Computers & Graphics 103 (2022) 201–211

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

# Graph neural network-accelerated Lagrangian fluid simulation☆

Zijie Li [a], Amir Barati Farimani [b,*]

[a] Mechanical Engineering, Carnegie Mellon University, United States of America
[b] Mechanical Engineering and Machine Learning, Carnegie Mellon University, United States of America

ARTICLE INFO

*Article history:*
Received 12 October 2021
Received in revised form 28 January 2022
Accepted 4 February 2022
Available online 14 February 2022

*Keywords:*
Particle-based fluid dynamics
Graph neural networks
Data-driven modeling

ABSTRACT

We present a data-driven model for fluid simulation under Lagrangian representation. Our model, Fluid Graph Networks (FGN), uses graphs to represent the fluid field. In FGN, fluid particles are represented as nodes and their interactions are represented as edges. Instead of directly predicting the acceleration or position correction given the current state, FGN decomposes the simulation scheme into separate parts — advection, collision, and pressure projection. For these different predictions tasks, we propose two kinds of graph neural network structures, node-focused networks and edge-focused networks. We show that the learned model can produce accurate results and remain stable in scenarios with different geometries. In addition, FGN is able to retain many important physical properties of incompressible fluids, such as low velocity divergence, and adapt to time step sizes beyond the one used in the training set. FGN is also computationally efficient compared to classical simulation methods as it operates on a smaller neighborhood and does not require iteration at each timestep during the inference.



## 1. Introduction

For many science and engineering problems, fluid is an essential part. Over the years, a wide array of numerical models for simulating fluid dynamics have been developed. Despite the tremendous advances in both simulation methods and computing power, high-quality fluid simulation is still computationally expensive. The time of simulation increases drastically when the resolution of the simulation scene scales up.

An attractive alternative of direct simulation based on numerical methods is data-driven model. Recent progress in the machine learning domain opens up the possibility of employing machine learning techniques to build data-driven model to simulate fluid dynamics. By surrogating the most computational intensive part in the simulation with machine learning algorithms, data-driven model can achieve real-time level simulation speed [1–4]. Data-driven model can also achieve substantial acceleration by exploiting a reduced-order representation of the dynamical system [5–8]. These methods provide effective data-driven recipes for systems that can be discretized and processed on regular grids, but there are also many systems that cannot be represented as structured grid-based data. To cope with this irregularity, a large class of recent works adopt graph neural networks (GNN) [9,10] to represent the system and reason about the underlying interaction, including fluids and various other materials [11–20]. Despite impressive progress has been made on predicting physics with machine learning algorithms, satisfying accuracy cannot be achieved without an over-parameterized model and a large amount of training data. This is primarily due to the high complexity of underlying physics phenomena.

In this work, we propose a fluid simulator based on graph neural networks (Fluid Graph Networks, FGN),[1] which leverages message passing mechanism [21] to transform and pass information on the graph. Our method is based on the particle-based representation of fluids. In the model, the fluid particles are represented as nodes in a graph and their interactions are represented as edges. The design of FGN is inspired from classical numerical methods, where a single update step in the fluid dynamics is decomposed into two sub-steps — prediction based on body force and correction based on pressure projection. The decomposition enables us to approximate the fluid dynamics with shallower neural networks and less training data, since these sub-dynamical systems are much simpler than directly predicting the update dynamics of the full time step. In addition, the physics-based design of FGN makes it enjoy traceability of many physical properties without direct supervision, like low velocity-divergence and constant particle density. Experiments demonstrate that our model can remain stable and accurate in long-term simulation, for conditions that are within or beyond

☆ This article was recommended for publication by M. Teschner.

* Corresponding author.
*E-mail addresses:* zijieli@andrew.cmu.edu (Z. Li), barati@cmu.edu (A.B. Farimani).

[1] The code for this project is publicly available at: https://github.com/BaratiLab/FGN.

https://doi.org/10.1016/j.cag.2022.02.004

training data distribution. Furthermore, we show that FGN substantially improves the computational efficiency compared to classical simulation method and offers competitive performance among other data-driven simulators.

## 2. Related works

### 2.1. Particle-based fluid simulation

Our model is built upon the Lagrangian representation of fluids. The most prominent advantage of the Lagrangian representation is that the particle boundary is the material interface, which makes boundary conditions easy to impose, especially when the material interface is large and changing violently. A popular Lagrangian method is Smooth Particle Hydrodynamics (SPH) [22]. SPH and its variants are widely used in the numerical simulation, especially fluid dynamics under various environments. Particle-based fluid (PBF) [23] first introduces SPH model to simulate fluids and generate realistic visual effects for computer graphics application. Weakly compressible SPH (WSPH) [24] uses equation of state to model the pressure during the simulation. Moving particle semi-implicit method (MPS) [25] markedly improves the accuracy and stability of incompressible fluid simulation by introducing an implicit pressure projection procedure that emulates Eulerian grid-based methods. Later works Predictive–corrective incompressible SPH (PCISPH) [26], Implicit Incompressible SPH (IISPH) [27] and Divergence-free SPH (DFSPH) [28] leverage iterative scheme to improve the stability under larger time step sizes and enforce low volume compression in incompressible flow simulation.

### 2.2. Learning-based fluid simulation

Learning-based model, especially deep neural networks, has recently emerged as a promising approach for learning complex dynamics from data. Ladický et al. [1] proposes the first machine-learning-surrogate particle-based fluid model by reformulating the Navier–Stokes equation as a regression problem and then use random forest to predict the acceleration at each time step. Later work Zhang et al. [29] proposes an end-to-end particle-based fluid simulator which uses shared multi-layer perceptron to extract point features and predict velocity for each frame. Other than surrogation, machine learning can also be combined with classical solvers to improve the calculation efficiency and accuracy, such as using convolutional neural networks (CNNs) to learn and approximate the most computationally intensive step [2,3] or learning the reduced-order representation of the dynamical system [5–8]. In addition, neural networks can be used to enhance and reconstruct the coarse simulation results [13,30–34].

Graph neural networks (GNNs) [9,35], a neural network structure that operates on graph, have emerged as a powerful and flexible model for learning the interaction between entities in different systems such as $N$-body system [17,36], mesh-based system [13,14,16], and particle-based system [12,15,19,20,37]. Specifically for particle-based dynamical system, learning and reasoning under graph representation offers the following benefits and conveniences. First, in most particle-based systems, interactions between particles only happen within a local region. This imposes an inductive bias for the model: dynamics have strong locality. The locality of unstructured data under graph representation can be captured by a variety of message passing and aggregation operation on graphs, such as Graph Convolutional Neural Networks (GCNs) [38] and other variants like GraphSAGE [10], Graph Attention Networks [39], Message Passing Neural Networks [21]. Second, unlike grid-based methods, particle-based methods do not have explicit and structured grid. This makes standard CNNs cannot be directly applied to particles without feature processing like voxelization or projection, while graph neural networks can directly represent these unstructured entities as nodes and their interaction as edges.

## 3. Methodology

### 3.1. Fluid dynamics

The governing equations for incompressible fluids are the Navier–Stokes equation and the continuity equation as follows [40] (we use bold symbol to denote vector and non-bold symbol to denote scalar throughout the illustration):

$$\frac{D\mathbf{u}}{Dt} = -\frac{\nabla p}{\rho} + \nu\nabla^2\mathbf{u} + \mathbf{g}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{2}$$

In this work, we adopt a Lagrangian viewpoint to describe fluids. We follow the numerical model in Smooth Particle Hydrodynamics (SPH) [22] to discretize fluid field. In SPH, a scalar (or vector) field $A(\mathbf{x})$ at arbitrary location $\mathbf{x}$ can be represented by a convolution:

$$A(\mathbf{x}) = \int A(\mathbf{x}') W\left(\left|\mathbf{x} - \mathbf{x}'\right|, r_0\right) dV(\mathbf{x}'), \tag{3}$$

where $W$ is the weighting function or smooth kernel as defined in SPH, $r_0$ is the cutoff length, which defines the range of particles to be considered, and $V(\mathbf{x}')$ is the volume at $\mathbf{x}$. Numerically, the interpolation can be approximated by replacing the integration with a summation.

With discretization, the numerical solution of Eqs. (1) and (2) can be derived and then used to update the system. In particle-based fluid simulation, predictor–corrector scheme is usually adopted, which comprises two major steps — prediction based on advection and correction based on physics constraints (e.g. divergence-free constraint).

### 3.2. Model

Fluid is a time-dependent dynamical system, with locations of particles $\mathbf{x}$ described by the equation of form: $d\mathbf{x}/dt = f(\mathbf{x})$. When building a data-driven model to approximate the solution of this system, we assume it is Markovian, that is, the state of the system at time step $n+1$ depends only on the state of the previous time step $n$. Based on this assumption, the update mechanism in our model is defined as:

$$\{\mathbf{x}^{n+1}, \mathbf{v}^{n+1}\} = G_\theta\left(\{\mathbf{x}^n, \mathbf{v}^n\}\right). \tag{4}$$

Here $\{\mathbf{x}^n, \mathbf{v}^n\}$ denotes the Cartesian coordinates and velocities of fluid particles at time step $n$. The deep neural network $G_\theta$, parameterized by $\theta$, maps the state of time step $n$ to time step $n+1$.

In order to build an efficient model and enable the model to give interpretable output without compromising many physical properties of the system, we propose a model architecture that mimics the predictor–corrector scheme of many classical fluid solvers. The model comprises three sub-networks, advection net, collision net, and pressure net (see Fig. 1). They can be divided into two types of graph networks (GNs) according to the network structure [35]. Specifically, advection net and pressure net are node-focused graph networks, while collision net is edge-focused network. As each of these networks has a specific task and different output, they are trained on different data separately.

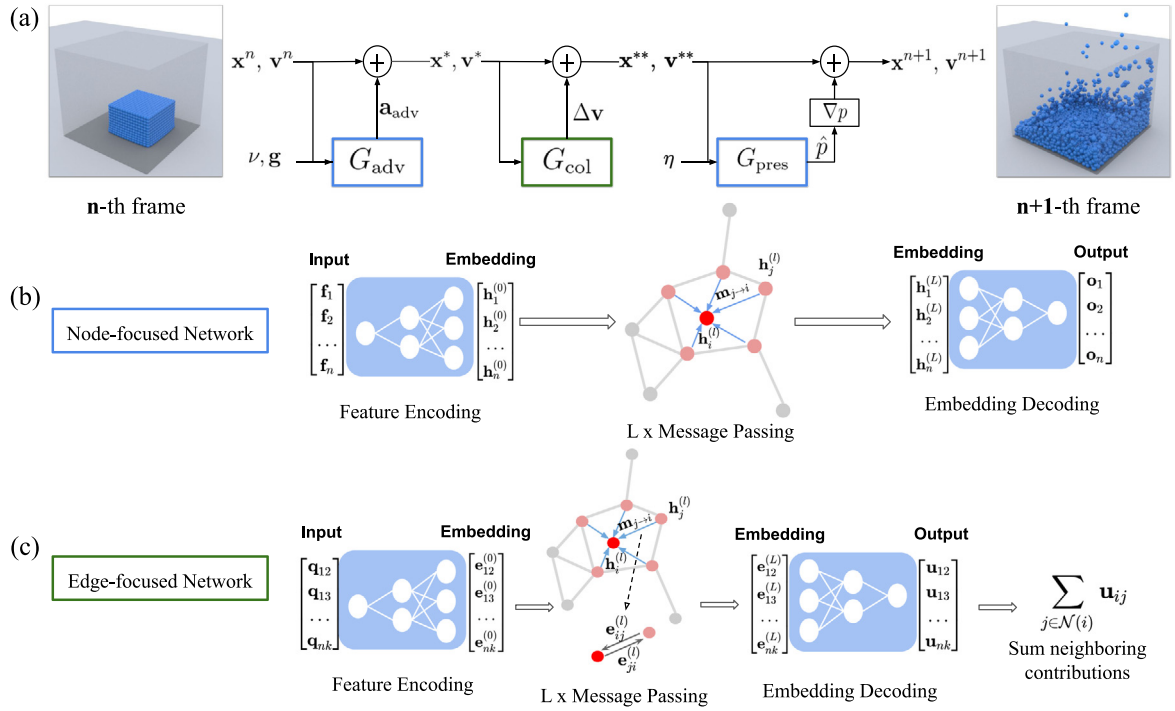



**Fig. 1.** **(a)** Schematic of Fluid Graph Networks (FGN). During each time step, $G_{\text{adv}}$ applies the effect of body force and viscosity to the fluids. $G_{\text{pres}}$ predicts the pressure. $G_{\text{col}}$ handles collision between particles. Among them, $G_{\text{adv}}$ and $G_{\text{pres}}$ are node-focused graph networks, and $G_{\text{col}}$ is an edge-focused graph network. **(b)** In the node-focused network, input are represented as node features and then updated recursively via particle-wise message passing. **(c)** In the edge-focused network, input are represented as edge features in a directed graph, and then updated recursively via particle-wise message passing.

### 3.3. Node-focused graph network

Node-focused network transforms and passes information that lies on nodes. The targets of it are per-particle quantities that can be represented as node features, such as pressure and body-force. Therefore, advection net and pressure net are designed as node-focused networks.

In the node-focused network, considering a particle $i$, the input node features $\mathbf{f}_i$ are first encoded into high dimensional encoding $\mathbf{h}_i^{(0)}$ via a learnable encoder $\epsilon^V$:

$$\mathbf{h}_i^{(0)} = \epsilon^V(\mathbf{f}_i), \tag{5}$$

where the encoder is implemented as multi-layer perceptron (MLP). The encoding process aims to fully leverage the expressiveness of neural networks by lifting features into high-dimensional embedding. The node embedding is then updated based on a recursive message passing scheme, where center node $i$ receives messages $\mathbf{m}_{j\to i}$ from its neighbor nodes $j$, $\forall j \in \mathcal{N}(i)$. The message is conditioned on the distance between two particles represented by node $i$ and $j$, along with node embedding from source node $j$:

$$\mathbf{m}_{j\to i}^{(l)} = \mathbf{h}_j^{(l-1)} \odot W\left(\left\|\mathbf{x}_i - \mathbf{x}_j\right\|_2, r_0\right), \tag{6}$$

where $\mathbf{h}_j^{(l-1)}$ denotes the node embedding of neighbor node $j$, $\odot$ denotes element-wise multiplication, $W(r, r_0)$ denotes the weight function with cutoff $r_0$ (i.e. $W(r, r_0) = 0, \forall r > r_0$), and $\left\|\mathbf{x}_i - \mathbf{x}_j\right\|_2$ denotes the Euclidean distance between node $i$ and $j$. After calculating the messages, they are aggregated together via a aggregation function, which is defined as:

$$\mathbf{M}_i^{(l)} = \frac{\sum_j \mathbf{m}_{j\to i}^{(l)}}{\sum_j W\left(\left\|\mathbf{x}_i - \mathbf{x}_j\right\|, r_0\right)}, \forall j \in \mathcal{N}(i). \tag{7}$$

Note that here our design of message and aggregation function is analogous to the kernel interpolation model in SPH, which assure a smooth response under varying neighborhoods and reduce the total number of model parameters compared to fully MLP-based message passing function [17,21]. Using distance-based weights when aggregating neighborhood features are a widely adopted technique in point cloud deep learning, as it incorporates a very important inductive bias: the dependency between a pair of particles is closely related to their distance and there are stronger dependency between closer particles. For instance, PointNet++ [41] uses the inverse of squared distance as the weight function for point cloud feature interpolation, SchNet [42] proposes a learnable weight function conditioned on inter-atomic distance. Similar to our work, Roy et al. [34] also uses SPH-based kernel as weight function to aggregate neighbor features. The main difference is that Roy et al. [34] use kernel to calculate weighted average of neighborhood coordinates, while we use kernel as weight for neural messages sent by neighbor nodes.

Finally, the node embedding of center node $i$ is updated with a learnable function $\Phi$:

$$\mathbf{h}_i^{(l)} = \Phi^{(l)}\left(\mathbf{h}_i^{(l-1)}, \mathbf{M}_i^{(l)}\right). \tag{8}$$

After $L$ layers of recursive message passing, the final prediction $\mathbf{o}_i$ is derived by decoding the final node embedding $\mathbf{h}_i^{(L)}$ via a learnable decoder $\gamma^V$:

$$\mathbf{o}_i = \gamma^V(\mathbf{h}_i^{(L)}). \tag{9}$$

For advection net, the prediction $\mathbf{o}_i$ is the per-particle advection forces (net sum of viscosity and body force). In pressure net, the prediction $\mathbf{o}_i$ is the pressure of each particle.

### 3.4. Edge-focused graph network

Contrast to the node-focused network, edge-focused network uses edge features as input and predicts the interaction between

nodes. It can be used to reason about pairwise relationship or directional quantities that depends on a pair of nodes. For instance, when two particles exchange momentum through collision, the momentum exchange can be represented as directed edges between two nodes. To prevent particle penetration and improve simulation stability, we use an edge-focused network to predict the collision between particles.

Similar to node-focused network, edge-focused network also adopts an encoding – recursive message passing – decoding scheme. Given a pair of particles $i$ and $j$, the learnable edge encoder $\epsilon^E$ encodes input edge features $\mathbf{q}_{ij}$ into initial edge encoding $\mathbf{e}_{ij}^{(0)}$,

$$\mathbf{e}_{ij}^{(0)} = \epsilon^E(\mathbf{q}_{ij}). \tag{10}$$

The initial node embedding $\mathbf{h}_i^{(0)}$ is derived by aggregating the edge embeddings:

$$\mathbf{h}_i^{(0)} = \sum_j \mathbf{e}_{ij}^{(0)}, \forall j \in \mathcal{N}(i). \tag{11}$$

Then the edge embeddings and node embeddings are updated recursively via message passing. The message $\mathbf{m}_{j\to i}^{(l)}$ on the edge $j \to i$ is defined as:

$$\mathbf{m}_{j\to i}^{(l)} = \Theta^{(l)}(\mathbf{e}_{ij}^{(l-1)}, \mathbf{h}_j^{(l-1)}). \tag{12}$$

The messages are aggregated via summation:

$$\mathbf{M}_i^{(l)} = \sum_j \mathbf{m}_{j\to i}^{(l)}, \forall j \in \mathcal{N}(i). \tag{13}$$

The edge embeddings and node embeddings are then updated as:

$$\mathbf{e}_{ij}^{(l)} = \mathbf{m}_{j\to i}^{(l)}, \tag{14}$$

$$\mathbf{h}_i^{(l)} = \Phi^{(l)}(\mathbf{h}_i^{(l-1)}, \quad \mathbf{M}_i^{(l)}). \tag{15}$$

The $\epsilon^E$, $\Theta^{(l)}$ and $\Phi^{(l)}$ in the above equations are learnable functions implemented as MLPs.

At the final layer, the edge embeddings $\mathbf{e}_{ij}^{(L)}$ are decoded into the prediction of influence $\mathbf{u}_{ij}$ imposed by node $j$ on node $i$ via learnable decoder $\gamma^E$.

$$\mathbf{u}_{ij} = \gamma^E(\mathbf{e}_{ij}^{(L)}) \tag{16}$$

The net total sum of the influence is:

$$\mathbf{U}_i = \sum_j \mathbf{u}_{ij}, \forall j \in \mathcal{N}(i) \tag{17}$$

In collision net, the momentum exchange between two particles can be interpreted as $\mathbf{u}_{ij}$ in the (16), and the resulted net momentum change can be interpreted as $\mathbf{U}_i$ in the (17).

In general, the message passing in edge-focused network is similar to the one in node-focused network. The main differences lie in two aspects. First, there is no edge embedding being updated recursively in the node-focused network. Second, the neighbor-to-center message ($\mathbf{m}_{j\to i}$) in the node-focused network conditioned only on the Euclidean distance and neighbor node features, while edge-focused network adopts a more flexible message function defined upon edge features and neighbor node features. By extending the graph topology beyond Euclidean space, edge-focused network can learn and reason about more complex relationships between particles. However, this also makes it more computational intensive as message passing in the edge-focused network is more complex and involves more calculation.

### 3.5. Implementation

All the learnable functions including encoder, decoder of embeddings and message function are implemented as multi-layer perceptrons (MLPs). All the encoder/decoder MLP has three layers, with encoder's shape as: [Input channels, 32, 64], and decoder has a shape: [64, 32, Output channels]. Intermediate message functions are MLPs with shape: [64, 32, 64]. We adopt Rectified Linear Units (ReLUs) [43] as the non-linear activation function for each hidden layer. For advection net and pressure net, the number of recursive message passing layers is 2. For collision net, we find that 1 layer of message passing can already generate accurate prediction and adding more layers of message passing does not result in significant improvement, and therefore we use 1 layer of message passing in the collision net.

We adopt the numerical model in SPH to evaluate the physical quantities like particle number density, and differential operators including gradient and Laplacian. To construct graph representation for particles, we establish edges between all pair of particles $i, j$ within the cutoff radius $r_0$, such that $\|\mathbf{x}_i - \mathbf{x}_j\|_2 < r_0$. The cutoff radius is chosen such that each particle has roughly 10 to 20 neighbors. Each time step of simulation with FGN requires two rounds of neighbor search. The first neighbor search is used to build the graph for advection net and collision net. This graph is shared across advection net and collision net. The second neighbor search is performed after advection net and collision net, which is used to build the graph for pressure net.

### 3.6. Update scheme

Given the state (position $\mathbf{x}^n$ and velocity $\mathbf{v}^n$) of the current time step $n$, the state of next time step $n+1$ is derived by passing state information through advection net, collision net, and pressure net sequentially. The input features to advection net are positions and velocities of particles, $[\mathbf{x}^n, \mathbf{v}^n]$, along with $\mathbf{g}$, which indicates the external body force per mass of fluid, and viscosity parameter $\nu$. The advection net predicts acceleration of particles:

$$\mathbf{a}^{\text{adv}} = G_{\text{adv}}\left(\mathbf{x}^n, \mathbf{v}^n, \mathbf{g}, \nu\right), \tag{18}$$

and updates the state of fluid particles to an intermediate state $[\mathbf{x}^*, \mathbf{v}^*]$:

$$\mathbf{v}^* = \mathbf{v}^n + \mathbf{a}^{\text{adv}}\Delta t, \tag{19}$$

$$\mathbf{x}^* = \mathbf{x}^n + \mathbf{v}^*\Delta t, \tag{20}$$

where $\mathbf{a}^{\text{adv}} = [\mathbf{a}_1^{\text{adv}}, \ldots, \mathbf{a}_N^{\text{adv}}]$, $\mathbf{x}^n = [\mathbf{x}_1^n, \ldots, \mathbf{x}_N^n]$, $\mathbf{v}^n = [\mathbf{v}_1^n, \ldots, \mathbf{v}_N^n]$, $\forall i \in \{1, \ldots, N\}$, with $N$ the total number of particles in the system. We will use the same notation throughout illustration.

The collision net takes relative positions and velocities between particles $[\mathbf{x}_i^* - \mathbf{x}_j^*, \mathbf{v}_i^* - \mathbf{v}_j^*]$ as input, and predicts correction to the velocities:

$$\Delta\mathbf{v} = G_{\text{col}}(\mathbf{x}_r^*, \mathbf{v}_r^*), \tag{21}$$

where $[\mathbf{x}_r^*, \mathbf{v}_r^*]$ denotes the relative positions and velocities in intermediate state. The velocities are then updated with predicted correction, along with positions:

$$\mathbf{v}^{**} = \mathbf{v}^* + \Delta\mathbf{v}, \tag{22}$$

$$\mathbf{x}^{**} = \mathbf{x}^* + \Delta\mathbf{v}\Delta t. \tag{23}$$

This process will repel the particles that are getting too close to each other and improve overall stability.

In the final step, the updated intermediate positions and velocities are taken as input by the pressure net, along with particle number density $\eta$. The goal of pressure net is to impose a pressure projection to mitigate temporary compression in the fluid field introduced by advection.

$$\hat{\mathbf{p}} = G_{\text{pres}}(\mathbf{x}^{**}, \mathbf{v}^{**}, \eta). \tag{24}$$

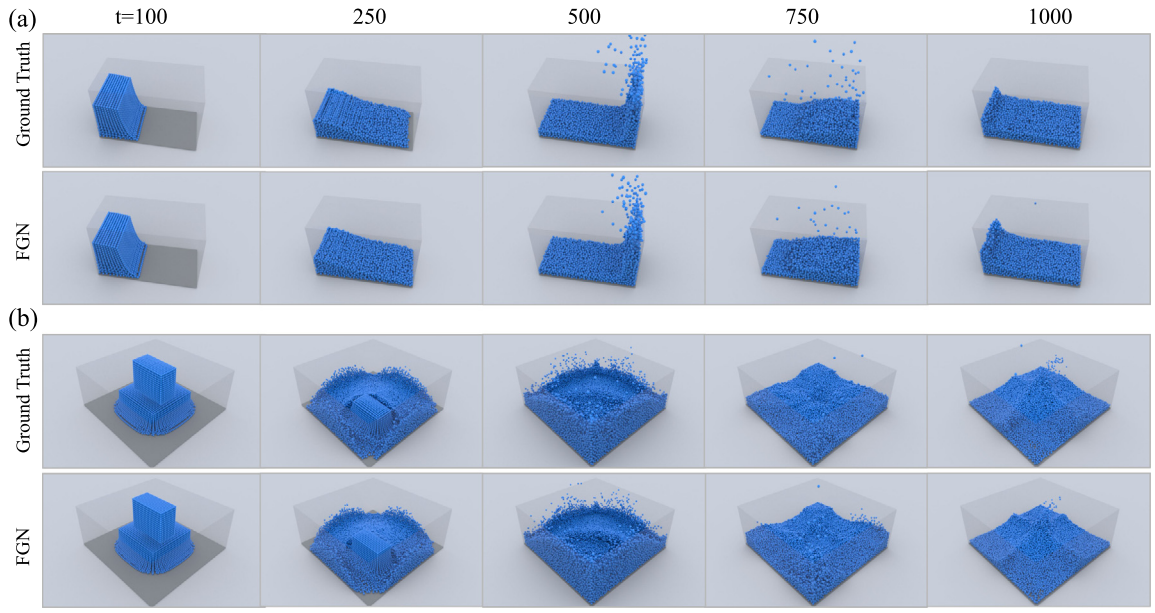



**Fig. 2.** Visualization on: **(a)** Dam collapse. **(b)** Water fall.

Finally, the state of fluid field is updated to the next time step $n+1$,

$$\mathbf{v}^{n+1} = \mathbf{v}^{**} - \frac{\nabla \hat{\mathbf{p}}}{\rho_0} \Delta t, \tag{25}$$

$$\mathbf{x}^{n+1} = \mathbf{x}^{**} + (\mathbf{v}^{n+1} - \mathbf{v}^{**}) \Delta t. \tag{26}$$

Where $\hat{\mathbf{p}} = [\hat{p}_1, \ldots, \hat{p}_N]$ denotes the pressure of fluid particles, $\boldsymbol{\eta} = [\eta_1, \ldots, \eta_N]$ denotes the particle number density of fluid particles, and $\rho_0$ denotes the density parameter of the fluids.

During the above calculation, the global positional information $\mathbf{x}$ is only used to construct the graph and will not be passed into graph networks as features.

## 4. Results

### 4.1. Training

*Dataset.* We use the Moving Particle Semi-implicit method (MPS) [25] with an improved pressure solver [44] to generate high-fidelity simulation data of incompressible flow. MPS is a numerical method based on SPH which prioritizes accuracy over calculation speed. It enforces the incompressibility of the fluid field by pressure projection based on the Poisson equation. We create 20 scenes by randomly placing fluid blocks, rigid body obstacles (cube, cylinder) in a cubic box, and initialize fluid particles with random velocities.

*Loss function and optimization.* We train three networks — advection net, collision net, and pressure net separately. Each network is trained in a supervised way by minimizing the mean squared error between prediction $\hat{y}$ and ground truth $y$.

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left\| y_i - \hat{y}_i \right\|_2^2 \tag{27}$$

Based on the above loss function, the parameters of the model are optimized with Adam [45] optimizer (more details on dataset and model training can be found in the supplementary information).

### 4.2. Metrics

We quantitatively evaluate model performance based on different metrics. We use the asymmetric Chamfer Distance to measure the spatial difference between the FGN simulated results and ground truth. Chamfer distance is a widely adopted distance metric for point clouds comparison, as it is permutation invariant and straightforward to evaluate. The asymmetric Chamfer distance for two collections of points $\mathcal{X}, \mathcal{Y}$ (from $\mathcal{X}$ to $\mathcal{Y}$) is defined as:

$$\mathcal{L}_{\text{CD}}(\mathcal{X}, \mathcal{Y}) = \frac{1}{N} \sum_{\mathbf{x} \in \mathcal{X}} \min_{\mathbf{y} \in \mathcal{Y}} d(\mathbf{x}, \mathbf{y}), \tag{28}$$

where $N$ is the total point number of point cloud collection $\mathcal{X}$, and here distance function $d(\mathbf{x}, \mathbf{y})$ is evaluated using L2-norm $\|\mathbf{x} - \mathbf{y}\|_2$. We further investigate two essential physical quantities for incompressible fluid simulation — velocity divergence and density deviation of the fluid field. Under SPH's formulation, the velocity divergence at location $\mathbf{r}_i$ can be approximated as:

$$\nabla \cdot \mathbf{v}_i = \frac{d}{\eta^*} \sum_{j \in \mathcal{N}(i)} (\mathbf{v}_j - \mathbf{v}_i) \cdot \nabla W\left(\left|\mathbf{r}_i - \mathbf{r}_j\right|, h\right), \tag{29}$$

where $\eta^*$ is the constant particle number density derived by calculating the maximum particle number density at the initial frame, $d$ is the dimension number. The density deviation of fluid field at location $\mathbf{r}_i$ is evaluated as:

$$\Delta\eta = \eta_i - \eta^*. \tag{30}$$

For perfectly incompressible flow, the density deviation and velocity divergence should remain 0.

We use normalized mean absolute error (MAE) and relative tolerance to evaluate the error of advection net, pressure net on prediction respectively. The normalized MAE of prediction $\hat{y}$ to ground truth $y$ is defined as:

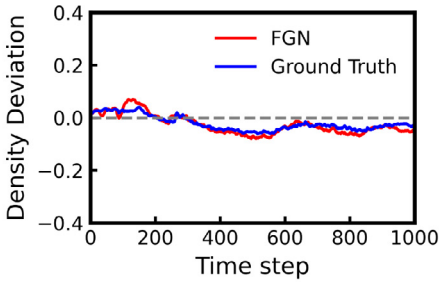
$$\mathcal{L}_{\text{MAE}} = \frac{1}{N} \sum_{i=1}^{N} \frac{|\hat{y}_i - y_i|}{|y_i|}. \tag{31}$$

The relative tolerance of the numerical solution $\hat{\mathbf{x}}$ to a linear system $A\mathbf{x} = \mathbf{b}$ is defined as:

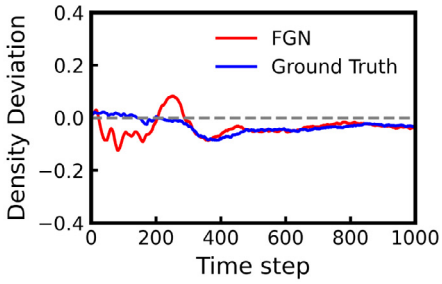
$$\mathcal{L}_{\text{tol}} = \frac{\left\| A\hat{\mathbf{x}} - \mathbf{b} \right\|_2}{\|\mathbf{b}\|_2}. \tag{32}$$

**Table 1**
Statistics of generated trajectories. For resolution, the dam collapse case contains 4k fluid particles and water fall case contains 25k fluid particles.

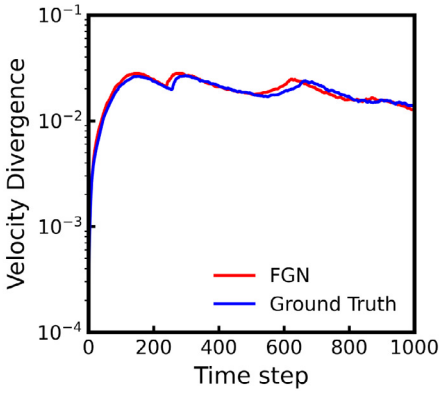
| Case | Model | Absolute density deviation | | Velocity divergence ($s^{-1}$) | | Chamfer distance (mm) | |
|---|---|---|---|---|---|---|---|
| | | Mean | Max | Mean | Max | Mean | Max |
| Dam collapse | FGN | 0.0461 | 0.0900 | 0.0195 | 0.0280 | 24.2 | 30.1 |
| | Ground truth | 0.0380 | 0.0710 | 0.0190 | 0.0268 | – | – |
| Water fall | FGN | 0.0541 | 0.1350 | 0.0207 | 0.0462 | 24.8 | 29.6 |
| | Ground truth | 0.0429 | 0.0966 | 0.0196 | 0.0398 | – | – |



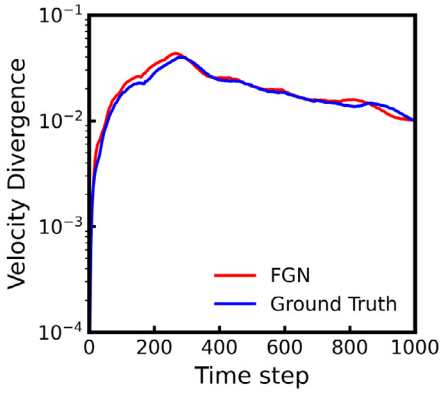

(a) Density deviation in the Dam collapse case.



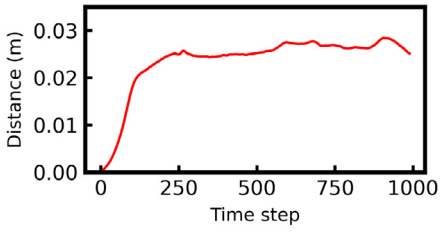

(b) Density deviation in the Water fall case.



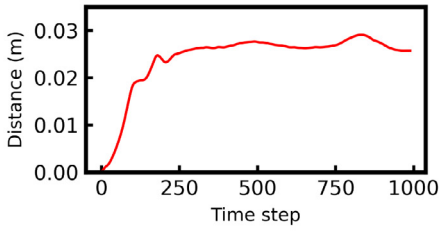

(c) Velocity divergence in the Dam collapse case.




(d) Velocity divergence in the Water fall case.




(e) Chamfer distance to ground truth in the Dam collapse case.




(f) Chamfer distance to ground truth in the Water fall case.

**Fig. 3.** Quantitative analysis of FGN generated trajectories.

### 4.3. Evaluation

To measure the performance of our proposed FGN model as a fluid simulator, we performed simulations on several different test cases (the rendered videos can be found in supplementary materials). In the first experiment, we conduct simulation on two cases, dam collapse and water fall. We qualitatively compare the generated trajectories of FGN to the ground truth (Fig. 2). The FGN generated trajectories in both cases are visually plausible and have little difference with ground truth. We then evaluate the results quantitatively based on metrics discussed in Section 4.2. For all the metrics, we report their mean and max value among all snapshots in a trajectory ( Table 1) and plot their changing trend during the simulation (Fig. 3). The results show that FGN retains crucial physical properties including low divergence and small density fluctuation. Specifically, as shown in Fig. 3(a), 3(b), 3(c), 3(d), the density deviation is close to 0 throughout the simulation and the velocity divergence is maintained at a relatively low level. In addition to the physics-based metrics, we use Chamfer distance to measure the difference of spatial distribution of two particle systems. As shown in the figures (Fig. 3(e), 3(f)), the positional error (difference between FGN's result and ground truth) accumulates at the initial stage due to the chaotic dynamics, but then converge to an equilibrium towards the end of the simulation.

In the second experiment, we study how our model will predict the pressure distribution of circular flow around a cylinder. The experiment scene contains an inflow on the left side which keeps emitting particles during the simulation and an outflow on the right side. The result (Fig. 4) shows that FGN can predict

**Table 2**
Quantitative analysis of advection net and pressure net.

| Model | Dynamical system | Model output | Evaluation metric | Error |
|---|---|---|---|---|
| Advection Net | $\dot{\mathbf{u}} = \mathbf{g} + \nu\nabla^2\mathbf{u}$ | prediction of $\dot{\mathbf{u}}$ | Normalized MAE | $12.4\% \pm 5.6\%$ |
| Pressure Net | $\nabla^2 p = \frac{1}{\Delta t}\nabla \cdot \mathbf{u}^*$ | prediction of $p$ | Relative tolerance | $8.3\% \pm 1.1\%$ |

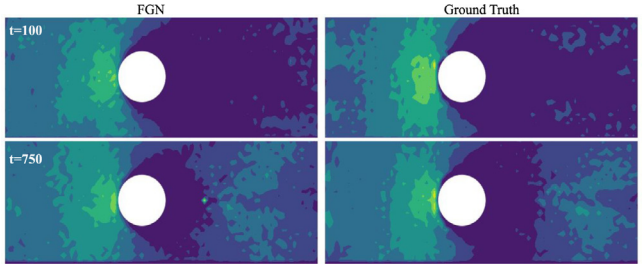
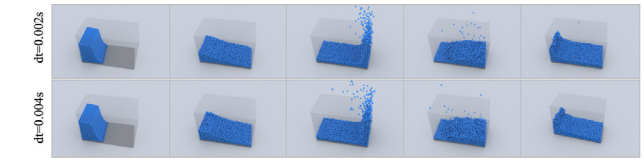
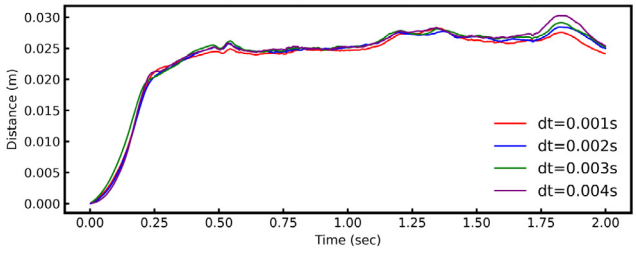



**Fig. 4.** Pressure distribution contour comparison.




(a) Visualization of trajectories generated using different time step sizes.




(b) Chamfer distance to the ground truth of trajectories generated using different time step sizes.

**Fig. 5.** Analysis of model under different time step sizes.

pressure distribution that is consistent with ground truth, where it captures the shape and shift of high pressure regions.

We further investigate the generalization capability of the model in two aspects — its adaptivity under different time step sizes and performance under more complex scenarios. The visualization and quantitative analysis of FGN under different time step sizes are shown in Fig. 5. The results show that FGN can generate consistent trajectories under different time step sizes despite it is trained on the data with a fixed time step size of $dt = 0.002s$. This demonstrates that FGN can be trained on small scale temporally fine-grained data to obtain high-accuracy, while during the test time it can simulate dynamics using a larger step size to reduce computational cost. We also challenge FGN with more complex geometries which are beyond the distribution of training data. Results (Fig. 6) show that, although the training data only contains basic geometries such as cube and cylinder, the trained FGN model can be extrapolated to complex irregular geometries and produce plausible results.

Finally, we study the performance of each sub-network in our model as stand-alone solvers for sub-dynamical systems, particularly the advection net and pressure net, as they are the sub-dynamical systems that dominates the fluid dynamics (collision net only affects the particles that get very close, it has

**Table 3**
Quantitative comparison of different message passing function.

| Message passing function | Relative MAE | |
|---|---|---|
| | Advection Net | Pressure Net |
| FGN | 0.124 | **0.013** |
| GCN [38] | 0.257 | 0.056 |
| GraphSAGE [10] | 0.232 | 0.043 |
| GN [17] | **0.108** | 0.021 |
| MLP | 0.374 | 0.064 |

minor influence on the overall fluid motion). For the advection net, we test it with a different set of material parameters (i.e. different gravity and viscosity parameters) and then report the normalized mean absolute error (MAE) between prediction and ground truth. For pressure net, we evaluate the relative tolerance of its predicted solution $\hat{\mathbf{p}}$ to the discretized pressure Poisson equation (i.e. $A\mathbf{p} = \mathbf{b}$). We generate ten trajectories with different set of material parameters (each containing 1000 snapshots) as test data for advection net. The gravity $\mathbf{g}$ for test data is uniformly chosen from 1.0 to 100.0 and viscosity parameter $\nu$ is uniformly chosen from 0.1 to 0.001. As for the test data of pressure net, we generate another ten trajectories (each containing 1000 snapshots) by initializing fluid block with random velocities and random positions. The result ( Table 2) shows that advection net can predict the viscosity and body force under a wide range of system parameters. Moreover, the prediction of pressure net can fit in with the Poisson equation with relatively low residuals, which accounts for the low velocity divergence and density deviation in FGN's generated trajectories.

### 4.4. Ablation study

In this section, we present the analysis on the effect of different architectural choices. We analyze two key hyperparameters in our model, the number of recursive message passing layers and different choices of message passing function. The test data used in this section is same as the data used for evaluating sub-networks in Section 4.3, except for the collision net, whose data is generated from a elastic collision system.

The first key hyperparameter we studied is the number of recursive message passing layer. The accuracy of different architectures is measured by the relative mean absolute error. Fig. 7 shows that increasing the layer number of message passing generally improves the inference accuracy across different sub-networks. By adding more layers of message passing, the network can capture longer-range dependencies between particles and the model gains stronger fitting capability with more parameters. However as the computational cost also increases when there are more layers of message passing in the network, we choose the number of layers such that increasing it will not bring in further significant performance improvement (i.e. $<\sim 5\%$).

We then investigate how different message passing mechanisms influence the performance by using different message passing functions in advection net and pressure net (as discussed in Section 4.3, collision net has minor influence on overall fluid motion). Specifically, we compare FGN's message passing mechanism with standard Graph Convolutional Neural Networks (GCN) [38],

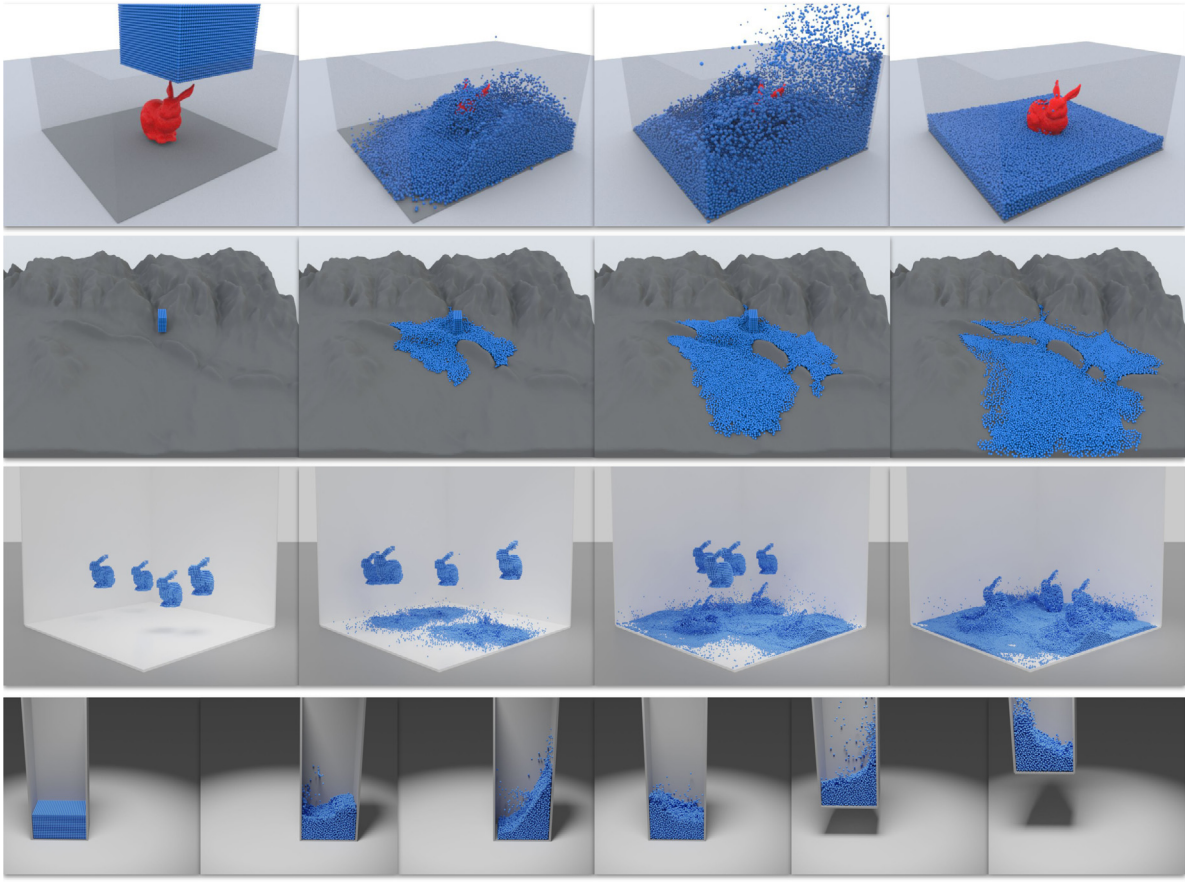


**Fig. 6.** Generalization to different scenarios. **Top row**: A fluid block drops on Stanford bunny and converges to an equilibrium. **Second row**: A fluid emitter is placed at the front of Grand Canyon 3D model and emitted fluids gradually fill up reservoir at downstream. **Third row**: Viscous fluid blocks (viscosity parameter $\nu = 0.1$) with Stanford bunny shape dropping onto the ground. **Bottom**: A fluid container moving towards different directions.

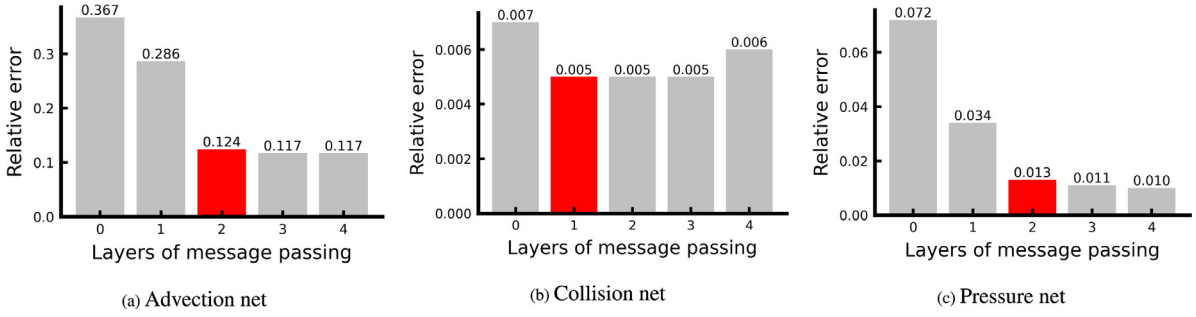



**Fig. 7.** Ablation of message passing layer numbers on different sub-networks.

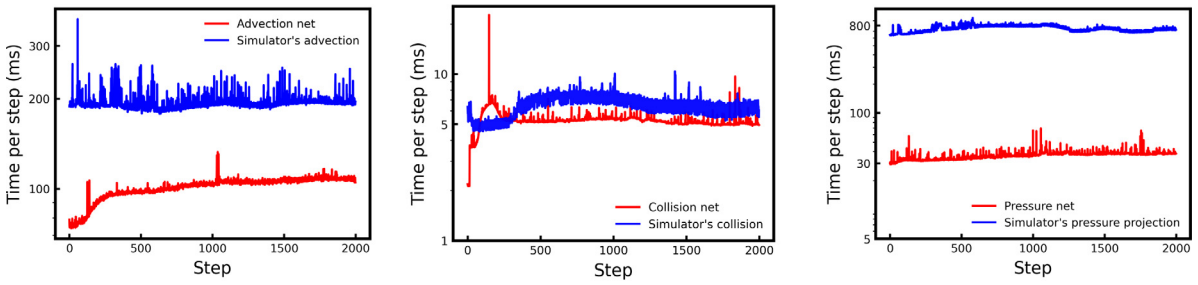
GraphSAGE [10], Graph Networks (GN) [17] and pure MLP architecture without message passing around neighbor nodes. As shown in the Table 3, the message passing function in FGN outperforms standard GCN, MLP, GraphSAGE, and is comparable to GN with much less trainable parameters.

We also analyze the influence of different training strategies and architectural design by merging each sub-network together. We evaluate the position error of next two predicted frames by calculating the mean squared distance from predicted result to the ground truth, and use Chamfer distance to evaluate the accuracy of long-term simulation.

As advection net and collision net shares the same graph, a natural design choice would be merging them into a single network. As shown in the Table 4, without the supervision of intermediate labels, the performance of FGN drops on all the metrics. If all the sub-networks are merged together, the performance

**Table 4**
Ablation of different training strategy. [*A*, *B*] indicates two sub-network A, B are merged together and the original training label for A is no longer used. The position MSE is normalized by the closest inter-particle distance at the initial frame. The average Chamfer distance is evaluated on the 1000-frame rollout sequence of testing scenes.

| Network | Position MSE ($\times 10^{-6}$) | | Average Chamfer distance (mm) |
|---|---|---|---|
| | n+1 | n+2 | |
| $Adv + Col + Prs$ | **1.414** | **5.811** | **25.0** |
| $[Adv,\ Col] + Prs$ | 2.192 | 8.335 | 26.4 |
| $[Adv,\ Col,\ Prs]$ | 3.206 | 16.419 | 32.3 |




(a) Advection (b) Collision (c) Pressure

**Fig. 8.** Runtime per step of different sub-networks and corresponding components in the ground truth simulator. The $Y$-axis for all plots are shown in log scale.

**Table 5**
Average running time (ms) per step for each component of the calculation.

| Model | Advection | | Collision | | Pressure | |
|---|---|---|---|---|---|---|
| | Neighbor | Calculation | Neighbor | Calculation | Neighbor | Calculation |
| FGN | 93.3 | 6.5 | 3.4 | 1.7 | 28.7 | 8.0 |
| MPS | 105.6 | 90.0 | 5.5 | 1.2 | 78.5 | 620.7 |

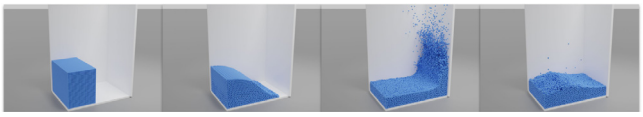


**Fig. 9.** The benchmark dam collapse scene, which contains 30k fluid particles.

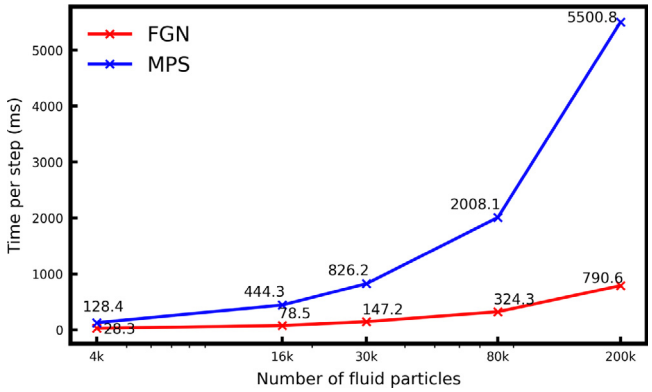



**Fig. 10.** Simulation time (ms) per step under different resolution, for each resolution the computation time is averaged over five simulations. $X$-axis is shown in log scale.

of FGN will deteriorate more significantly. This demonstrates the superiority of network decomposition and intermediate label supervision.

### 4.5. Benchmark

We analyze the computational efficiency of FGN by comparing its simulation speed against the ground truth simulator — MPS. We implement the iterative pressure solver in MPS based on publicly available PyTorch conjugate gradient (CG) solver.[2] During the benchmark, we set the absolute tolerance of CG solver to be 0.1 and the maximum iteration to be 10 (note that these hyperparameters are set to be $1.0 \times 10^{-5}$ and 100 respectively when used to generate training data). The benchmark simulation was run on a dam collapse scenario with 30k fluid particles as shown in Fig. 9 (for runtime report of scenes presented in Fig. 6, we refer reader to the Supplementary Information). We use a platform equipped with an i7-8700K CPU and a GTX 1080Ti GPU to carry out the experiment. FGN's message passing on graph is implemented with Deep Graph Library (DGL) [46]. All the calculation are mainly carried out on the GPU.

Fig. 8 shows the computing time each sub-network takes for every step. Table 5 shows the breakdown analysis of averaged computation time. For advection net and pressure net in FGN, *Neighbor* includes the time of: updating neighbor lists, converting them into DGL's sparse adjacency matrix and evaluating kernel value $W(r_{ij})$ on every edge. For all sub-networks, *Calculation* comprises the time used to derive input features like particle number density, the forward pass on the network and post processing such as evaluating the gradient of pressure. Note that in FGN, collision net shares the same graph with advection net (which means no neighbor list update is performed), but drop the edges between particles that are larger than the collision cutoff and recalculate the inter-particle distance using updated position. For MPS, the calculation in pressure projection includes calculating velocity divergence, assembling matrix equation, solving pressure Poisson equation using a iterative conjugate gradient solver and evaluating pressure gradient. Specifically, the iterative solving scheme accounts for most of the calculation time in MPS' pressure projection, which takes 580.4 ms in average for each time step.

As shown in the results, the most significant improvement comes from the pressure net. Compared to iterative pressure solver, pressure net does not require iteration in the inference stage. In addition, pressure net uses a smaller cutoff ($0.5\times$) when constructing neighbor list, this in turn makes pressure net operates on a much smaller neighborhood (around 4–8 times fewer particles) and thus requires less kernel value evaluation and lower neighbor searching time. For advection net and collision net, they contribute approximately $2\times$ speed-up in total. A general computational advantage of sub-networks (advection net, pressure net) in FGN is that they require a smaller neighborhood than the counterpart in physics-based simulator, as in the graph neural networks longer range interaction between particles can be captured by multiple round of recursive message passing.

Next we analyze FGN's calculation time for each step on different resolution. For each resolution, we run five simulation by placing the cubic fluid block at a random location inside the container, and initialize all fluid particles with a random velocity. Fig. 10 shows the runtime statistics of MPS and FGN under different resolution. Overall, FGN achieves ∼5–8 times acceleration over MPS under different resolution.

[2] https://github.com/sbarratt/torch_cg.

**Table 6**
Runtime complexity analysis of different learning-based simulation models and point cloud deep learning architectures on the dam collapse scene as shown in Fig. 9. The running time is averaged over a simulation sequence containing 2000 snapshots.

| Model | # Param. (M) | Network time (ms) | Neighbor search (ms) | Total (ms) |
|---|---|---|---|---|
| FGN | **0.05** | **16.3** | 125.5 | 141.8 |
| GNS-small [12] | 1.12 | 185.5 | 73.1 | 258.6 |
| CConv [15] | 0.69 | 84.3 | 71.5 | 155.8 |
| PointNet++ [41] | 0.96 | 39.0 | **15.3** | **54.3** |
| DGCNN[a] [47] | 1.81 | 41.3 | 304.8 | 346.1 |
| FlowNet3D [48] | 3.75 | 47.5 | 24.6 | 72.1 |

[a]Due to the memory limit, the testing on DGCNN is conducted by decomposing each snapshot into 8 patches, with each patch center picked by farthest point sampling.

**Table 7**
Accuracy of different data-driven models on test scenes. For CConv, we use the pre-trained weights provided in the official Github repo. Both FGN and GNS-small adopt a time step size of $dt = 2$ ms and use forward Euler scheme to update. CConv adopts a time step size of $dt = 20$ ms and directly updates position at each step.

| Model | FGN | GNS-small | CConv |
|---|---|---|---|
| Chamfer distance (mm) | **25.0** | 25.4 | 30.1 |
| Training iterations | 30k | 100k | 50k |

We also compare FGN's running time to two data-driven simulators — Graph Network Simulator (GNS) [12], a generic particle-based simulator and Continuous Convolution (CConv) [15], a Lagrangian fluid simulator based on continuous convolution, along with three commonly used point cloud deep learning models — PointNet++ [41], Dynamic Graph CNN (DGCNN) [47] and FlowNet3D [48]. Results are listed in Table 6. We report the number of trainable parameters of each model, average inference/neighbor searching time on each snapshot of a simulation sequence. The benchmark is run on the dam collapse scenario with 30k fluid particles (Fig. 9). We implement GNS using PyTorch [49] and PyTorch Geometric [50]. Due to the memory limit of GTX 1080 Ti, a truncated version of GNS is adopted, where it only contains 8 layers of message passing (10 in the original paper). For CConv, we adopt the publicly available official PyTorch implementation.[3] The implementation of PointNet++, DGCNN, FlowNet3D are based on an open-source PyTorch PointNet project[4] and use PyTorch3D [51] for $k$-nearest neighbor search.

As shown in Table 6, FGN has the smallest size among other point-based deep learning architectures, which in turn makes it has the fastest inference time. Moreover, Table 7 shows that FGN offers competitive performance with less training iterations required. In general, compared to other data-driven models (which use neural networks to surrogate the entire update step), the decomposition of a single forward step in the fluid dynamics into multiple simpler sub-dynamical systems enables FGN to predict the fluid motion accurately with a far less parameterized model. The major bottleneck of FGN lies in the neighbor searching, where FGN requires more time than the other models as it needs doing two rounds of neighbor search in each step. Note that PointNet++, FlowNet3D are much faster in terms of neighbor searching because at each layer only part of the points require neighbor querying and the whole point cloud will be downsampled when propagating through layers, yet this will result in neighbor information loss compared to models that retain full resolution through the network. DGCNN's neighbor searching is much slower because it retains full resolution when doing neighbor search and needs to query $k$-nearest neighbor in the high-dimensional embedding space.

[3] https://github.com/intel-isl/DeepLagrangianFluids.
[4] https://github.com/erikwijmans/Pointnet2_PyTorch.

## 5. Conclusion

In this paper, we present a data-driven Lagrangian fluid model for incompressible fluid simulation, which decomposes simulation scheme into separate sub-steps based on Navier–Stokes equation. The physics-based architecture makes the model can preserve many essential physical properties, such as low volume compression, and reasonable pressure distribution. Our model's generalization capability also enable it to remain stable and accurate when extrapolating to a wide range of different geometries and can adapt to different time step sizes. Furthermore, we show that our proposed method improves computational efficiency compared to both classical method and other data-driven models. In general, our work is an advance in learning on unstructured data with graph neural networks, and enriches the paradigm of combining learning-based methods with physical models.

## CRediT authorship contribution statement

**Zijie Li:** Conceived the manuscript, Performed the simulations, Developed the model, Wrote the manuscript. **Amir Barati Farimani:** Conceived the manuscript, Wrote the manuscript.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgment

This work is supported by the start-up fund provided by CMU Mechanical Engineering.

## Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cag.2022.02.004.

## References

[1] Ladický L'ubor, Jeong SoHyeon, Solenthaler Barbara, Pollefeys Marc, Gross Markus. Data-driven fluid simulations using regression forests. ACM Trans Graph 2015;34(6). http://dx.doi.org/10.1145/2816795.2818129.
[2] Tompson Jonathan, Schlachter Kristofer, Sprechmann Pablo, Perlin Ken. Accelerating Eulerian fluid simulation with convolutional networks. 2016, CoRR. arXiv:1607.03597.
[3] Xiao X, Zhou Y, Wang H, Yang X. A novel CNN-based Poisson solver for fluid simulation. IEEE Trans Vis Comput Graph 2020;26(03):1454–65. http://dx.doi.org/10.1109/TVCG.2018.2873375.
[4] Um Kiwon, Hu Xiangyu, Thuerey Nils. Liquid splash modeling with neural networks. 2018, arXiv:1704.04456.
[5] Wiewel Steffen, Becher Moritz, Thuerey Nils. Latent-space physics: Towards learning the temporal evolution of fluid flow. 2018, CoRR. arXiv:1802.10123.
[6] Morton Jeremy, Jameson Antony, Kochenderfer Mykel J, Witherden Freddie. Deep dynamical modeling and control of unsteady fluid flows. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, editors. Advances in neural information processing systems 31. Curran Associates, Inc. 2018, p. 9258–68, URL http://papers.nips.cc/paper/8138-deep-dynamical-modeling-and-control-of-unsteady-fluid-flows.pdf.
[7] Wang Z, Xiao D, Fang F, Govindan R, Pain CC, Guo Y. Model identification of reduced order fluid dynamics systems using deep learning. Internat J Numer Methods Fluids 2018;86(4):255–68. http://dx.doi.org/10.1002/fld.4416, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.4416. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.4416.

[8] Pant Pranshu, Doshi Ruchit, Bahl Pranav, Barati Farimani Amir. Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations. Phys Fluids 2021;33(10):107101. http://dx.doi.org/10.1063/5.0062546.

[9] Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The graph neural network model. IEEE Trans Neural Netw 2009;20(1):61–80.

[10] Hamilton William L, Ying Rex, Leskovec Jure. Inductive representation learning on large graphs. 2017, arXiv:1706.02216.

[11] Bapst V, Keck T, Grabska-Barwińska A, Donner C, Cubuk ED, Schoenholz SS, et al. Unveiling the predictive power of static structure in glassy systems. Nat Phys 2020;16(4):448–54. http://dx.doi.org/10.1038/s41567-020-0842-8.

[12] Sanchez-Gonzalez Alvaro, Godwin Jonathan, Pfaff Tobias, Ying Rex, Leskovec Jure, Battaglia Peter W. Learning to simulate complex physics with graph networks. 2020, arXiv:2002.09405.

[13] de Avila Belbute-Peres Filipe, Economon Thomas D, Kolter J Zico. Combining differentiable PDE solvers and graph neural networks for fluid flow prediction. 2020, arXiv:2007.04439.

[14] Ogoke Francis, Meidani Kazem, Hashemi Amirreza, Farimani Amir Barati. Graph convolutional neural networks for body force prediction. 2020, arXiv:2012.02232.

[15] Ummenhofer Benjamin, Prantl Lukas, Thuerey Nils, Koltun Vladlen. Lagrangian fluid simulation with continuous convolutions. In: International conference on learning representations. 2020.

[16] Pfaff Tobias, Fortunato Meire, Sanchez-Gonzalez Alvaro, Battaglia Peter W. Learning mesh-based simulation with graph networks. 2021, arXiv:2010.03409.

[17] Battaglia Peter W, Pascanu Razvan, Lai Matthew, Rezende Danilo Jimenez, Kavukcuoglu Koray. Interaction networks for learning about objects, relations and physics. 2016, CoRR. arXiv:1612.00222.

[18] Chang Michael B, Ullman Tomer, Torralba Antonio, Tenenbaum Joshua B. A compositional object-based approach to learning physical dynamics. 2016, CoRR. arXiv:1612.00341.

[19] Li Yunzhu, Wu Jiajun, Tedrake Russ, Tenenbaum Joshua B, Torralba Antonio. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. 2018, CoRR. arXiv:1810.01566.

[20] Mrowca Damian, Zhuang Chengxu, Wang Elias, Haber Nick, Fei-Fei Li F, Tenenbaum Josh, et al. Flexible neural representation for physics prediction. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, editors. Advances in neural information processing systems 31. Curran Associates, Inc. 2018, p. 8799–810, URL http://papers.nips.cc/paper/8096-flexible-neural-representation-for-physics-prediction.pdf.

[21] Gilmer Justin, Schoenholz Samuel S, Riley Patrick F, Vinyals Oriol, Dahl George E. Neural message passing for quantum chemistry. 2017, arXiv:1704.01212.

[22] Monaghan JJ. An introduction to SPH. Comput Phys Comm 1988;48(1):89–96. http://dx.doi.org/10.1016/0010-4655(88)90026-4, URL http://www.sciencedirect.com/science/article/pii/0010465588900264.

[23] Müller Matthias, Charypar David, Gross Markus. Particle-based fluid simulation for interactive applications. In: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on computer animation. SCA '03, Goslar, DEU: Eurographics Association; 2003, p. 154–9.

[24] Becker Markus, Teschner Matthias. Weakly compressible SPH for free surface flows. In: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on computer animation. SCA '07, Goslar, DEU: Eurographics Association; 2007, p. 209–17.

[25] Koshizuka S, Oka Y. Moving-particle semi-implicit method for fragmentation of incompressible fluid. Nucl Sci Eng 1996;123(3):421–34. http://dx.doi.org/10.13182/NSE96-A24205.

[26] Solenthaler B, Pajarola R. Predictive-corrective incompressible SPH. In: ACM SIGGRAPH 2009 papers. SIGGRAPH '09, New York, NY, USA: Association for Computing Machinery; 2009, http://dx.doi.org/10.1145/1576246.1531346.

[27] Ihmsen Markus, Cornelis Jens, Solenthaler Barbara, Horvath Christopher, Teschner Matthias. Implicit incompressible SPH. IEEE Trans Vis Comput Graphics 2014;20(3):426–35. http://dx.doi.org/10.1109/TVCG.2013.105.

[28] Bender Jan, Koschier Dan. Divergence-free smoothed particle hydrodynamics. In: Proceedings of the 2015 ACM SIGGRAPH/Eurographics symposium on computer animation. ACM; 2015, http://dx.doi.org/10.1145/2786784.2786796.

[29] Zhang Yalan, Ban Xiaojuan, Du Feilong, Di Wu. FluidsNet: End-to-end learning for Lagrangian fluid simulation. Expert Syst Appl 2020;152:113410. http://dx.doi.org/10.1016/j.eswa.2020.113410, URL https://www.sciencedirect.com/science/article/pii/S0957417420302347.

[30] Xie You, Franz Erik, Chu Mengyu, Thuerey Nils. TempoGAN: A temporally coherent, volumetric GAN for super-resolution fluid flow. ACM Trans Graph 2018;37(4). http://dx.doi.org/10.1145/3197517.3201304.

[31] Kochkov Dmitrii, Smith Jamie A, Alieva Ayya, Wang Qing, Brenner Michael P, Hoyer Stephan. Machine learning–accelerated computational fluid dynamics. Proc Natl Acad Sci 2021;118(21). http://dx.doi.org/10.1073/pnas.2101784118, arXiv:https://www.pnas.org/content/118/21/e2101784118.full.pdf. URL https://www.pnas.org/content/118/21/e2101784118.

[32] Um Kiwon, Brand Robert, Fei Yun, Holl Philipp, Thuerey Nils. Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. 2021, arXiv:2007.00016.

[33] Bai Kai, Li Wei, Desbrun Mathieu, Liu Xiaopei. Dynamic upsampling of smoke through dictionary-based learning. 2019, arXiv:1910.09166.

[34] Roy Bruno, Poulin Pierre, Paquette Eric. Neural upflow. Proc ACM Comput Graph Interact Tech 2021;4(3):1–26. http://dx.doi.org/10.1145/3480147.

[35] Battaglia Peter W, Hamrick Jessica B, Bapst Victor, Sanchez-Gonzalez Alvaro, Zambaldi Vinícius Flores, Malinowski Mateusz, et al. Relational inductive biases, deep learning, and graph networks. 2018, CoRR. arXiv:1806.01261.

[36] Kipf Thomas, Fetaya Ethan, Wang Kuan-Chieh, Welling Max, Zemel Richard. Neural relational inference for interacting systems. 2018, arXiv:1802.04687.

[37] Sanchez-Gonzalez Alvaro, Heess Nicolas, Springenberg Jost Tobias, Merel Josh, Riedmiller Martin A, Hadsell Raia, et al. Graph networks as learnable physics engines for inference and control. 2018, CoRR. arXiv:1806.01242.

[38] Kipf Thomas N, Welling Max. Semi-supervised classification with graph convolutional networks. 2016, CoRR. arXiv:1609.02907.

[39] Veličković Petar, Cucurull Guillem, Casanova Arantxa, Romero Adriana, Liò Pietro, Bengio Yoshua. Graph attention networks. 2018.

[40] Batchelor GK. An introduction to fluid dynamics. Cambridge mathematical library, Cambridge University Press; 2000, http://dx.doi.org/10.1017/CBO9780511800955.

[41] Qi Charles R, Yi Li, Su Hao, Guibas Leonidas J. PointNet++: Deep hierarchical feature learning on point sets in a metric space. 2017, arXiv:1706.02413.

[42] Kristof T Schütt, Kindermans Pieter-Jan, Sauceda Huziel E, Chmiela Stefan, Tkatchenko Alexandre, Müller Klaus-Robert. Schnet: a continuous-filter convolutional neural network for modeling quantum interactions. 2017.

[43] Nair Vinod, Hinton Geoffrey E. Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th international conference on international conference on machine learning. ICML'10, Madison, WI, USA: Omni Press; 2010, p. 807–14.

[44] Lee Byung-Hyuk, Park Jong-Chun, Kim Moo-Hyun, Hwang Sung-Chul. Step-by-step improvement of MPS method in simulating violent free-surface motions and impact-loads. Comput Methods Appl Mech Engrg 2011;200(9):1113–25. http://dx.doi.org/10.1016/j.cma.2010.12.001, URL http://www.sciencedirect.com/science/article/pii/S0045782510003464.

[45] Kingma Diederik P, Ba Jimmy. Adam: A method for stochastic optimization. 2014, arXiv:1412.6980.

[46] Wang Minjie, Zheng Da, Ye Zihao, Gan Quan, Li Mufei, Song Xiang, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. 2019, arXiv preprint arXiv:1909.01315.

[47] Wang Yue, Sun Yongbin, Liu Ziwei, Sarma Sanjay E, Bronstein Michael M, Solomon Justin M. Dynamic graph CNN for learning on point clouds. 2019, arXiv:1801.07829.

[48] Liu Xingyu, Qi Charles R, Guibas Leonidas J. FlowNet3D: Learning scene flow in 3D point clouds. 2019, arXiv:1806.01411.

[49] Paszke Adam, Gross Sam, Massa Francisco, Lerer Adam, Bradbury James, Chanan Gregory, et al. PyTorch: An imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché Buc F, Fox E, Garnett R, editors. Advances in neural information processing systems 32. Curran Associates, Inc. 2019, p. 8024–35, URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[50] Fey Matthias, Lenssen Jan E. Fast graph representation learning with PyTorch Geometric. In: ICLR workshop on representation learning on graphs and manifolds. 2019.

[51] Ravi Nikhila, Reizenstein Jeremy, Novotny David, Gordon Taylor, Lo Wan-Yen, Johnson Justin, et al. Accelerating 3D deep learning with PyTorch3D. 2020, arXiv:2007.08501.