationship with a classical numerical analysis method.

- We provide methods to incorporate existing machine learning approaches into our model and demonstrate that they can satisfy conservation laws and symmetries under our modifications, if any.

- Finally, we demonstrate that the proposed method shows high generalizability for unseen spatial domains because of the included physical laws.

2. Background and Related Work

2.1. Classical PDE Solvers and FVM

We focus on a class of PDEs called conservation form in a n-dimensional domain $\Omega \in \mathbb{R}^n$ expressed as

$$\frac{\partial}{\partial t}\boldsymbol{u} = -\nabla \cdot \boldsymbol{F}(\boldsymbol{u}) + \boldsymbol{S}, \qquad (1)$$

where $\mathcal{U} \ni \boldsymbol{u} : [0, \infty) \times \Omega \to \mathbb{R}^d$, $\boldsymbol{F} : \mathcal{U} \to \mathcal{U}$, and $\boldsymbol{S}$ represent an unknown d-dimensional vector field to be solved, a known operator that may be nonlinear and contain spatial differential operators, and a known source term, respectively. This formulation indicates that $\boldsymbol{u}$ is conservative if $\boldsymbol{S} = \boldsymbol{0}$ (LeVeque, 1992), and therefore, it is called the conservation form. Although this form looks specific, various PDEs such as the convection–diffusion equation and Navier–Stokes equations can be expressed as conservation forms. For example, in the case of the diffusion equation with diffusion coefficient D, $\boldsymbol{F}_{\text{diffusion}}(u) := D\nabla u$, where u represents a scalar field ($d = 1$ in this case).

FVM[1] spatially discretizes the equation using meshes,

---
[1]Although many variants exist for FVM, we introduce a funda-

which are discretized representations of the analysis domains. By applying spatial integration over the i-th cell (with volume V_i) in a mesh and Stokes' theorem, Equation (1) becomes

$$\frac{\partial}{\partial t} \int_{V_i} \boldsymbol{u} dV = -\int_{\partial V_i} \boldsymbol{F}(\boldsymbol{u}) \cdot \boldsymbol{n} dS + \int_\Omega \boldsymbol{S} dV, \quad (2)$$

where ∂V_i and $\boldsymbol{n} : \partial V_i \to \mathbb{R}^n$ represent the boundary of the i-th cell and the unit normal vector field pointing outside of the cell, respectively.

Using linear interpolation, Equation (2) can be approximated as

$$\frac{\partial}{\partial t} V_i \boldsymbol{u}_i = -\sum_{j \in \mathcal{N}_i} S_{ij} \boldsymbol{n}_{ij} \cdot [\boldsymbol{F}(\boldsymbol{u})]_{ij} + \boldsymbol{S}_i V_i, \quad (3)$$

where $\mathcal{N}_i$, $\boldsymbol{u}_i$, $\boldsymbol{S}_i$, S_{ij}, $\boldsymbol{n}_{ij}$, and $[\boldsymbol{F}(\boldsymbol{u})]_{ij}$ represent the set of neighboring cells to the i-th cell, value of $\boldsymbol{u}$ at the centroid of the i-th cell, value of $\boldsymbol{S}$ at the centroid of the i-th cell, area of the face (i, j), normal vector of the face (i, j) pointing outside of the i-th cell, and value of $\boldsymbol{F}(\boldsymbol{u})$ at the face (i, j), referred to as *numerical flux*, respectively. Extant studies have developed various computational methods for the numerical flux because the values at the faces need to be estimated using the ones at the cell centers. The conservation law is expressed as $\int_\Omega \boldsymbol{u} dV \approx \sum_i V_i \boldsymbol{u}_i = \text{Constant}$.

FVM holds local conservation, which is a significant property for solving conservation forms, i.e., the balance between the influx from the j-th cell to the i-th cell neighboring through the face (i, j), $\boldsymbol{\psi}_{ij} := S_{ij} \boldsymbol{n}_{ij} \cdot [\boldsymbol{F}(\boldsymbol{u})]_{ij}$, and the opposite, $\boldsymbol{\psi}_{ji}$, expressed as

$$\boldsymbol{\psi}_{ji} = S_{ji} \boldsymbol{n}_{ji} \cdot [\boldsymbol{F}(\boldsymbol{u})]_{ji} = -S_{ij} \boldsymbol{n}_{ij} \cdot [\boldsymbol{F}(\boldsymbol{u})]_{ij} = -\boldsymbol{\psi}_{ij}, \quad (4)$$

using the relationships $S_{ij} = S_{ji}$, $[\boldsymbol{F}(\boldsymbol{u})]_{ij} = [\boldsymbol{F}(\boldsymbol{u})]_{ji}$, and $\boldsymbol{n}_{ij} = -\boldsymbol{n}_{ji}$. Because $\boldsymbol{\psi}_{ij} + \boldsymbol{\psi}_{ji} = \boldsymbol{0}$, interchanging the physical quantity $\boldsymbol{u}$ through face (i, j) does not change the total integral of $\boldsymbol{u}$, and therefore, the global conservation laws hold if $\boldsymbol{S}_i = \boldsymbol{0}$ ($\forall i \in \mathcal{V}$). This property inspired us to develop the proposed method that has local conservation.

2.2. Machine Learning Models for Conservation Laws

Driven by the critical need for conservation, extensive research has focused on exploring approaches to realize conservation using machine learning. For example, Matsubara et al. (2020) included structure-preserving numerical methods into their models to obtain conservations in Hamiltonian systems. Further, Richter-Powell et al. (2022) leveraged physics-informed neural networks to parametrize the vector

potential, thereby generating a divergence-free, i.e., conservative, vector field. Hansen et al. (2024) focused on FVM and demonstrated its effectiveness in one-dimensional datasets. These approaches provided insights to achieve conservation and focused on regular or fixed domains; however, they did not investigate irregular domains and spatial extrapolation, which are the main focuses of this study.

2.3. Machine Learning Models for Symmetries

Equivariance can be incorporated into machine learning models to obtain models that respect symmetries. Equivariance for a group G is expressed as $f : X \to Y$ satisfying $f(g \cdot x) = g \cdot f(x)$ for any $g \in G$ and $x \in X$, assuming that G acts on X and Y. Equivariance for $\mathrm{E}(n)$, i.e., translation, rotation, and reflection, or $SE(n)$, i.e., $E(n)$ without reflection, has been investigated in various works (Thomas et al., 2018; Fuchs et al., 2020; Satorras et al., 2021; Eijkelboom et al., 2023) and found to be effective in machine learning tasks related to geometry and physics. For example, Horie et al. (2021) realized efficient $\mathrm{E}(n)$-equivariant GNNs by using a message-passing method inspired by a classical PDE solver and applying simple $\mathrm{E}(n)$-equivariant multilayer perceptrons (MLPs) expressed as $\boldsymbol{f}_{\mathrm{E}(n)}(\boldsymbol{h}) := \mathrm{MLP}(\|\boldsymbol{h}\|)\boldsymbol{h}$, where $\boldsymbol{h}$ and $\|\boldsymbol{h}\|$ represent an input to the layer and its L^2-norm, respectively. Further, scale equivariance has been examined extensively in several studies (Worrall & Welling, 2019; Sosnovik et al., 2020; Yang et al., 2023). For example, Wang et al. (2021) used a simple formulation of scale-equivariant convolutional neural networks (CNNs) expressed as $\boldsymbol{f}_{\mathrm{scaling}}(\boldsymbol{h}) := \mathrm{CNN}(\boldsymbol{h}/\sigma)\sigma$, where $\sigma := \max[\boldsymbol{h}] - \min[\boldsymbol{h}]$ represents the scale of $\boldsymbol{h}$. Further, Wang et al. (2021) demonstrated that $\mathrm{E}(n)$- and scale- equivariance separately improved the predictive performance of machine learning models on fluid dynamics datasets. However, they did not construct a similarity-equivariant model, i.e., an equivariant model considering both $\mathrm{E}(n)$ and scaling. We successfully constructed and demonstrated the effectiveness of such a model in this study.

2.4. GNNs and Graph Neural PDE Solvers

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a tuple of the vertex set $\mathcal{V} \subset \mathbb{Z}$ and edge set $\mathcal{E} \subset \mathbb{Z} \times \mathbb{Z}$. GNNs should be able to handle arbitrary-length data because different graphs can have a different number of vertices and different edge connectivities. Consider a vertex signal $\boldsymbol{H} : \mathcal{V} \to \mathbb{R}^d$ that corresponds to vectors on the vertices of a graph. For a graph with three vertices, the vertex signal can be expressed as $\boldsymbol{H} := (\boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3)^\top$ where $\boldsymbol{h}_i \in \mathbb{R}^d$ represents a signal at the i-th vertex. A GNN $\mathcal{F} : \mathcal{H} \to \mathcal{H}'$ can be considered an operator converting a vertex signal to another, where $\mathcal{H}$ and $\mathcal{H}'$ denote sets of vertex signals. A message-passing neural network (MPNN) (Gilmer et al., 2017), which is a

---

mental formulation. For more details on FVM, refer to, e.g., Jasak (1996); Darwish & Moukalled (2016).

typical formulation of GNNs, is expressed as

$$m_{ij} = f_{\text{message}}(h_i, h_j, h_{ij}),$$

$$h_i^{\text{out}} = f_{\text{update}}\left(h_i, \sum_{j \in \mathcal{N}_i} m_{ij}\right), \quad (5)$$

where h_{ij} and $\mathcal{N}_i$ represent a signal on the edge (i, j) and a set of vertices neighboring the i-th vertex, respectively. MPNN comprises two neural networks: a message function f_{message} expressing the interaction between vertices and an update function f_{update} describing the vertex-wise updates of features.

In the context of spatiotemporal discretization, a typical neural PDE solver is formulated as $\mathcal{F} : \mathcal{H} \ni H^t \mapsto H^{t+1} \in \mathcal{H}$, where its objective to predict the time series of the vertex signal H^t. Given that practical physical simulations often involve irregular spatial data, such as point clouds and meshes, which can naturally be represented as graphs, GNNs emerge as a natural choice for constructing neural PDE solvers (Chang & Cheng, 2020; Sanchez-Gonzalez et al., 2020; Pfaff et al., 2021; Lam et al., 2023).

Notably, Brandstetter et al. (2022) have proposed effective methods for learning and predicting time-dependent PDEs, drawing inspiration from classical PDE solvers. Among their techniques, temporal bundling stands out, as it involves predicting states at multiple time steps through a single forward computation of a model followed by the application of 1D CNN layers in the temporal direction. This approach results in significantly enhanced computational efficiency.

Furthermore, Horie & Mitsume (2022) have developed PDE solvers based on $\mathrm{E}(n)$-equivariant GNNs, as initially proposed by Horie et al. (2021). Their work is characterized by a rigorous treatment of boundary conditions and an efficient time evolution computation method inspired by nonlinear optimization techniques. They introduced boundary encoders, a concept in which encoders for boundary conditions are combined with those for the corresponding input vertex signals. This allows for the encoding of boundary conditions within the same space as vertex signals. The proposed time evolution method is expressed as

$$H^{[i+1]} := H^{[i]} - \alpha_{\text{BB}}^{[i]}\left[H^{[i]} - H^{[0]} - \mathcal{F}(H^{[i]})\Delta t\right], \quad (6)$$

where $\cdot^{[i]}$, $H^{[0]} = H(t)$, and $\alpha_{\text{BB}}^{[i]}$ represent the state at the i-th nonlinear optimization step, a known graph signal at time t, and a coefficient determined using the Barzilai–Borwein method (Barzilai & Borwein, 1988), which contains global pooling operations, respectively. Most existing graph neural PDE solvers use the encode-process-decode architecture proposed by Battaglia et al. (2018), wherein input features are embedded into a higher dimensional space

using the encoder, message passing is computed in the encoded space, and encoded features are converted into an output space using the decoder, as done in this study.

3. Proposed Method

Here, we introduce the proposed method, FluxGNN. We first describe an abstract and general formulation of locally conservative GNN, then build FluxGNN on top of that, incorporating the essence of FVM. Proofs of all theorems and lemmas claimed in this section can be found in Appendix C.

3.1. Locally Conservative GNN

In this study, we focus on a set of undirected graphs, i.e., $(i, j) \in \mathcal{E}$ implies $(j, i) \in \mathcal{E}$ for any $i, j \in \mathcal{V}$. This framework is employed to represent the exchange of physical quantities between vertices. The condition for a conservative MPNN is outlined as follows:

Theorem 3.1. *An MPNN $\mathcal{F} : \mathcal{H} \to \mathcal{H}$, formulated in Equation (5) with continuous activation functions, exhibits conservation properties for any graphs and vertex signals (i.e., $\sum_{i \in \mathcal{V}} h_i$ remains constant for a given graph) if and only if the following conditions are satisfied.*

$$m_{ij} = -m_{ji}, \quad \forall(i, j) \in \mathcal{E} \quad (7)$$

$$f_{\text{update}}\left(h_i, \sum_{i \in \mathcal{N}_i} m_{ij}\right) = h_i + \sum_{i \in \mathcal{N}_i} m_{ij}. \quad (8)$$

We categorize GNNs that meet the criteria outlined in Theorem 3.1 as *locally conservative GNNs*. The manner in which information is exchanged in these GNNs bears resemblance to the concept of numerical flux in the FVM, specifically concerning the preservation of local conservation principles, as illustrated in Equation (4). The proof of this theorem necessitates the introduction of the following lemma.

Lemma 3.2. *A vertex-wise continuous update function should be linear to achieve conservation.*[2]

Following this lemma, the condition for the conservative encode-process-decode architecture can be obtained as

Theorem 3.3. *An encode-process-decode architecture in the form $\mathcal{F}_{\text{e-p-d}} := \mathcal{F}_{\text{decode}} \circ \mathcal{F}_{\text{L}}^{(N)} \circ \cdots \circ \mathcal{F}_{\text{L}}^{(1)} \circ \mathcal{F}_{\text{encode}}$ with locally conservative GNNs $\mathcal{F}_{\text{L}}^{(i)}$, a vertex-wise continuous encoder $\mathcal{F}_{\text{encode}}$, and a decoder $\mathcal{F}_{\text{decode}}$ is conservative if and only if the encoder is linear and the decoder is the pseudoinverse (more specifically, left inverse) of the encoder, i.e., $\mathcal{F}_{\text{decode}} \circ \mathcal{F}_{\text{encode}} = \text{id}$, where id is the identity map.*

---

[2]One can construct a conservative continuous nonlinear function despite the lemma, unless vertex-wise. For example, the following $\bar{\mathcal{F}}$ is conservative. $\bar{\mathcal{F}}(H) := \mathcal{F}(H)[\sum_{i \in \mathcal{V}} H / \sum_{i \in \mathcal{V}} \mathcal{F}(H)]$. However, these are not locally conservative in general, and therefore, we omit these nonlinearities for generalization in this study.

Interestingly, while Horie & Mitsume (2022) proposed the use of pseudoinverse decoders to satisfy boundary conditions, our approach adopts them primarily to ensure conservation. Nevertheless, we can also utilize pseudoinverse decoders to fulfill boundary conditions in our context. In summary, locally conservative GNNs are characterized by their nonlinearity being confined to the message passing functions, without involving update ones.

3.2. FluxGNN

Based on the formulation of the FVM (Equation (3)) and the locally conservative GNN (Equations (7) and (8)), we construct a class of GNN layers $\mathcal{F}_{\mathrm{Flux}}$ expressed as

$$V_i[\mathcal{F}_{\mathrm{Flux}}(\boldsymbol{H})]_i := V_i \boldsymbol{h}_i - \sum_{j \in \mathcal{N}_i} S_{ij} \boldsymbol{n}_{ij} \cdot \boldsymbol{F}_{\mathrm{ML}}\left(\boldsymbol{h}_i, \boldsymbol{h}_j, \boldsymbol{h}_{ij}\right),$$
(9)

where $\boldsymbol{F}_{\mathrm{ML}}$ and $\boldsymbol{h}_{ij}$ represent a machine learning model constructed using neural networks and a feature at the face (i, j) corresponding to an edge feature in MPNN, respectively. Further, vertices correspond to the cells of the mesh, as shown in Figure 1. We refer to $\boldsymbol{F}_{\mathrm{ML}}$ as a *flux function* because it is a message function for the numerical flux.

Theorem 3.4. *A machine learning model defined in Equation (9) with a symmetric edge signal $\boldsymbol{h}_{ij} = \boldsymbol{h}_{ji}$ is a locally conservative GNN if and only if a flux function $\boldsymbol{F}_{\mathrm{ML}}$ is a permutation-invariant function for the first and second arguments, i.e., $\boldsymbol{F}_{\mathrm{ML}}(\boldsymbol{h}_i, \boldsymbol{h}_j, \boldsymbol{h}_{ij}) = \boldsymbol{F}_{\mathrm{ML}}(\boldsymbol{h}_j, \boldsymbol{h}_i, \boldsymbol{h}_{ij})$*

We define models $\mathcal{F}_{\mathrm{Flux}}$ described in Equation (9) as *FluxGNNs*, when the condition outlined in Theorem 3.4 is satisfied, with encoders and decoders also adhering to the criteria specified in Theorem 3.3. FluxGNNs are designed to model numerical fluxes using machine learning techniques, hence the name. The attainment of the former condition can be realized through the utilization of deep sets, as presented in (Zaheer et al., 2017), which offer the necessary and sufficient class of permutation-invariant functions. FluxGNN exhibits, at a minimum, the same level of expressive power as the FVM concerning information propagation in space. This equivalence is attributed to the presence of the formulation of spatial discretization in FVM (Equation (3)) within Equation (9). Further, FluxGNN demonstrates similarity equivariance when $\boldsymbol{F}_{\mathrm{ML}}$ exhibits such properties. It is because other components, such as area S_{ij}, volume V_i, and normal vector $\boldsymbol{n}_{ij}$, are considered geometrical quantities, and therefore, exhibit similarity-equivariance.

3.3. Incorporation of Existing Methods

FluxGNN is designed with a level of simplicity that allows for compatibility with numerous existing methods. In this section, we explore how to integrate certain aspects of these

methods while maintaining conservation. All operations are vertex-wise, and therefore, we write $\boldsymbol{h}$ instead of $\boldsymbol{h}_i$ for the i-th vertex feature ($i \in \mathcal{V}$) for simplicity.

3.3.1. SIMILARITY-EQUIVARIANT MLPS

Similarity-equivariant MLPs can be achieved by combining the existing $\mathrm{E}(n)$-equivariant layer and scale-equivariant layer as

$$\boldsymbol{f}_{\mathrm{sim}}(\boldsymbol{h}) := \mathrm{MLP}(\|\boldsymbol{h}/\sigma\|)\boldsymbol{h},$$
(10)

where $\sigma > 0$ is determined using local quantities, such as spatial resolution Δx and temporal resolution Δt, to make the input physical quantities dimensionless. We avoid using global operators, such as $\min$ and $\max$ used in Wang et al. (2021), to retain the localness of our model. Further, our choice of σ has a strong connection with central concepts in classical PDE solvers. For example, assuming that the input feature is in the dimension of velocity, our similarity equivariant layer turns to

$$\mathrm{MLP}\left(\left\|\frac{\boldsymbol{h}}{\Delta x/\Delta t}\right\|\right)\boldsymbol{h} = \mathrm{MLP}(C)\boldsymbol{h},$$
(11)

where C represents the Courant number that characterizes the stability of the simulation. This formulation is natural because a condition with a different Courant number can require a different treatment for efficient computations because of a different numerical stability. In the case of the diffusion phenomena, the direct input to MLP turns to the diffusion number, which is another essential quantity for numerical stability. In addition, Huang et al. (2022) suggest a similar form to our formulation, showing its universality. That justifies our choice of the similarity-equivariant MLPs. Using similarity-equivariant MLPs as building blocks, one can construct flux functions as presented in Appendix D.2.

3.3.2. TEMPORAL BUNDLING

Due to the constraints imposed by encoders and decoders on conservation, it is necessary to modify the temporal bundling method. In scenarios where we predict K steps simultaneously using a single forward computation of a FluxGNN, the approach involves dividing the weight matrices of the encoder and decoder into K distinct segments. Each of these segments is specifically applied to encode or decode features. Subsequently, we concatenate these processed features in the feature direction, as follows: $\mathrm{Concat}\left[\boldsymbol{W}^{(1)}\boldsymbol{h}, \boldsymbol{W}^{(2)}\boldsymbol{h}, \dots, \boldsymbol{W}^{(K)}\boldsymbol{h}\right]$. Once the decoding process is completed using the pseudoinverse decoder, the next step involves the application of multiple linear 1D-CNN layers. One such layer, exemplifying this approach, can be represented in the following form: $\boldsymbol{h}_{\mathrm{out}}^{(t)} := (1-w)\boldsymbol{h}^{(t-1)} + w\boldsymbol{h}^{(t)}$, where w represents a trainable parameter and $\boldsymbol{h}^{(t)}$ denotes the outputs of decoders at
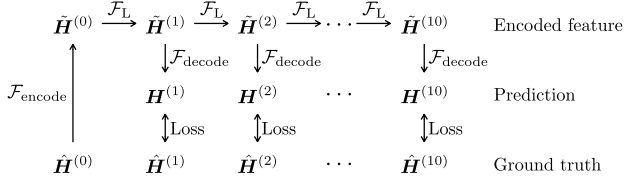
Figure 2. Overview of the FluxGNN model for the convection–diffusion equation with an encoder $\mathcal{F}_{\text{encode}}$, locally conservative GNN $\mathcal{F}_{\text{L}}$, and decoder $\mathcal{F}_{\text{decode}}$. The model outputs a time series, and the loss is computed using all steps of the model's output.

time t, to facilitate the temporal smoothness. This form differs from that proposed in Brandstetter et al. (2022) because we need this operation to be linear for conservation.

3.3.3. NEURAL NONLINEAR SOLVER AND BOUNDARY ENCODER

The neural nonlinear solver (Equation (6)) can be applied to FluxGNN without modification because it does not break conservation laws and symmetries. Although it contains global operations to compute $\alpha_{\text{BB}}^{[i]}$, the model remains locally conservative because it exists outside of MLPs. Further, boundary encoders do not need modifications because they are encoders, which are linear in our formulation.

4. Experiments

We showcase the outcomes of our experiments, demonstrating that our proposed method achieves conservation and similarity-equivariance with notable accuracy and computational efficiency. We have implemented all our models using PyTorch 1.9.1 (Paszke et al., 2019). The code is available at `https://github.com/yellowshippo/fluxgnn-icml2024`. For a comprehensive overview and detailed insights into the experiments, please refer to Appendices E and F.

4.1. Convection–Diffusion Equation

In the current set of experiments, we demonstrate the expressive capabilities of the core FluxGNN model using the convection–diffusion equation, a fundamental model for the transport of conservative quantities. The equation for a conservative scalar field u is expressed as

$$\frac{\partial}{\partial t} u = -\nabla \cdot (\boldsymbol{c} u - D \nabla u), \qquad (12)$$

where $\boldsymbol{c}$ and $D \geq 0$ represent a known velocity field and known diffusion coefficient, respectively.

For our datasets, we generated 100 trajectories for training, 10 for validation, and 10 for testing. These were derived using the exact solution of the equation, with random variations in uniform velocity u from 0.0 to 0.2, and in the
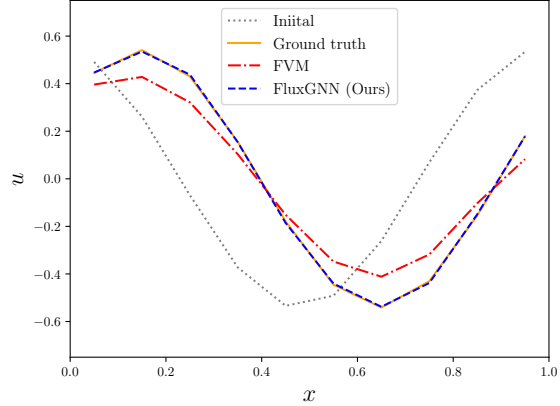

Figure 3. Comparison of the initial condition, ground truth, prediction of FVM, and prediction of FluxGNN taken from a sample in the test dataset at time $t = 1.0$.

amplitude and phase of the sinusoidal initial condition. We set the diffusion coefficient $D = 10^{-4}$, corresponding to convection-dominant problems, which are generally more challenging than diffusive ones. The spatial and temporal resolution parameters were set to $\Delta x = 0.1$ and $\Delta t = 0.1$, respectively, and the analysis domain length was one unit.

We constructed a straightforward FluxGNN model with autoregressive time series modeling, as illustrated in Figure Figure 2. The model generates time series data from given initial conditions and velocities. It was trained on a CPU (Intel Xeon CPU E5-2695 v2 @ 2.40 GHz) for 3 hours, using MSE loss and an Adam optimizer (Kingma & Ba, 2014). As a baseline, we established an FVM solver making all trainable functions identity ones, enabling most implementations to be shared between the solver and FluxGNN. This setup highlights the enhancements brought by machine learning. The accuracy of our FVM implementation was verified across multiple problems (Appendix D.1).

Figure 3 and Table 1 present the results of the experiments in comparison with the ground truth and the FVM solver. The conservation error is defined by $\int_{t \in (0, T_{\max}]} \left[\int_{\Omega} (\boldsymbol{H}(t) - \boldsymbol{H}(t=0)) dV \right] dt$, where $T_{\max} = 1.0$. Our model exhibited higher accuracy than the FVM at the same spatiotemporal resolution. Notably, the FVM tends to exhibit increased diffusivity due to numerical diffusion from the low spatial resolution. Moreover, FluxGNN maintained a conservation property almost equivalent to that of the FVM, indicating that the neural networks incorporated in the model do not disrupt the conservation law. Consequently, we conclude that our machine learning model possesses superior expressive power and can leverage existing data to achieve higher accuracy compared to classical methods.
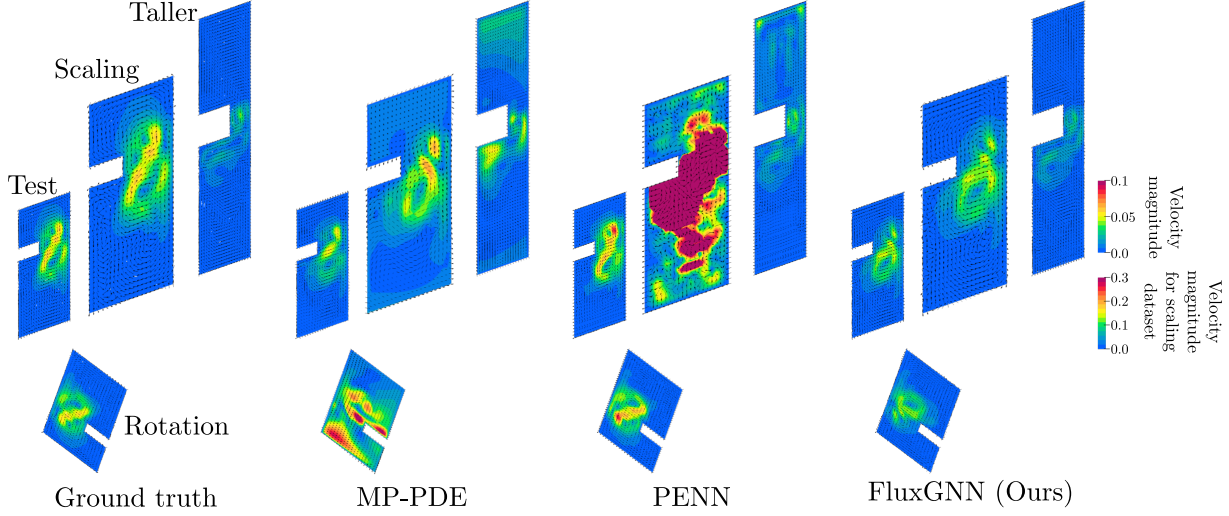
Figure 4. Visual comparison of the velocity field between the ground truth, MP-PDE, PENN, and FluxGNN.

Table 1. MSE loss ($\pm$ standard error of the mean) on the test dataset of the convection–diffusion equation.

Method	Loss $\left(\times 10^{-5}\right)$	Conservation error $\left(\times 10^{-8}\right)$
FVM	1408.47 ± 32.75	1.63 ± 0.19
FluxGNN (Ours)	$\mathbf{3.08} \pm 0.08$	1.75 ± 0.32

4.2. Navier–Stokes Equations with Mixture

To evaluate the practical applicability of our model, we conducted tests on a more complex problem, integrating existing methods into FluxGNN. This scenario leads to intricate interactions between velocity $\boldsymbol{u}$, pressure p, and density ρ. These interactions are expressed as follows

$$\frac{\partial}{\partial t}(\rho \boldsymbol{u}) = - \nabla \cdot \left[\rho \boldsymbol{u} \otimes \boldsymbol{u} - \mu(\nabla \otimes \boldsymbol{u} + (\nabla \otimes \boldsymbol{u})^{\top})\right]$$
$$- \nabla p_{\rho g y} - \boldsymbol{g} \cdot \boldsymbol{y} \nabla \rho, \qquad (13)$$

$$\frac{\partial}{\partial t}\alpha = -\nabla \cdot \left[\boldsymbol{u}\alpha - D_\alpha \nabla \alpha\right], \qquad (14)$$

where μ and α represent the dynamic viscosity, volume fraction of the low-density fluid, respectively. α satisfies the relationship $\rho = \alpha \rho_{\text{low}} + (1 - \alpha)\rho_{\text{high}}$ (ρ_{low} and ρ_{high} denote scalar values of low and high density, respectively). Furthermore, D_α, $\boldsymbol{g}$, $\boldsymbol{y}$, and $p_{\rho g y} = p - \rho \boldsymbol{g} \cdot \boldsymbol{y}$ represent the diffusion coefficient for α, gravity acceleration, height in the direction of the gravity, and pressure without the effect of gravitational potential, respectively.

We generated 200 trajectories for training, 8 for validation, and 8 for the test dataset. These trajectories encompassed various shapes and initial conditions, and they were generated using OpenFOAM,[3] a widely recognized classical PDE

solver. We generated five different shapes for training, two for validation, and two for testing, with variations in the dimensions characterizing each shape. To ensure realism, we employed random numerical analysis to create the initial conditions for each shape. The primary objective of the machine learning task was to predict ground truth trajectories projected onto domains with lower spatiotemporal resolution. To achieve this, we set the time step $\Delta t = 0.2$ with the maximum time reaching 1.6 for training and 3.2 for evaluation, allowing us to assess the model's ability for temporal extrapolation. In addition to these datasets, we generated two supplementary datasets, each comprising eight trajectories. These datasets were created by applying rotation and scaling operations to the test dataset, respectively (named *rotation* and *scaling* datasets). Further, to test the model's generalizability, we generated a dataset featuring three shapes that were taller than the test samples (named *taller* dataset) to validate the model's generalizability.

We developed FluxGNN models following the autoregressive approach detailed in Section 4.1, but with the incorporation of components from existing research. These included similarity-equivariant MLPs, temporal bundling, neural nonlinear solvers, and boundary encoders. The model inputs were aligned with those used in classical numerical analysis, including mesh geometry, initial conditions, boundary conditions, and material properties. The model outputs were time series data for velocity ($\boldsymbol{u}$), pressure ($p_{\rho g y}$), and volume fraction (α) fields. We employed RMSE loss for each physical quantity, weighted by their respective standard deviations, to balance the loss magnitudes across different physical quantities.

For comparison, we selected MP-PDE (Brandstetter et al., 2022) and PENN (Horie & Mitsume, 2022) as baseline
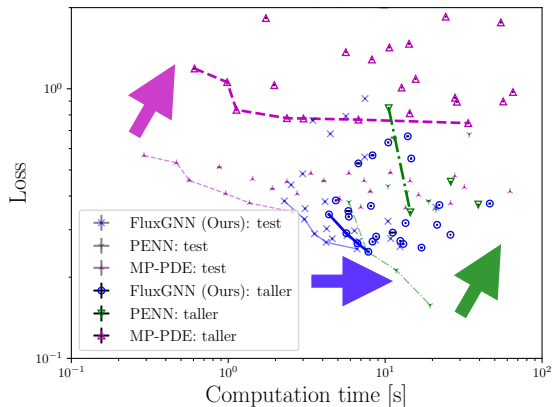
---

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

Figure 5. Speed–accuracy tradeoff obtained through hyperparameter studies for MP-PDE, PENN, and FluxGNN (proposed method), with error bars corresponding to the standard error of the mean. Light and dark colors correspond to the evaluation of the test and taller datasets, respectively. Lines represent Pareto fronts, with arrows denoting shifts of the fronts caused by changes in the dataset considered for the evaluation. The results of FVM are excluded because they are far from the Pareto front and are shown in Figure 15.

models, representing state-of-the-art machine learning approaches that accommodate irregular domains and boundary conditions. That comparison highlighted our model's superiority because we incorporated these methods into ours. Additionally, we included our FVM implementation to examine the enhanced expressibility achieved through machine learning, similar to the approach in Section 4.1. All machine learning models were trained on GPUs (NVIDIA A100 80GB PCIe) over a period of three days.

Figure 4 and Table 2 present both qualitative and quantitative comparisons for each method. All machine learning models demonstrate excellent performance, indicating the success of the training process. Furthermore, all models exhibit the ability to predict time series longer than those observed during training. The proposed method demonstrates a high level of conservation, even when incorporating components from existing machine learning methods. In contrast, other baseline machine learning models do not exhibit the same level of conservation, highlighting the need for special treatment to adhere to conservation laws. When considering the rotation dataset, PENN performs admirably due to its $E(n)$-equivariance. However, the MP-PDE model begins to degrade under these conditions. On the other hand, PENN displays divergent behavior when applied to the scaling dataset, primarily because of the significant changes in input values, leading to extrapolation. In contrast, FluxGNN exhibits nearly identical performance before and after scaling, showcasing its scale equivariance. Nevertheless, we observe a slight performance variation attributed to numerical errors accumulated during autoregressive computation.

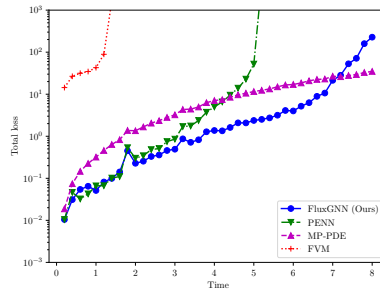Our method performs well on the taller dataset despite no

Figure 6. Time evolution of total loss for FVM, MP-PDE, PENN, and the proposed method, FluxGNN. The vertical axis is in log scale.
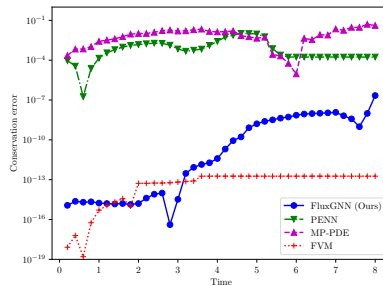
Figure 7. Time evolution of conservation error for FVM, MP-PDE, PENN, and the proposed method, FluxGNN. The vertical axis is in log scale.

scaling, which implies that conservation is essential for realizing generalizability because PENN degrades its performance even with $E(n)$-equivariance and shows the spurious emergence of velocity on the top of the taller sample. Further, our model leverages the expressive power of neural networks because FVM tends to be divergent and never reaches convergence on the taller dataset for all tested settings. Therefore, we claim that FluxGNN incorporates methods of the classical PDE solver and machine learning at a high level, realizing high generalizability and expressibility.

Figure 5 presents the speed–accuracy tradeoff obtained using machine learning models with various hyperparameters on one core of the same CPU as Section 4.1. All models have no clear advantage on the test dataset. However, FluxGNN has a clear advantage compared to the baseline models when evaluated on the taller dataset. Moreover, the Pareto front of our model does not shift significantly in the direction of the loss, which means that FluxGNN generalizes for spatial extrapolation without losing accuracy.

We performed prediction using each method for even further temporal rollout. Figure 6 presents the time evolution of total loss. FluxGNN exhibits the lowest loss until time

Table 2. RMSE loss and conservation error ($\pm$ standard error of the mean) on the evaluation datasets of the Navier–Stokes equation with mixture. Each metric is normalized using standard deviation.

Method	Dataset	Loss $\boldsymbol{u}$ $\left(\times 10^{-1}\right)$	Loss p $\left(\times 10^{-1}\right)$	Loss α $\left(\times 10^{-1}\right)$	Conservation error α $\left(\times 10^{-5}\right)$
FVM	test	19.501 ± 0.149	37.706 ± 0.194	2.896 ± 0.030	$\mathbf{0.01} \pm 0.00$
MP-PDE	test	1.532 ± 0.010	0.941 ± 0.008	1.021 ± 0.010	1001.95 ± 120.43
PENN	test	$\mathbf{0.619} \pm 0.005$	$\mathbf{0.598} \pm 0.005$	$\mathbf{0.358} \pm 0.005$	1356.62 ± 283.00
FluxGNN (Ours)	test	1.202 ± 0.008	1.143 ± 0.008	$\mathbf{0.349} \pm 0.005$	0.06 ± 0.03
FVM	rotation	18.363 ± 0.128	22.366 ± 0.123	2.930 ± 0.030	$\mathbf{0.01} \pm 0.00$
MP-PDE	rotation	11.523 ± 0.075	10.883 ± 0.078	9.528 ± 0.072	5307.85 ± 1009.04
PENN	rotation	$\mathbf{0.622} \pm 0.004$	$\mathbf{0.592} \pm 0.005$	$\mathbf{0.355} \pm 0.005$	1302.98 ± 153.02
FluxGNN (Ours)	rotation	1.207 ± 0.007	1.175 ± 0.008	0.351 ± 0.005	$\mathbf{0.01} \pm 0.00$
FVM	scaling	19.411 ± 0.149	38.106 ± 0.193	2.918 ± 0.030	$\mathbf{0.01} \pm 0.00$
MP-PDE	scaling	5.072 ± 0.034	4.211 ± 0.040	5.604 ± 0.046	6401.93 ± 1806.50
PENN	scaling	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
FluxGNN (Ours)	scaling	$\mathbf{1.219} \pm 0.009$	$\mathbf{1.228} \pm 0.008$	$\mathbf{0.356} \pm 0.005$	0.05 ± 0.01
FVM	taller	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN	NaN $\pm$ NaN
MP-PDE	taller	8.377 ± 0.081	2.472 ± 0.024	2.823 ± 0.033	16770.80 ± 3293.59
PENN	taller	1.476 ± 0.016	1.887 ± 0.014	0.357 ± 0.006	4799.89 ± 1574.43
FluxGNN (Ours)	taller	$\mathbf{1.184} \pm 0.009$	$\mathbf{0.966} \pm 0.008$	$\mathbf{0.337} \pm 0.006$	$\mathbf{0.02} \pm 0.00$

$t \approx 7.0$. After that, it starts to show a higher loss than MP-PDE due to numerical instability. Figure 7 indicates that the proposed method demonstrates a high level of conservation. Even after instability stands out, the conservation error of our method is lower than other machine learning methods by orders of magnitude. The results imply the potential of FluxGNN for a more extended temporal rollout. We did not include the pushforward trick proposed by Brandstetter et al. (2022), which realizes a more stable prediction for time series prediction. Incorporating that trick into our method may address this point. In addition, incorporating flux and slope limiters, known as effective methods for stability in a computational physics domain, may also help. The visualization of longer rollout predictions is shown in Appendix F.5.

Appendix F.6 elaborates on an ablation study of the FluxGNN model. That suggests that all components included in the model contribute to spatial out-of-domain generalizability.

5. Conclusion

We presented FluxGNNs, which combines the expressibility of GNNs and the generalizability of FVM. Our method shows high generalizability for spatial domain extrapolation, where other machine learning models exhibit performance degradation. This study focused on PDEs corresponding to flow and transport phenomena; however, it can be applied to a broader class of problems, e.g., optimal transport and traffic flow on graphs because the locally conservative GNNs proposed in this study have a generic structure.

Although our model shows high expressibility and generalizability, we did not demonstrate performance analysis

comparing well-optimized classical solvers, which is a limitation of the work. However, we demonstrated a clear improvement compared to FVM, which shared most of the implementation with our model. Therefore, one possible next step would be implementing FluxGNN on top of the well-optimized solvers, which may drastically improve speed and accuracy, as demonstrated in the experiments.

Another limitation is that the method needs to be more stable for long-term prediction. This point may be addressed by incorporating flux and slope limiters, known as effective methods for stability in a computational physics domain. We leave it as a future work because our focus of the present work is to construct a reliable method towards spatially out-of-domain generalization.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Barzilai, J. and Borwein, J. M. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

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

Bonfanti, A., Santana, R., Ellero, M., and Gholami, B. On the hyperparameters influencing a PINN's generalization beyond the training domain. *arXiv preprint arXiv:2302.07557*, 2023.

Brandstetter, J., Worrall, D. E., and Welling, M. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vSix3HPYKSU.

Chang, K.-H. and Cheng, C.-Y. Learning to simulate and design for structural engineering. In *International Conference on Machine Learning*, pp. 1426–1436. PMLR, 2020.

Darwish, M. and Moukalled, F. *The finite volume method in computational fluid dynamics: an advanced introduction with OpenFOAM® and Matlab®*. Springer, 2016.

Eijkelboom, F., Hesselink, R., and Bekkers, E. J. E (n) equivariant message passing simplicial networks. In *International Conference on Machine Learning*, pp. 9071–9081. PMLR, 2023.

Fuchs, F., Worrall, D., Fischer, V., and Welling, M. SE(3)-transformers: 3D roto-translation equivariant attention networks. *Advances in Neural Information Processing Systems*, 33, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272. JMLR. org, 2017.

Hansen, D., Maddix, D. C., Alizadeh, S., Gupta, G., and Mahoney, M. W. Learning physical models that can respect conservation laws. *Physica D: Nonlinear Phenomena*, 457:133952, 2024. ISSN 0167-2789. doi: https://doi.org/10.1016/j.physd.2023.133952. URL https://www.sciencedirect.com/science/article/pii/S0167278923003068.

Horie, M. and Mitsume, N. Physics-embedded neural networks: Graph neural PDE solvers with mixed boundary conditions. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=B3TOg-YCtzo.

Horie, M., Morita, N., Hishinuma, T., Ihara, Y., and Mitsume, N. Isometric transformation invariant and equivariant graph convolutional networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=FX0vR39SJ5q.

Huang, W., Han, J., Rong, Y., Xu, T., Sun, F., and Huang, J. Equivariant graph mechanics networks with constraints. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=SHbhHHfePhP.

Jasak, H. Error analysis and estimation for the finite volume method with applications to fluid flows. 1996.

Kannappan, P. *Functional equations and inequalities with applications*. Springer Science & Business Media, 2009.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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.

Ladický, L., Jeong, S., Solenthaler, B., Pollefeys, M., and Gross, M. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)*, 34(6): 1–9, 2015.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., et al. Learning skillful medium-range global weather forecasting. *Science*, pp. eadi2336, 2023.

Lax, P. and Wendroff, B. Systems of conservation laws. *Communications on Pure and Applied Mathematics*, 13(2):217–237, 1960. doi: https://doi.org/10.1002/cpa.3160130205. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.3160130205.

LeVeque, R. J. *Numerical methods for conservation laws*, volume 214. Springer, 1992.

Li, Q., Wang, T., Roychowdhury, V., and Jawed, M. K. Rapidly encoding generalizable dynamics in a euclidean symmetric neural network. *Extreme Mechanics Letters*, 58:101925, 2023.

Ling, J., Kurzawski, A., and Templeton, J. Reynolds averaged turbulence modelling using deep neural networks

with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

Matsubara, T., Ishikawa, A., and Yaguchi, T. Deep energy-based modeling of discrete-time physics. *Advances in Neural Information Processing Systems*, 33:13100–13111, 2020.

Miao, Z. and Chen, Y. Vc-pinn: Variable coefficient physics-informed neural network for forward and inverse problems of pdes with variable coefficient. *Physica D: Nonlinear Phenomena*, 456:133945, 2023.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=roNqYL0_XP.

Pichi, F., Moya, B., and Hesthaven, J. S. A graph convolutional autoencoder approach to model order reduction for parametrized pdes. *Journal of Computational Physics*, pp. 112762, 2024.

Prabhune, B. C. and Suresh, K. A fast matrix-free elasto-plastic solver for predicting residual stresses in additive manufacturing. *Computer-Aided Design*, 123:102829, 2020.

Richter-Powell, J., Lipman, Y., and Chen, R. T. Neural conservation laws: A divergence-free perspective. *Advances in Neural Information Processing Systems*, 35: 38075–38088, 2022.

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.

Satorras, V. G., Hoogeboom, E., and Welling, M. E (n) equivariant graph neural networks. In *International conference on machine learning*, pp. 9323–9332. PMLR, 2021.

Sosnovik, I., Szmaja, M., and Smeulders, A. Scale-equivariant steerable networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJgpugrKPS.

Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., and Riley, P. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

Versteeg, H. and Malalasekera, W. Computational fluid dynamics. *The finite volume method*, pp. 1–26, 1995.

Wang, R., Walters, R., and Yu, R. Incorporating symmetry into deep dynamics models for improved generalization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=wta_8Hx2KD.

Worrall, D. and Welling, M. Deep scale-spaces: Equivariance over scale. *Advances in Neural Information Processing Systems*, 32, 2019.

Yang, Y., Dasmahapatra, S., and Mahmoodi, S. Scale-equivariant unet for histopathology image segmentation. *arXiv preprint arXiv:2304.04595*, 2023.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Advances in Neural Information Processing Systems*, 30, 2017.

A. Notation

Ω	Analysis domain
$\mathcal{U} : [0, \infty) \times \Omega \to \mathbb{R}^d$	Time-dependent vector field
V_i	Volume of the i-th cell
$S_{ij} = S_{ji}$	Area of the face (i, j)
$\boldsymbol{n}_{ij} = -\boldsymbol{n}_{ji}$	Normal vector of the face (i, j), pointing outside of the i-th cell
$\boldsymbol{x}_i$	Position of the centroid of the i-th cell
$\boldsymbol{x}_{ij} = \boldsymbol{x}_{ji}$	Position of the centroid of the face (i, j)
$\mathcal{V} \subset \mathbb{Z}$	Vertex set
$\mathcal{E} \subset \mathbb{Z} \times \mathbb{Z}$	Edge set
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Graph
$\mathcal{N}_i \subset \mathcal{V}$	Set of neighboring vertices regarding the i vertex
$\mathcal{H}$	Set of graph signals
$\boldsymbol{H} : \mathcal{V} \to \mathbb{R}^d$	Graph signal
$\boldsymbol{h}_i := [\boldsymbol{H}]_i$	Value of a graph signal at the i-th vertex
$\mathcal{F} : \mathcal{H} \to \mathcal{H}'$	Operator mapping from a vertex signal to another, typically constructed using GNNs

B. Basics on Finite Volume Method

We review how to model the linear convection–diffusion equation of a scalar field expressed as

$$\frac{\partial}{\partial t} u = -\nabla \cdot (\boldsymbol{c} u - D \nabla u), \tag{15}$$

because it is easy to generalize the scheme to vector or higher-order tensor fields.

Here, we consider two neighboring cells, i and j, assuming $\boldsymbol{n}_{ij}$ and $\boldsymbol{x}_i - \boldsymbol{x}_j$ are in the same direction, and the midpoint is at the center of the face (i, j), i.e., $(\boldsymbol{x}_i + \boldsymbol{x}_j)/2 = \boldsymbol{x}_{ij}$, for simplicity. The setting is illustrated in Figure 8. In case the assumption does not hold, refer to Jasak (1996); Versteeg & Malalasekera (1995); Darwish & Moukalled (2016).

B.1. Spatial discretization

Applying Equation (3) to Equation (15), we obtain

$$\frac{\partial}{\partial t} V_i u_i = \sum_{j \in \mathcal{N}_i} S_{ij} \boldsymbol{n}_{ij} \cdot [\boldsymbol{c} u - D \nabla u]_{ij} \tag{16}$$

$$\approx \sum_{j \in \mathcal{N}_i} S_{ij} \boldsymbol{n}_{ij} \cdot [\boldsymbol{c}_{ij} u_{ij} - D_{ij} [\nabla u]_{ij}], \tag{17}$$

where $\cdot_{ij}$ represents a value at the face (i, j).

To obtain $\boldsymbol{u}_{ij}$ for the convection term, one may use the central interpolation

$$u_{ij}^{\text{linear}} := \frac{\|\boldsymbol{x}_{ij} - \boldsymbol{x}_j\|}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|} u_i + \frac{\|\boldsymbol{x}_{ij} - \boldsymbol{x}_i\|}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|} u_j, \tag{18}$$

or the upwind interpolation

$$u_{ij}^{\text{upwind}} := \begin{cases} u_i & \text{if } \boldsymbol{c}_{ij} \cdot \boldsymbol{n}_{ij} \geq 0 \\ u_j & \text{else.} \end{cases} \tag{19}$$
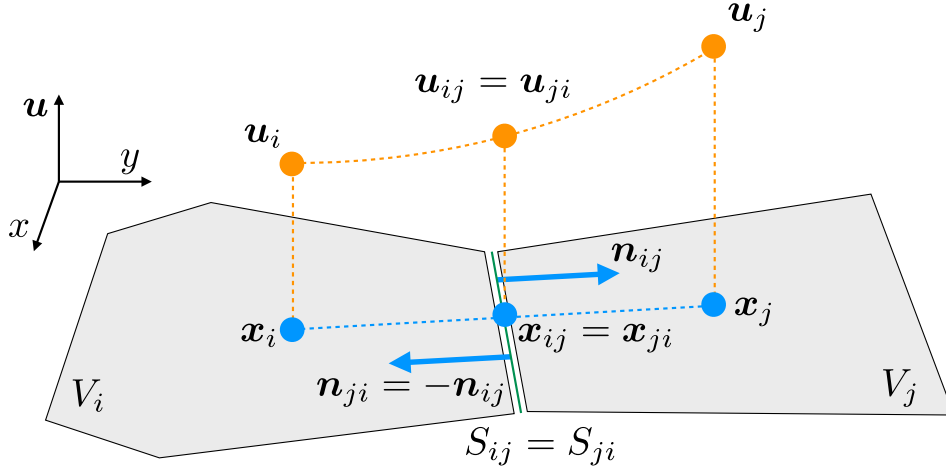
Figure 8. Geometry and variables used to construct FVM, focusing on the i-th and j-th cells.

There are extensive variations of the interpolation, such as the semi-Lagrangian method and Lax–Wendroff method (Lax & Wendroff, 1960). The gradient in the direction of $\boldsymbol{d}_{ij} := \boldsymbol{x}_j - \boldsymbol{x}_i$ for the diffusion term can be computed as

$$\left[\nabla_{\hat{\boldsymbol{d}}_{ij}} u\right]_{ij} \approx \frac{u_j - u_i}{\|\boldsymbol{d}_{ij}\|} \hat{\boldsymbol{d}}_{ij}, \tag{20}$$

where $\hat{\boldsymbol{d}}_{ij} := \boldsymbol{d}_{ij}/\|\boldsymbol{d}_{ij}\|$. Because $\boldsymbol{n}_{ij} = \hat{\boldsymbol{d}}_{ij}$ in the present setting, we obtain

$$\boldsymbol{n}_{ij} \cdot [\nabla u]_{ij} = \hat{\boldsymbol{d}}_{ij} \cdot \left[\nabla_{\hat{\boldsymbol{d}}_{ij}} u\right]_{ij} \approx \frac{u_j - u_i}{\|\boldsymbol{d}_{ij}\|}. \tag{21}$$

B.2. Boundary Condition

Boundary conditions are essential for solving PDEs because the behavior of the solution drastically differs depending on the conditions. We consider two types of boundary conditions: 1) the Dirichlet boundary condition defining the value of the solution on the boundary and 2) the Neumann boundary condition defining the value of the gradient of the solution on the boundary. These boundary conditions are expressed as

$$u(\boldsymbol{x}) = \hat{u}(\boldsymbol{x}), \text{ on } \partial\Omega_{\text{Dirichlet}} \tag{22}$$
$$\boldsymbol{n} \cdot [\nabla u](\boldsymbol{x}) = \hat{g}(\boldsymbol{x}), \text{ on } \partial\Omega_{\text{Neumann}}, \tag{23}$$

where $\hat{\cdot}$ represents a given value, and $\partial\Omega_{\text{Dirichlet}}$ and $\partial\Omega_{\text{Neumann}}$ represent Dirichlet and Neumann boundaries satisfying $\partial\Omega_{\text{Dirichlet}} \cap \partial\Omega_{\text{Neumann}} = \emptyset$ and $\partial\Omega_{\text{Dirichlet}} \cup \partial\Omega_{\text{Neumann}} = \partial\Omega$, respectively. Under spatial discretization, these conditions are

$$u_{ij} = \hat{u}_{ij}, \ \forall(i,j) \in \mathcal{E}_{\text{Dirichlet}} \tag{24}$$
$$\boldsymbol{n}_{ij} \cdot [\nabla u]_{ij} = \hat{g}_{ij}, \ \forall(i,j) \in \mathcal{E}_{\text{Neumann}}, \tag{25}$$

where $\mathcal{E}_{\text{Dirichlet}}$ and $\mathcal{E}_{\text{Neumann}}$ denote the sets of faces corresponding to the Dirichlet and Neumann boundaries, respectively. These expressions indicate that boundary conditions correspond to edge features in the GNN settings. Further, the j-th cell does not exist if the i-th cell exists and (i, j) is on the boundary. Here, j represents a virtual index to characterize boundary faces.

In the case of the convection term, the boundary conditions can be applied as

$$\begin{aligned} u_{ij} &\leftarrow \hat{u}_{ij}, & (i,j) \in \mathcal{E}_{\text{Dirichlet}} \\ u_{ij} &\leftarrow u_{ij} + \hat{g}_{ij}(\boldsymbol{x}_{ij} - \boldsymbol{x}_i) \cdot \boldsymbol{d}_{ij}, & (i,j) \in \mathcal{E}_{\text{Neumann}} \end{aligned} \tag{26}$$

13

The boundary conditions for the diffusion term are expressed as

$$\boldsymbol{n}_{ij} \cdot (\nabla u)_{ij} \leftarrow \frac{\hat{u}_{ij} - u_i}{\|\boldsymbol{x}_{ij} - \boldsymbol{x}_i\|}, \quad (i,j) \in \mathcal{E}_{\text{Dirichlet}}$$
$$\boldsymbol{n}_{ij} \cdot (\nabla u)_{ij} \leftarrow \hat{g}_{ij}, \qquad (i,j) \in \mathcal{E}_{\text{Neumann}}$$
(27)

where "$\leftarrow$" represents the variable overwriting during computation.

B.3. Temporal Discretization

We consider the temporal discretization for the general conservation form. To specify the time, we write variables explicitly as functions of time, e.g., $\boldsymbol{u}(t)$ denoting the vector field $\boldsymbol{u}$ at time t.

As is the case with spatial discretization, there are numerous methods for temporal discretization. One of the simplest schemes is the explicit Euler method expressed as

$$\boldsymbol{u}(t + \Delta t) = \boldsymbol{u}(t) - \nabla \cdot \boldsymbol{F}(\boldsymbol{u})(t)\Delta t,$$
(28)

where one can compute an unknown state $\boldsymbol{u}(t + \Delta t)$ by evaluating the right-hand side using a known state $\boldsymbol{u}(t)$. The other one is the implicit Euler method written as

$$\boldsymbol{u}(t + \Delta t) = \boldsymbol{u}(t) - \nabla \cdot \boldsymbol{F}(\boldsymbol{u})(t + \Delta t)\Delta t.$$
(29)

The implicit method contains unknowns on both sides, requiring more expensive computations to solve the equation compared to that required by the explicit method.

C. Proofs

C.1. Proof of Lemma 3.2

Proof. We show the linearity of update functions by contradiction. Assume that there exists a strictly nonlinear operator $\mathcal{F}_{\text{NL}}$ that is vertex-wise, continuous, and conservative. Here, one can let $\mathcal{F}_{\text{NL}}(\boldsymbol{0}) = \boldsymbol{0}$ without the loss of generality because of the shift-invariance of the input and output conservative fields. Now, consider a graph with two vertices and the vertex signal $\boldsymbol{H} = (\boldsymbol{h}_1, \boldsymbol{h}_2)^\top$. Applying the vertex-wise nonlinear operator to the signal, the converted conservation constant $\tilde{\boldsymbol{C}}$ can be expressed as

$$\mathcal{F}_{\text{NL}}(\boldsymbol{h}_1) + \mathcal{F}_{\text{NL}}(\boldsymbol{h}_2) = \tilde{\boldsymbol{C}}.$$
(30)

Converting a different input $(\boldsymbol{h}_1 + \boldsymbol{h}_2, \boldsymbol{0})^\top$ results in

$$\mathcal{F}_{\text{NL}}(\boldsymbol{h}_1 + \boldsymbol{h}_2) + \mathcal{F}_{\text{NL}}(0) = \tilde{\boldsymbol{C}}.$$
(31)

Using $\mathcal{F}_{\text{NL}}(\boldsymbol{0}) = \boldsymbol{0}$, one can find that

$$\mathcal{F}_{\text{NL}}(\boldsymbol{h}_1) + \mathcal{F}_{\text{NL}}(\boldsymbol{h}_2) = \mathcal{F}_{\text{NL}}(\boldsymbol{h}_1 + \boldsymbol{h}_2),$$
(32)

which results in the Cauchy's functional equation. The solution of the equation is known to be linear if $\mathcal{F}_{\text{NL}}$ is continuous (at least one point) (Kannappan, 2009), which is a contradiction. $\square$

Remark C.1. If a vertex-wise continuous conservative operator $\mathcal{F}$ is a map to the same space, $\mathcal{F}$ is an identity because $\mathcal{F}(\boldsymbol{h}_1)$ should be conservative in a graph with one vertex.

C.2. Proof of Theorem 3.1

Proof. By Remark C.1, the update function is the identity. Thus, MPNN is expressed as

$$\tilde{\boldsymbol{h}}_i = \boldsymbol{h}_i + \sum_{j \in \mathcal{N}_i} \boldsymbol{m}_{ij}.$$
(33)

The condition for conservation is $\sum_{i \in \mathcal{V}} h_i = \sum_{i \in \mathcal{V}} \tilde{h}_i$, which results in

$$\sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} m_{ij} = \mathbf{0}. \tag{34}$$

From the equation, $m_{ii} = \mathbf{0}$ for all $i \in \mathcal{V}$. Considering the complete graph with two vertices, we see $m_{12} = -m_{21}$ from Equation (34). Now, assume that the GNN is conservative for graphs with K edges. Adding the $(K+1)$-th edge (i, j) requires $m_{ij} = -m_{ji}$ to retain conservation. The converse can be verified easily. $\square$

C.3. Proof of Theorem 3.3

Proof. Following Lemma 3.2, encoders and decodes must be linear. We show that the decoder should be the left inverse of the encoder. Consider a graph with one vertex with the vertex signal $H = (h_1)^\top$. With that graph,

$$\sum_{i \in \mathcal{V}} h_i = h_1 = C, \tag{35}$$

where C represents a conservation constant. By applying a conservative encode-process-decode architecture $\mathcal{F}_{\text{decode}} \circ \mathcal{F}_{\text{L}} \circ \mathcal{F}_{\text{encode}}$,

$$\begin{aligned}
\mathcal{F}_{\text{decode}} \circ \mathcal{F}_{\text{L}} \circ \mathcal{F}_{\text{encode}}(u_i) &= \mathcal{F}_{\text{decode}} \circ \mathcal{F}_{\text{L}} \circ \mathcal{F}_{\text{encode}}(C) \\
&= \mathcal{F}_{\text{decode}} \circ \mathcal{F}_{\text{L}}(\tilde{C}) \\
&= \mathcal{F}_{\text{decode}}(\tilde{C}),
\end{aligned} \tag{36}$$

where $\tilde{C} := \mathcal{F}_{\text{encode}}(C)$ represents the conservation constant in the encoded space. Because of conservation, Equation (36) is equal to C. Therefore,

$$\mathcal{F}_{\text{decode}}(\tilde{C}) = \mathcal{F}_{\text{decode}}(\mathcal{F}_{\text{encode}}(C)) = C. \tag{37}$$

This relationship holds for any C. Therefore, $\mathcal{F}_{\text{decode}}$ should be the left inverse of $\mathcal{F}_{\text{encode}}$. $\square$

C.4. Proof of Theorem 3.4

Proof. First, the condition for update functions (Equation (8)) can be satisfied by defining $g_i := V_i h_i$. The message function of Equation (9) is expressed as

$$m_{ij} = -S_{ij} n_{ij} \cdot F_{\text{ML}} \left(\frac{1}{V_i} g_i, \frac{1}{V_j} g_j, \frac{1}{V_{ij}} g_{ij} \right), \tag{38}$$

where $V_{ij} = V_{ji}$ represents the interpolated volume at the face (i, j). Computing m_{ji} results in

$$m_{ji} = -S_{ji} n_{ji} \cdot F_{\text{ML}} \left(\frac{1}{V_j} g_j, \frac{1}{V_i} g_i, \frac{1}{V_{ji}} g_{ji} \right) \tag{39}$$

$$= S_{ij} n_{ij} \cdot F_{\text{ML}} \left(\frac{1}{V_j} g_j, \frac{1}{V_i} g_i, \frac{1}{V_{ij}} g_{ij} \right) \tag{40}$$

The condition for conservation (Equation (7)) requires $m_{ij} = -m_{ji}$ for all $(i, j) \in \mathcal{E}$, and therefore,

$$F_{\text{ML}} (h_i, h_j, h_{ij}) = F_{\text{ML}} (h_j, h_i, h_{ij}). \tag{41}$$

The converse can be shown similarly. $\square$

Remark C.2. With FluxGNN, $G := (g_1, g_2, \ldots, g_{|\mathcal{V}|})$ is conservative rather than H, which means that

$$\sum_{j \in \mathcal{V}} g_i = \sum_{j \in \mathcal{V}} V_i h_i \approx \int_\Omega h \, dV \tag{42}$$

is conservative as with FVM.
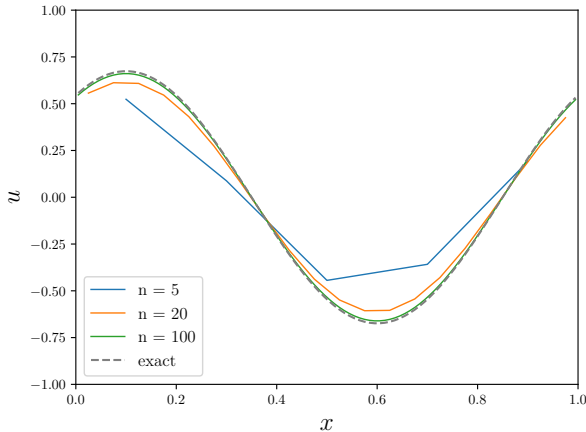
Figure 9. Results of the convergence study of FVM for the convection–diffusion equation at $t = 1.0$.

Table 3. Results of the convergence study of FVM for the convection–diffusion equation.

n_{cell}	Δx	RMSE
5	0.2	0.128
10	0.1	0.077
20	0.05	0.043
50	0.02	0.018
100	0.01	0.009

D. FluxGNN Details

D.1. Validation of FVM

Our model relies considerably on FVM implementation, and therefore, validating the implemented FVM is critical for successfully modeling FluxGNN. Here, we introduce the results of the validation.

CONVECTION–DIFFUSION EQUATION

We performed the study under the same condition as the machine learning task, i.e., periodic boundary condition with a sinusoidal initial condition. Table 3 and Table 3 present the results of the convergence study in the convection–diffusion equation. We confirmed that the convergence is in the second order for the spatial resolution, as expected from the formulation of FVM.

NAVIER–STOKES EQUATIONS

We performed a convergence study of the Navier–Stokes equation for the 2D Hagen–Poiseuille flow. We generated a rectangular shape with the length in the x and y directions of two and one, respectively. The inlet is on the side with the minimum x coordinate. The magnitude of inflow is one, and the Reynolds number is one. Under this setting, the x component of the velocity as a function of the y coordinate, $u_x(y)$, is given as

$$u_x(y) = 6y(1 - y). \tag{43}$$

Figure 10 and Table 4 present the results, which re-confirm the second-order convergence.

NAVIER–STOKES EQUATIONS WITH MIXTURE

We conducted a test of our FVM using the Navier–Stokes equations with mixture. The experimental setup closely mirrors that of the machine learning task, with the exception of the diffusion coefficient for α, denoted as D_α, which has been set to 10^{-4} for stability reasons. Constructing an exact solution for this class of problems is challenging, and therefore, we opted
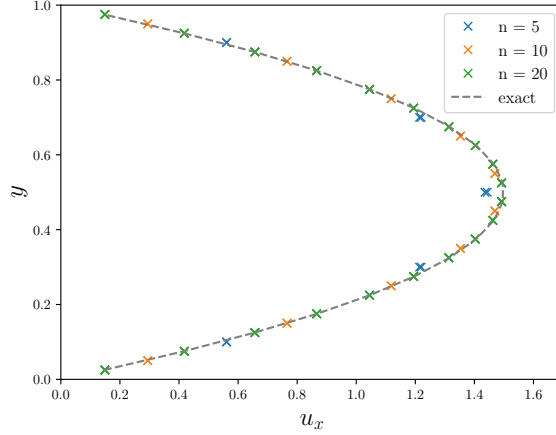
Figure 10. Results of the convergence study of FVM for the Navier–Stokes equations with the 2D Hagen—Poiseuille flow at the steady state.

Table 4. Results of the convergence study of FVM for the Navier–Stokes equations with the 2D Hagen—Poiseuille flow.

n_{cell} in the y direction	Δx	RMSE
5	0.2	0.041
10	0.1	0.010
20	0.05	0.003

to compare our numerical solution with that of OpenFOAM, a well-established classical solver that employs FVM. Figure 11 presents the qualitative comparison between OpenFOAM and our FVM implementation. While our implementation tends to exhibit slightly higher diffusivity at this resolution, both implementations display similar behavior.

D.2. Flux Function

Flux functions $\boldsymbol{F}_{\text{ML}}$ form the core of FluxGNNs. Here, we present definitions of the flux functions used for the following experiments. The flux function for convection, $\boldsymbol{F}_{\text{conv}}$, is defined as

$$\boldsymbol{F}_{\text{conv}}(\tilde{u}_i, \tilde{u}_j, \tilde{u}_{ij}, \tilde{\boldsymbol{c}}_{ij}) := \boldsymbol{f}_{\text{sim}}\left(\tilde{\boldsymbol{c}}_{ij} F_{\text{int}}(\tilde{u}_i, \tilde{u}_j, \tilde{u}_{ij})\right), \tag{44}$$

where $\boldsymbol{f}_{\text{sim}}$, $\tilde{\cdot}$, and F_{int} respectively represent a similarity-equivariant MLP defined in Equation (10), feature encoded using a linear map, and flux function for interpolation. F_{int} is expressed as

$$F_{\text{int}}(\tilde{u}_i, \tilde{u}_j, \tilde{u}_{ij}) := \frac{1}{4}\left[f_{\text{sim}}^{\text{vertex}}(\tilde{u}_i) + f_{\text{sim}}^{\text{vertex}}(\tilde{u}_j) + f_{\text{sim}}^{\text{linear}}(\tilde{u}_{ij}^{\text{linear}}) + f_{\text{upwind}}^{\text{vertex}}(\tilde{u}_{ij}^{\text{upwind}})\right], \tag{45}$$

where $\tilde{u}_{ij} = (\tilde{u}_{ij}^{\text{linear}}, \tilde{u}_{ij}^{\text{upwind}})$ is computed using FVM schemes explained in Appendix B. The similarity-equivariant MLP for vertex signals is shared to achieve permutation-equivariance for vertex signals. However, this does not apply to the MLPs for edge signals.

The gradient for the diffusion term is modeled as

$$\boldsymbol{F}_{\text{grad}}(\tilde{u}_i, \tilde{u}_j, \tilde{u}_{ij}) := \boldsymbol{f}_{\text{sim}}\left(\frac{\tilde{u}_j - \tilde{u}_i}{\|\boldsymbol{d}_{ij}\|} \frac{\boldsymbol{d}_{ij}}{\|\boldsymbol{d}_{ij}\|}\right). \tag{46}$$

The boundary conditions are applied using the same procedure as Appendix B; however, in the encoded space.
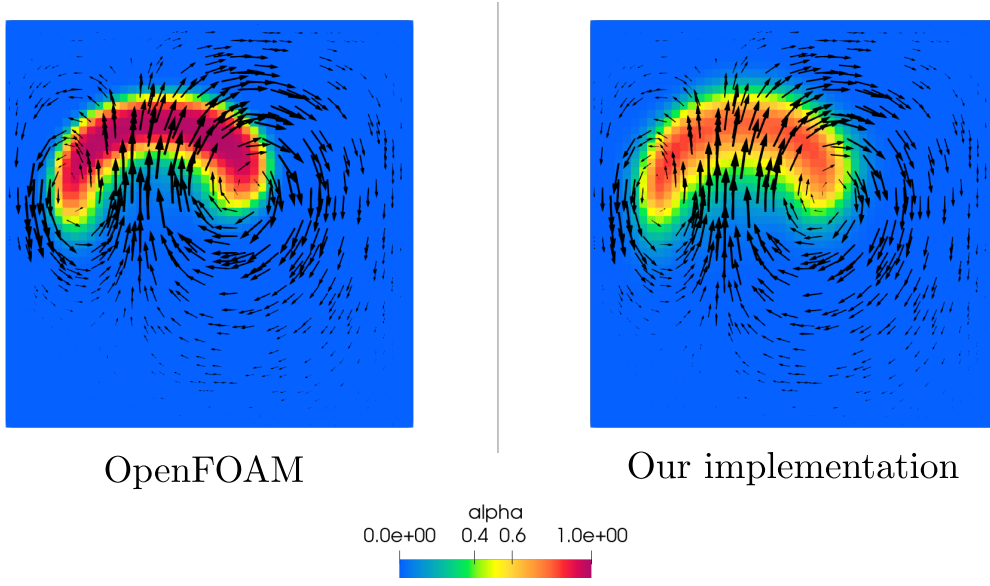
Figure 11. Results of the validation of our FVM for the Navier–Stokes equation with mixture. Arrows indicate the velocity fields.

E. Experiment Details: Convection–Diffusion Equation

E.1. Dataset

The equation of interest is

$$\frac{\partial}{\partial t} u = -\nabla \cdot (\boldsymbol{c}u - D\nabla u), \ \ (t, x) \in (0, T_{\max}) \times [0, 1), \tag{47}$$

with periodic boundary conditions, where $D = 10^{-4}$ and $\|\boldsymbol{c}\| \in [0.0, 0.2]$ is obtained from the uniform distribution. We define the initial condition as

$$u(t = 0, x) = u_{\mathrm{amp}} \cos(2\pi(x + x_0)), \ \ x \in [0.0, 1.0) \tag{48}$$

with $u_{\mathrm{amp}} \in [0.5, 1.0]$ and $x_0 \in [0.0, 1.0]$ obtained from the uniform distribution. A precise solution to the specified initial condition, when subjected to periodic boundary conditions is

$$u(t, x) = u_{\mathrm{amp}} \exp(-(2\pi)^2 Dt) \cos(2\pi(x - ct + x_0)). \tag{49}$$

We generated the dataset by randomly varying the initial condition and convecting velocity.

E.2. FluxGNN Modeling

The FluxGNN model used for the convection–diffusion equation is expressed as explained below.

ENCODER

All encoders are linear following Theorem 3.3. Using vertex-wide linear functions $\{\epsilon_{\cdot}\}$, each input feature is encoded as

$$\begin{aligned} u: & \quad \tilde{u}_i = \epsilon_u(u_i), & i \in \mathcal{V} \\ \text{Dirichlet for } u: & \quad \tilde{\hat{u}}_{ij} = \epsilon_u(\hat{u}_{ij}), & (i, j) \in \mathcal{E}_{\mathrm{Dirichlet}} \\ \text{Neumann for } u: & \quad \tilde{\hat{g}}_{ij} = \epsilon_u(\hat{g}_{ij}), & (i, j) \in \mathcal{E}_{\mathrm{Neumann}} \\ \boldsymbol{c}: & \quad \tilde{\boldsymbol{c}}_{ij} = \epsilon_{\boldsymbol{c}}(\boldsymbol{c}_{ij}), & (i, j) \in \mathcal{E} \\ D: & \quad \tilde{D}_{ij} = \epsilon_D(D_{ij}), & (i, j) \in \mathcal{E} \end{aligned} \tag{50}$$

We used encoders with embedded dimensions of 64.
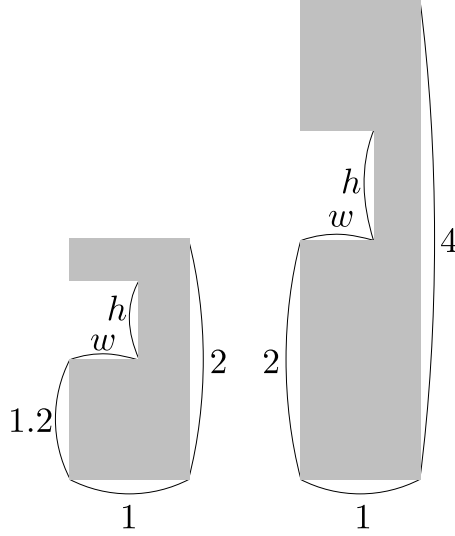
Figure 12. Template shapes for training, validation, and test dataset (left) and taller dataset (right). Both use two parameters, h and w.

PROCESSOR

The processing part is constructed using FVM formulation and FluxGNN as

$$\tilde{u}(t + \Delta t) = \tilde{u}(t) + \sum_{j \in \mathcal{N}_i} S_{ij} \boldsymbol{n}_{ij} \cdot \left[\boldsymbol{F}_{\text{conv}}(\tilde{u}_i, \tilde{u}_j, \tilde{u}_{ij}, \tilde{\boldsymbol{c}}_{ij}) - \tilde{D}_{ij} \boldsymbol{F}_{\text{grad}}(\tilde{u}_i, \tilde{u}_j, \tilde{u}_{ij}) \right] \Delta t, \tag{51}$$

where we apply the explicit Euler scheme for time evolution. For MLPs used in similarity-equivariant MLPs, we used the structure

$$\text{MLP}(\boldsymbol{h}) := \tanh \circ \text{Linear}_{64 \leftarrow 64}^{(2)} \circ \tanh \circ \text{Linear}_{64 \leftarrow 64}^{(1)}(\boldsymbol{h}), \tag{52}$$

where $\text{Linear}_{64 \leftarrow 64}^{(\cdot)}$ denotes a linear layer mapping from 64 features to 64 features.

DECODER

We obtain a time series of encoded u using the autoregressive computation in the processing step. Applying the decoder δ_u that satisfies $\delta_u \circ \epsilon_u = \text{id}$, we obtain

$$u_i^{\text{pred}}(t) := \delta_u(\tilde{u}_i(t)), \quad \forall i \in \mathcal{V}, \forall t \in \{\Delta t, 2\Delta t, \dots, T_{\max}\}. \tag{53}$$

F. Experiment Details: Navier–Stokes Equations with Mixture

F.1. Dataset

We generated shapes with varying dimensions, as shown in Figure 12. Specifically, we varied h and w across these settings: 0.2, 0.4, and 0.6, resulting in a total of nine distinct shapes. To avoid data leakage, we partitioned these shapes into three sets: five for training, two for validation, and two for testing. Subsequently, we created four initial conditions for each shape using random numerical analysis. These initial conditions were employed in OpenFOAM simulations, utilizing fine meshes with a spatial resolution of $\Delta x = 1/320$. Additionally, we configured the simulation with the following parameters: kinematic viscosity $\nu = 10^{-3}$, diffusion coefficient $D_\alpha = 10^{-6}$, density of solvent $\rho_{\text{high}} = 1000$, density of solute $\rho_{\text{low}} = 990$, and gravity acting in the y direction with a magnitude of 9.81. We employed the isolated boundary condition with walls, specifically:

$$\begin{aligned}
\boldsymbol{u}(\boldsymbol{x}) &= \boldsymbol{0}, && \text{on} && \partial\Omega \\
p(\boldsymbol{x}) &= 0, && \text{on} && \partial\Omega_{\min y} \\
\boldsymbol{n}(\boldsymbol{x}) \cdot \nabla p(\boldsymbol{x}) &= 0, && \text{on} && \partial\Omega \setminus \partial\Omega_{\min y} \\
\boldsymbol{n}(\boldsymbol{x}) \cdot \nabla \alpha(\boldsymbol{x}) &= 0, && \text{on} && \partial\Omega
\end{aligned} \tag{54}$$

Table 5. Hyperparameter range used for FluxGNN.

Name	Range
n_{bundle}: # of steps bundled	2, 4
n_{feature}: # encoded features	8, 16, 32, 64
n_{rep}: # neural nonlinear steps	2, 4, 8

where $\partial\Omega_{\min y}$ represents the boundary at the bottom.

After OpenFOAM simulations, we interpolated the results to coarsened meshes with $\Delta x = 1/20$. This was done deliberately to create a more demanding machine learning problem because coarsened meshes are known to introduce significant numerical diffusion, resulting in difficulty in maintaining the density field sharp. Our primary objective with the machine learning task is intended to examine the ability of each method to effectively maintain the sharpness of the density field (equivalently, α), even when faced with pronounced numerical diffusion.

We divided the time series on coarse meshes into 100 trajectories with shifting time windows whose width and step size are 1.6 to augment the training dataset. Finally, we obtain 200, 8, and 8 trajectories for the training, validation, and test datasets.

We generated a rotation dataset by applying random rotation and a scaling dataset by applying a random scaling of space, time, and mass scales to evaluate the generalizability of each method. Further, we generated the taller dataset, which has taller shapes than the test dataset, as illustrated in Figure 12. We chose dimension parameters as $(h, w) = (0.5, 0.2), (0.5, 0.4)$, and $(0.5, 0.6)$. We generated one trajectory for each shape in the taller dataset using the same procedure to generate the test dataset.

F.2. FluxGNN modeling

The fundamental structure of FluxGNN remain consistent with the approach applied for the convection–diffusion equation, with the addition of established machine learning techniques, including temporal bundling and the neural nonlinear solver. The hyperparameter employed for the study is detailed in Table 5.

To obtain the pressure field, we applied the fractional step method as done in (Horie & Mitsume, 2022). The equations to solve are turned into

$$\frac{\partial}{\partial t}\alpha = -\nabla \cdot [\boldsymbol{u}\alpha - D_\alpha \nabla\alpha] \tag{55}$$

$$\widetilde{\rho\boldsymbol{u}} := \rho\boldsymbol{u} - \nabla \cdot \left[\rho\boldsymbol{u} \otimes \boldsymbol{u}\frac{\nu}{\rho}\left[\nabla \otimes \boldsymbol{u} - (\nabla \otimes \boldsymbol{u})^\top\right]\right]\Delta t \tag{56}$$

$$\nabla \cdot \nabla(p_{\rho g y}^+) = -\nabla \cdot (\boldsymbol{g} \cdot \boldsymbol{h}\nabla\rho) + \frac{1}{\Delta t}\left[\nabla \cdot \widetilde{\rho\boldsymbol{u}} + \frac{\partial}{\partial t}\rho\right] \tag{57}$$

$$\boldsymbol{u}^+ = \frac{1}{\rho}\left[\widetilde{\rho\boldsymbol{u}} - \Delta t(\nabla p_{\rho g h}^+ + \boldsymbol{g} \cdot \boldsymbol{h}\nabla\rho)\right], \tag{58}$$

where $\cdot^+$ denotes variables at the next time step. We utilized a matrix-free conjugate gradient (CG) method (Prabhune & Suresh, 2020), which corresponds to an iterative application of locally conservative diffusion operations. In the CG method, we refrain from using trainable functions to solve the Poisson equation for pressure. This choice is made due to the potentially large number of iterations required (e.g., 20 to 100 iterations), as the presence of trainable parameters in a deep loop often leads to instability in the backward process.

F.3. Baselines

The visibility of a wide range of hops in the model is attributed to FluxGNN's utilization of numerous iterations within the forward loop. Consequently, our hyperparameter investigations encompassed baseline models with varying levels of visible hops, in addition to exploring different configurations of hidden features. The comprehensive details of these study parameters can be found in Tables 6 to 8.

Table 6. Hyperparameter range used for MP-PDE.

Name	Range
n_{bundle}: # of steps bundled	2, 4, 8
n_{feature}: # encoded features	32, 64, 128
n_{neighbor}: # neighbors considered in one forward computation of a GNN layer	4, 8, 16

Table 7. Hyperparameter range used for PENN.

Name	Range
n_{feature}: # encoded features	4, 8, 16, 32, 64
n_{rep}: # neural nonlinear steps	4, 8, 16

F.4. Results

Figures 13 and 14 present additional visualizations of the results. Notably, FluxGNN exhibits robust performance, even when confronted with unseen shapes from the taller dataset. Interestingly, PENN also demonstrates proficient performance in predicting α for these unfamiliar shapes, likely owing to the $\text{E}(n)$-equivariance embedded within the model.

Figure 15 provides a concise overview of the tradeoff between speed and accuracy, as ascertained through hyperparameter studies. All data used for the plot are shown in Tables 9 to 12. In the tested domain, our model consistently achieves higher accuracy compared to MP-PDE, primarily because it incorporates the underlying laws of physics. Meanwhile, MP-PDE showcases commendable computational efficiency due to its less specific physics implementation, resulting in a simpler overall model structure. PENN, on the other hand, delivers superior accuracy compared to MP-PDE models but incurs a relatively longer runtime, as it lacks dedicated mechanisms for time-series computation. Our FluxGNN models have high generalizability, as demonstrated in Section 4. Furthermore, they tend to achieve higher computational efficiency than PENN, owing to the incorporation of temporal bundling, a method designed to streamline time-series computation.

Table 8. Hyperparameter range used for FVM.

Name	Range
$n_{\text{rep}\boldsymbol{u}}$: # loops for solver of $\boldsymbol{u}$	4, 8, 16, 32, 64, 128
$n_{\text{rep}\alpha}$: # loops for solver of α	4, 8, 16, 32, 64, 128
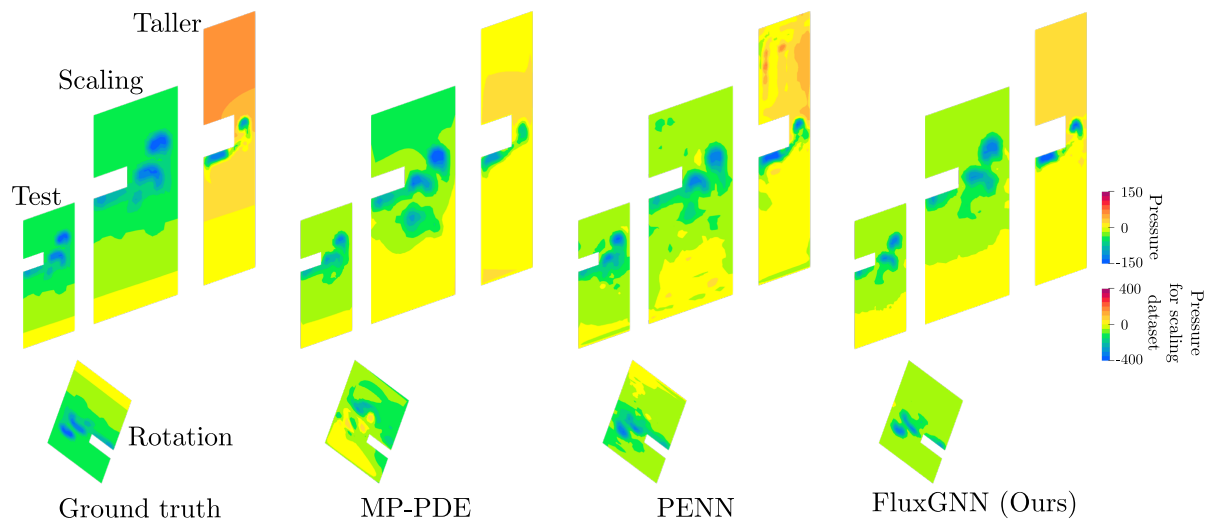
Figure 13. Visual comparison of pressure field between ground truth, MP-PDE, PENN, and FluxGNN.
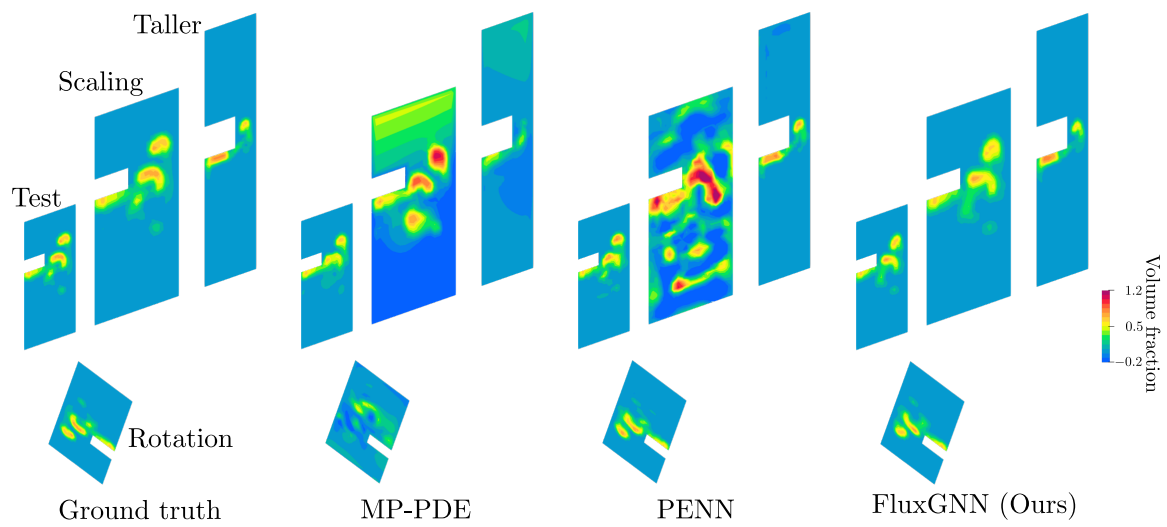
Figure 14. Visual comparison of volume fraction field between ground truth, MP-PDE, PENN, and FluxGNN.
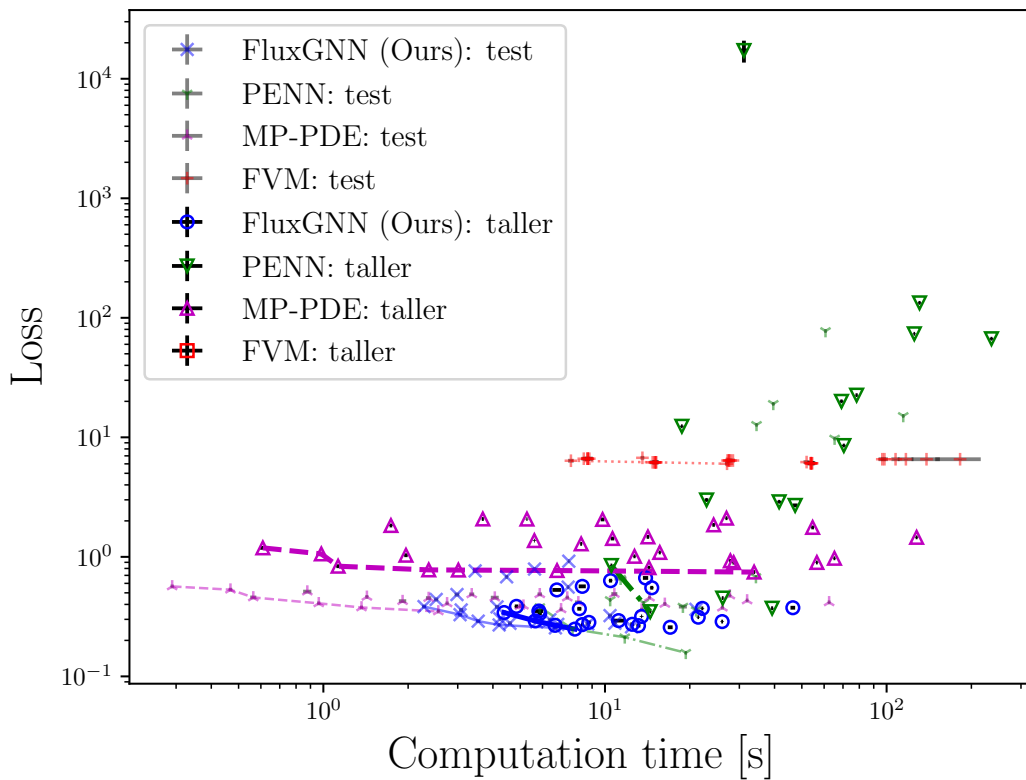
Figure 15. Speed–accuracy tradeoff of machine learning models (MP-PDE, PENN, and FluxGNN) and FVM, with error bars corresponding to the standard error of the mean. Light and dark colors correspond to the evaluation of the test and taller datasets, respectively. Lines represent Pareto fronts. FVM did not converge on the taller dataset with all settings tested, and therefore, there is no plot for FVM on the dataset. All computations are done on the same CPU (Intel Xeon CPU E5-2695 v2 @ 2.40 GHz) with one core.

Table 9. Detailed results of the hyperparameter study for FVM. All computations are done on the same CPU (Intel Xeon CPU E5-2695 v2 @ 2.40 GHz) with one core.

$n_{\mathrm{rep}u}$	$n_{\mathrm{rep}\alpha}$	Computation time on the test dataset [s]	Loss on the test dataset $(\times 10^{-1})$	Computation time on the taller dataset [s]	Loss on the taller dataset $(\times 10^{-1})$
4	4	5.48 ± 1.33	NaN $\pm$ NaN	39.83 ± 7.38	NaN $\pm$ NaN
8	4	5.81 ± 0.97	NaN $\pm$ NaN	43.83 ± 6.65	NaN $\pm$ NaN
16	4	6.61 ± 0.97	NaN $\pm$ NaN	42.23 ± 7.37	NaN $\pm$ NaN
32	4	7.71 ± 1.10	NaN $\pm$ NaN	46.14 ± 7.50	NaN $\pm$ NaN
64	4	8.99 ± 1.32	NaN $\pm$ NaN	50.53 ± 7.37	NaN $\pm$ NaN
128	4	11.06 ± 1.78	NaN $\pm$ NaN	60.31 ± 7.36	NaN $\pm$ NaN
4	8	7.57 ± 0.17	63.72 ± 0.33	108.40 ± 13.99	NaN $\pm$ NaN
8	8	8.41 ± 0.16	66.26 ± 0.34	98.27 ± 15.83	NaN $\pm$ NaN
16	8	8.69 ± 0.15	66.26 ± 0.34	100.47 ± 9.41	NaN $\pm$ NaN
32	8	8.66 ± 0.18	66.26 ± 0.34	106.87 ± 9.77	NaN $\pm$ NaN
64	8	8.80 ± 0.27	66.26 ± 0.34	110.52 ± 15.14	NaN $\pm$ NaN
128	8	8.69 ± 0.34	66.26 ± 0.34	125.54 ± 15.92	NaN $\pm$ NaN
4	16	13.58 ± 0.21	67.40 ± 0.27	179.02 ± 42.98	NaN $\pm$ NaN
8	16	15.02 ± 0.34	61.77 ± 0.26	186.34 ± 44.78	NaN $\pm$ NaN
16	16	14.77 ± 0.35	61.77 ± 0.26	188.38 ± 44.49	NaN $\pm$ NaN
32	16	15.20 ± 0.36	61.77 ± 0.26	196.69 ± 45.23	NaN $\pm$ NaN
64	16	15.16 ± 0.36	61.77 ± 0.26	213.58 ± 47.77	NaN $\pm$ NaN
128	16	15.07 ± 0.35	61.77 ± 0.26	242.97 ± 51.98	NaN $\pm$ NaN
4	32	27.13 ± 0.47	60.10 ± 0.25	338.46 ± 128.31	NaN $\pm$ NaN
8	32	27.63 ± 0.46	63.99 ± 0.26	342.37 ± 127.92	NaN $\pm$ NaN
16	32	27.51 ± 0.44	63.99 ± 0.26	350.57 ± 130.42	NaN $\pm$ NaN
32	32	27.24 ± 0.48	63.99 ± 0.26	365.23 ± 135.17	NaN $\pm$ NaN
64	32	28.47 ± 0.47	63.99 ± 0.26	392.99 ± 145.17	NaN $\pm$ NaN
128	32	27.94 ± 0.46	63.99 ± 0.26	441.77 ± 164.01	NaN $\pm$ NaN
4	64	51.99 ± 1.29	62.07 ± 0.25	810.78 ± 202.26	NaN $\pm$ NaN
8	64	54.14 ± 1.49	60.69 ± 0.25	771.13 ± 183.60	NaN $\pm$ NaN
16	64	53.82 ± 1.46	60.69 ± 0.25	834.60 ± 205.83	NaN $\pm$ NaN
32	64	53.93 ± 1.51	60.69 ± 0.25	862.31 ± 210.42	NaN $\pm$ NaN
64	64	54.43 ± 1.50	60.69 ± 0.25	923.02 ± 222.67	NaN $\pm$ NaN
128	64	53.76 ± 1.54	60.69 ± 0.25	1088.87 ± 267.00	NaN $\pm$ NaN
4	128	96.50 ± 2.89	65.58 ± 0.26	1830.75 ± 371.98	NaN $\pm$ NaN
8	128	98.27 ± 2.73	65.58 ± 0.26	1721.97 ± 444.38	NaN $\pm$ NaN
16	128	107.59 ± 3.66	65.58 ± 0.26	1749.10 ± 446.70	NaN $\pm$ NaN
32	128	117.08 ± 7.49	65.58 ± 0.26	1825.21 ± 468.47	NaN $\pm$ NaN
64	128	138.57 ± 15.99	65.58 ± 0.26	1915.15 ± 485.07	NaN $\pm$ NaN
128	128	182.36 ± 33.54	65.58 ± 0.26	2147.13 ± 538.70	NaN $\pm$ NaN

Table 10. Detailed results of the hyperparameter study for MP-PDE. All computations are done on the same CPU (Intel Xeon CPU E5-2695 v2 @ 2.40 GHz) with one core.

n_{bundle}	$n_{feature}$	$n_{neighbor}$	Computation time on the test dataset [s]	Loss on the test dataset $(\times 10^{-1})$	Computation time on the taller dataset [s]	Loss on the taller dataset $(\times 10^{-1})$
2	32	4	0.88 ± 0.03	5.12 ± 0.03	1.74 ± 0.02	18.29 ± 0.14
2	32	8	4.06 ± 0.13	4.56 ± 0.02	9.81 ± 0.15	20.56 ± 0.09
2	32	16	10.80 ± 0.27	4.89 ± 0.02	27.00 ± 0.23	21.06 ± 0.11
2	64	4	1.91 ± 0.06	4.26 ± 0.02	3.68 ± 0.03	20.74 ± 0.09
2	64	8	8.04 ± 0.22	4.16 ± 0.02	24.31 ± 0.25	18.51 ± 0.09
2	64	16	27.75 ± 0.31	4.75 ± 0.02	56.54 ± 0.42	8.97 ± 0.05
2	128	4	4.99 ± 0.06	3.88 ± 0.02	10.64 ± 0.15	14.23 ± 0.07
2	128	8	26.11 ± 0.32	3.76 ± 0.02	54.61 ± 0.60	17.62 ± 0.10
2	128	16	62.50 ± 0.69	4.16 ± 0.02	127.80 ± 1.03	14.56 ± 0.08
4	32	4	0.47 ± 0.01	5.31 ± 0.03	0.98 ± 0.01	10.58 ± 0.06
4	32	8	2.37 ± 0.05	4.52 ± 0.02	5.28 ± 0.02	20.70 ± 0.10
4	32	16	5.87 ± 0.11	4.86 ± 0.02	14.21 ± 0.15	14.68 ± 0.08
4	64	4	0.96 ± 0.01	4.08 ± 0.02	1.96 ± 0.03	10.32 ± 0.05
4	64	8	5.13 ± 0.07	3.90 ± 0.02	12.73 ± 0.11	10.10 ± 0.05
4	64	16	14.39 ± 0.16	4.56 ± 0.02	28.64 ± 0.29	8.95 ± 0.05
4	128	4	2.56 ± 0.03	3.49 ± 0.02	5.61 ± 0.02	13.67 ± 0.09
4	128	8	13.50 ± 0.17	3.61 ± 0.02	27.88 ± 0.27	9.26 ± 0.06
4	128	16	31.97 ± 0.35	4.31 ± 0.02	65.27 ± 0.51	9.73 ± 0.05
8	32	4	0.29 ± 0.00	5.66 ± 0.03	0.61 ± 0.01	11.89 ± 0.06
8	32	8	1.43 ± 0.02	4.63 ± 0.02	2.37 ± 0.07	7.78 ± 0.04
8	32	16	3.36 ± 0.07	4.86 ± 0.02	8.23 ± 0.06	12.84 ± 0.06
8	64	4	0.56 ± 0.01	4.57 ± 0.02	1.13 ± 0.02	8.35 ± 0.05
8	64	8	2.75 ± 0.04	4.08 ± 0.02	6.76 ± 0.06	7.68 ± 0.05
8	64	16	7.35 ± 0.10	4.56 ± 0.02	15.65 ± 0.13	10.88 ± 0.05
8	128	4	1.37 ± 0.02	3.77 ± 0.02	3.02 ± 0.03	7.76 ± 0.04
8	128	8	6.98 ± 0.09	3.64 ± 0.02	14.33 ± 0.16	8.11 ± 0.04
8	128	16	16.31 ± 0.19	4.04 ± 0.02	33.84 ± 0.28	7.46 ± 0.04

Table 11. Detailed results of the hyperparameter study for PENN. All computations are done on the same CPU (Intel Xeon CPU E5-2695 v2 @ 2.40 GHz) with one core.

$n_{feature}$	n_{rep}	Computation time on the test dataset [s]	Loss on the test dataset $(\times 10^{-1})$	Computation time on the taller dataset [s]	Loss on the taller dataset $(\times 10^{-1})$
4	4	5.87 ± 0.04	3.80 ± 0.02	10.54 ± 0.07	8.47 ± 0.07
4	8	10.43 ± 0.08	4.36 ± 0.02	18.75 ± 0.13	123.80 ± 1.43
4	16	11.37 ± 0.55	6.57 ± 0.03	31.15 ± 0.79	172376.18 ± 35812.77
8	4	7.67 ± 0.06	2.49 ± 0.01	14.47 ± 0.11	3.48 ± 0.02
8	8	13.72 ± 0.12	4.13 ± 0.02	26.18 ± 0.19	4.52 ± 0.02
8	16	18.91 ± 0.62	3.83 ± 0.02	47.34 ± 0.98	26.96 ± 0.34
16	4	11.76 ± 0.10	2.12 ± 0.01	22.95 ± 0.18	29.93 ± 0.87
16	8	21.07 ± 0.20	3.57 ± 0.02	41.53 ± 0.32	28.90 ± 0.63
16	16	39.59 ± 0.39	190.58 ± 2.09	78.26 ± 0.69	225.39 ± 1.59
32	4	19.36 ± 0.19	1.58 ± 0.01	39.21 ± 0.32	3.72 ± 0.02
32	8	34.52 ± 0.34	127.20 ± 0.80	69.15 ± 0.62	200.11 ± 3.77
32	16	65.37 ± 0.62	97.67 ± 0.42	130.88 ± 1.13	1333.77 ± 16.01
64	4	34.36 ± 0.32	6.78 ± 0.03	70.49 ± 0.60	85.39 ± 1.28
64	8	60.68 ± 0.63	775.32 ± 4.13	125.40 ± 0.80	734.80 ± 5.41
64	16	114.56 ± 1.10	150.98 ± 0.75	235.70 ± 1.71	668.38 ± 12.94

Table 12. Detailed results of the hyperparameter study for FluxGNN. All computations are done on the same CPU (Intel Xeon CPU E5-2695 v2 @ 2.40 GHz) with one core.

| n_{bundle}	n_{feature}	n_{rep}	Computation time on the test dataset [s]	Loss on the test dataset $(\times 10^{-1})$	Computation time on the taller dataset [s]	Loss on the taller dataset $(\times 10^{-1})$
2	8	2	2.28 ± 0.04	3.84 ± 0.02	4.38 ± 0.07	3.42 ± 0.02
2	8	4	3.06 ± 0.05	3.28 ± 0.02	5.66 ± 0.13	2.91 ± 0.02
2	8	8	4.47 ± 0.05	6.81 ± 0.04	8.31 ± 0.24	5.66 ± 0.04
2	16	2	2.52 ± 0.04	4.40 ± 0.03	4.85 ± 0.06	3.85 ± 0.03
2	16	4	3.55 ± 0.06	2.90 ± 0.01	6.65 ± 0.14	2.67 ± 0.02
2	16	8	5.62 ± 0.07	7.91 ± 0.04	10.47 ± 0.14	6.32 ± 0.04
2	32	2	3.09 ± 0.05	3.59 ± 0.02	5.87 ± 0.10	3.36 ± 0.02
2	32	4	4.60 ± 0.07	2.78 ± 0.01	8.77 ± 0.13	2.83 ± 0.02
2	32	8	7.40 ± 0.06	5.58 ± 0.03	14.66 ± 0.18	5.51 ± 0.05
2	64	2	4.14 ± 0.07	3.81 ± 0.03	8.11 ± 0.13	3.68 ± 0.03
2	64	4	6.67 ± 0.11	2.55 ± 0.01	13.15 ± 0.21	2.66 ± 0.02
2	64	8	10.42 ± 0.17	3.22 ± 0.02	22.16 ± 0.18	3.71 ± 0.02
4	8	2	2.98 ± 0.06	4.84 ± 0.02	5.82 ± 0.22	3.52 ± 0.02
4	8	4	4.21 ± 0.05	2.69 ± 0.01	7.82 ± 0.13	2.49 ± 0.01
4	8	8	7.44 ± 0.06	9.20 ± 0.04	13.93 ± 0.30	6.66 ± 0.04
4	16	2	3.46 ± 0.07	7.63 ± 0.04	6.75 ± 0.24	5.29 ± 0.05
4	16	4	5.72 ± 0.12	2.86 ± 0.01	11.18 ± 0.62	2.93 ± 0.02
4	16	8	8.69 ± 0.16	2.74 ± 0.01	17.05 ± 0.34	2.58 ± 0.01
4	32	2	4.24 ± 0.05	3.03 ± 0.01	8.31 ± 0.09	2.71 ± 0.02
4	32	4	6.80 ± 0.09	2.72 ± 0.02	13.49 ± 0.05	3.18 ± 0.03
4	32	8	12.43 ± 0.14	2.60 ± 0.01	26.06 ± 0.19	2.87 ± 0.02
4	64	2	6.37 ± 0.08	2.97 ± 0.01	12.49 ± 0.19	2.73 ± 0.02
4	64	4	10.84 ± 0.15	2.78 ± 0.02	21.44 ± 0.14	3.13 ± 0.02
4	64	8	21.03 ± 0.23	3.64 ± 0.02	46.46 ± 0.88	3.75 ± 0.02
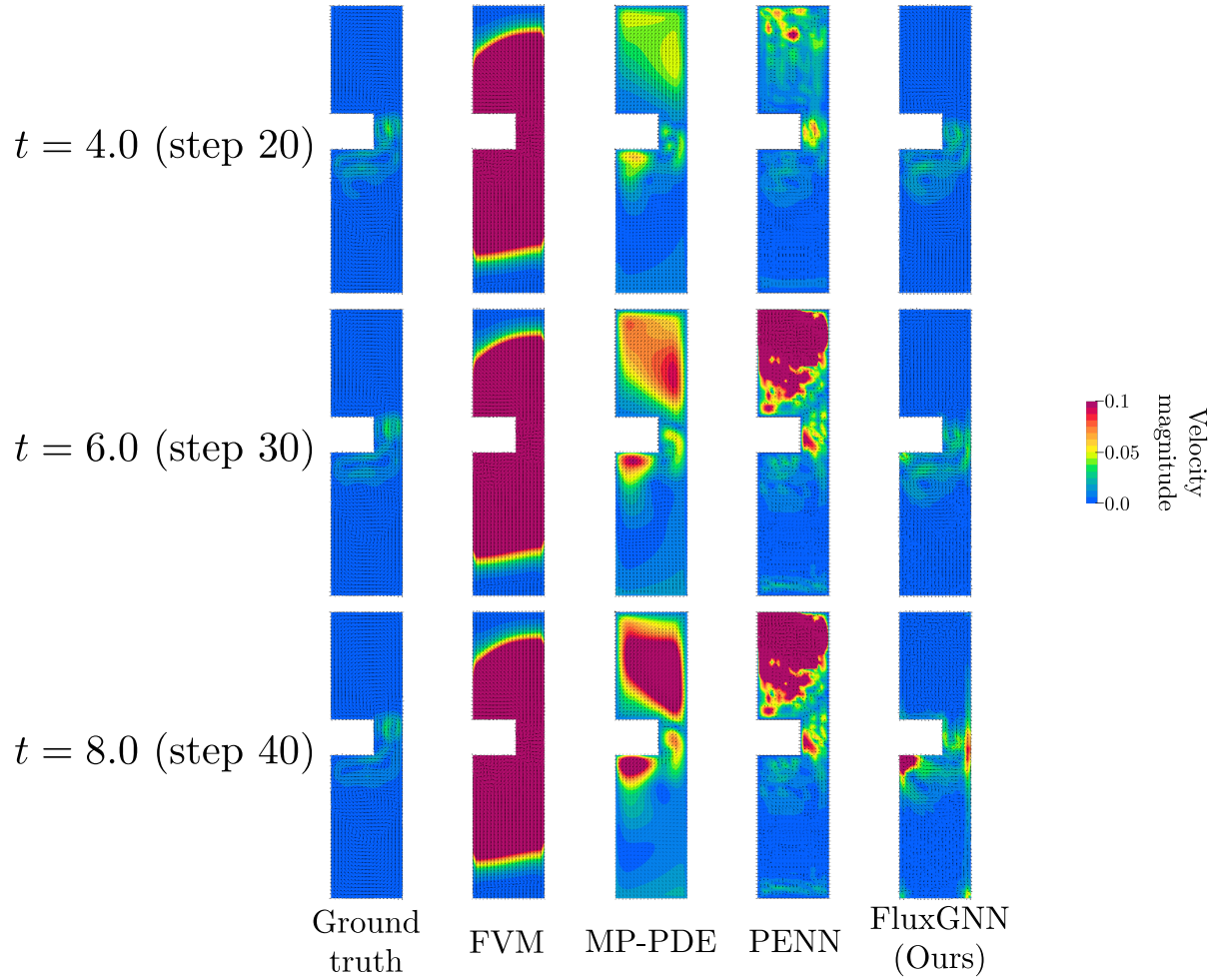
Figure 16. Time evolution of velocity field for ground truth, FVM, MP-PDE, PENN, and FluxGNN.

F.5. Longer Temporal Rollout

Figures 16 to 18 visualize velocity, pressure, and volume fraction fields of longer temporal rollout predictions. Our observations are as follows:

- A substantial part of the error of FluxGNN is mainly due to the pressure. It seems that pressure starts to be unstable earlier than velocity and volume fraction, implying that instability of pressure would be key to establishing a more stable model. MP-PDE has a lower loss than FluxGNN at $t = 8.0$, but these outputs are not qualitatively similar to those of ground truth.

- FluxGNN keeps the distribution of the volume fraction field (α) similar to ground truth, while other methods show entirely dissimilar fields. That may be due to the conservative property of our method because the volume fraction is strictly conservative in the present setting.
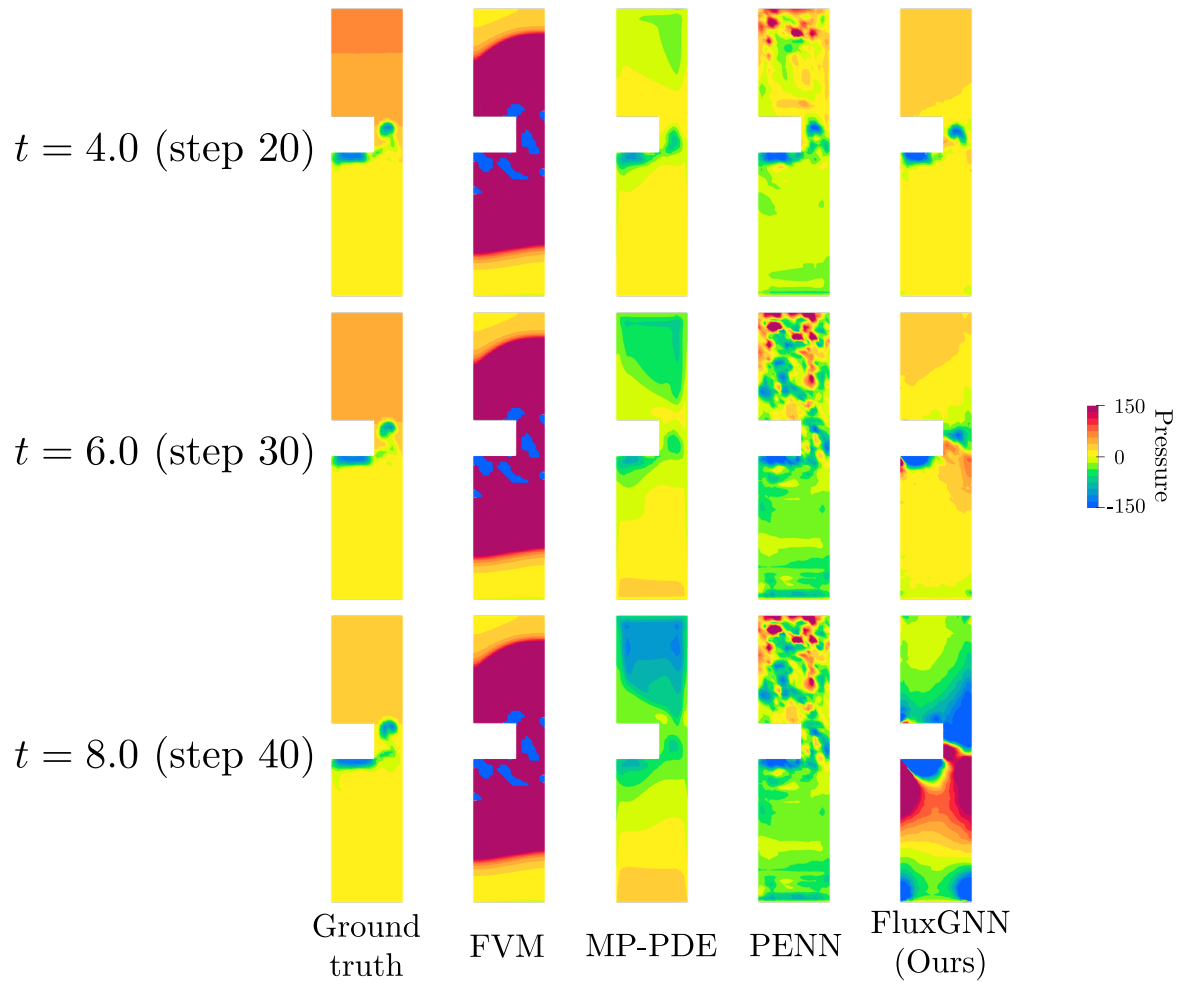
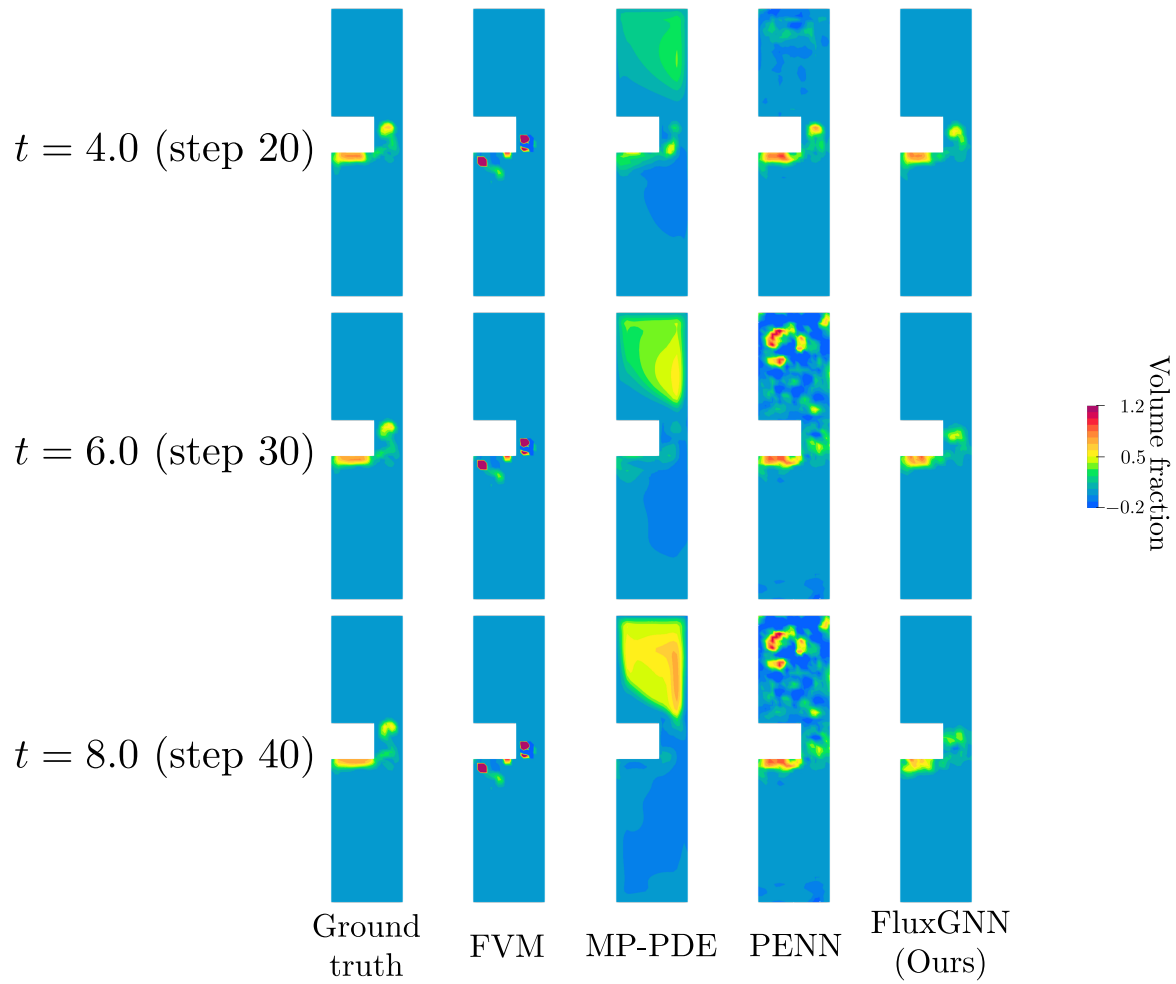*Figure 17.* Time evolution of pressure field for ground truth, FVM, MP-PDE, PENN, and FluxGNN.

Figure 18. Time evolution of volume fraction field for ground truth, FVM, MP-PDE, PENN, and FluxGNN.

F.6. Ablation Study

Table 13 presents the results of an ablation study. That suggests that all components included in the model contribute to spatial out-of-domain generalizability. We compared the following models:

- Without temporal bundling

- Without neural nonlinear solver, meaning explicit temporal differentiation

- Without conservation, meaning no permutation-invariance in the flux functions

- Without scaling-equivariance

- Without $\mathrm{E}(n)$-equivariance

- FluxGNN, the proposed method

The model without $\mathrm{E}(n)$-equivariance performs best on the test, rotation, and scaling datasets. In particular, it performs well on the rotation dataset, although the model has no $\mathrm{E}(n)$-equivariance. That implies that the inductive bias, introduced with an FVM-like computation procedure, works quite fine.

FluxGNN performs the best on the taller dataset. The model without $\mathrm{E}(n)$-equivariance degrades on the taller dataset, implying insufficient generalizability. Our primal goal is to construct reliable methods toward out-of-domain generalizability. Therefore, all the suggested components are necessary to achieve our goal.

Table 13. RMSE loss and conservation error ($\pm$ standard error of the mean) on the evaluation datasets of the Navier–Stokes equation with mixture for an ablation study. Each metric is normalized using standard deviation.

Method	Dataset	Loss $\boldsymbol{u}$ $\left(\times 10^{-1}\right)$	Loss p $\left(\times 10^{-1}\right)$	Loss α $\left(\times 10^{-1}\right)$	Conservation error α $\left(\times 10^{-5}\right)$
w/o temporal bundling	test	2.075 ± 0.015	5.209 ± 0.034	2.851 ± 0.030	$\mathbf{0.01} \pm 0.00$
w/o neural nonlinear solver	test	1.291 ± 0.009	1.791 ± 0.011	0.429 ± 0.005	0.91 ± 0.33
w/o conservation	test	1.252 ± 0.009	1.381 ± 0.014	0.405 ± 0.006	86.89 ± 3.88
w/o scaling-equivariance	test	1.835 ± 0.013	1.410 ± 0.010	0.492 ± 0.007	0.05 ± 0.01
w/o $\mathrm{E}(n)$-equivariance	test	$\mathbf{0.839} \pm 0.005$	1.192 ± 0.007	$\mathbf{0.282} \pm 0.004$	$\mathbf{0.01} \pm 0.00$
FluxGNN	test	1.202 ± 0.008	$\mathbf{1.143} \pm 0.008$	0.349 ± 0.005	0.06 ± 0.03
w/o temporal bundling	rotation	2.114 ± 0.012	3.703 ± 0.028	2.850 ± 0.030	$\mathbf{0.01} \pm 0.00$
w/o neural nonlinear solver	rotation	1.290 ± 0.007	1.859 ± 0.010	0.426 ± 0.005	$\mathbf{0.01} \pm 0.00$
w/o conservation	rotation	1.248 ± 0.007	1.966 ± 0.014	0.414 ± 0.006	86.89 ± 3.88
w/o scaling-equivariance	rotation	1.834 ± 0.010	1.620 ± 0.010	0.489 ± 0.007	$\mathbf{0.01} \pm 0.00$
w/o $\mathrm{E}(n)$-equivariance	rotation	$\mathbf{0.852} \pm 0.005$	1.306 ± 0.007	$\mathbf{0.296} \pm 0.004$	$\mathbf{0.01} \pm 0.00$
FluxGNN	rotation	1.207 ± 0.007	$\mathbf{1.175} \pm 0.008$	0.351 ± 0.005	$\mathbf{0.01} \pm 0.00$
w/o temporal bundling	scaling	2.070 ± 0.015	5.812 ± 0.037	2.850 ± 0.030	$\mathbf{0.01} \pm 0.00$
w/o neural nonlinear solver	scaling	1.316 ± 0.009	1.790 ± 0.011	0.438 ± 0.005	1.80 ± 0.87
w/o conservation	scaling	1.245 ± 0.009	1.415 ± 0.013	0.410 ± 0.006	86.90 ± 3.89
w/o scaling-equivariance	scaling	1.922 ± 0.014	8.102 ± 0.782	27.797 ± 3.080	0.05 ± 0.02
w/o $\mathrm{E}(n)$-equivariance	scaling	$\mathbf{0.839} \pm 0.005$	$\mathbf{1.189} \pm 0.007$	$\mathbf{0.280} \pm 0.003$	$\mathbf{0.01} \pm 0.00$
FluxGNN	scaling	1.219 ± 0.009	1.228 ± 0.008	0.356 ± 0.005	0.05 ± 0.01
w/o temporal bundling	taller	1.822 ± 0.017	4.329 ± 0.029	2.212 ± 0.034	$\mathbf{0.01} \pm 0.00$
w/o neural nonlinear solver	taller	1.268 ± 0.010	1.202 ± 0.010	0.403 ± 0.007	0.34 ± 0.05
w/o conservation	taller	1.269 ± 0.012	1.336 ± 0.012	0.473 ± 0.009	205.37 ± 90.01
w/o scaling-equivariance	taller	1.604 ± 0.015	1.102 ± 0.009	0.398 ± 0.007	0.03 ± 0.00
w/o $\mathrm{E}(n)$-equivariance	taller	$\mathbf{1.122} \pm 0.009$	1.129 ± 0.009	0.440 ± 0.008	$\mathbf{0.01} \pm 0.00$
FluxGNN	taller	1.184 ± 0.009	$\mathbf{0.966} \pm 0.008$	$\mathbf{0.337} \pm 0.006$	0.02 ± 0.00