Accepted for publication on 23rd April 2025.

International Journal of Numerical Methods for Heat and Fluid Flow.

DOI: https://doi.org/10.1108/HFF-10-2024-0800

Graph Network Simulators (GNS) for Modelling Fluid Flow with a Given Inlet Velocity.

Philip Pe[1] and Rajesh S. Ransing[1,*]

[1]Zienkiewicz Institute For Modelling, Data and AI

[*]Department of Mechanical Engineering

Swansea University, Swansea, SA1 8EN, United Kingdom

[*]Corresponding author: r.s.ransing@swansea.ac.uk

Abstract

Purpose: This paper introduces a Particle Trickle Release (PTR) algorithm for implementing an inlet velocity boundary condition in Graph Network Simulators (GNS) and explores the ability of GNS to extrapolate and apply the learned fluid dynamics to unseen, out-of-distribution examples.

Methodology: The study uses the 'WaterRamps' training dataset, which provides essential parameters for fluid particles. The training of the GNS is conducted using both the existing dynamics bootstrapping method and a sequential training approach to assess their effectiveness in capturing fluid dynamics accurately. The PTR algorithm is introduced to ensure realistic particle inflows at boundaries, calculated using a binomial distribution based on inflow velocity and inlet boundary length.

Results: The PTR algorithm demonstrated realistic particle release with minimal errors in particle count and area consistency compared to theoretical values. Sequential training resulted in a mean squared error (MSE) of 13.9×10^{-3}, slightly higher than the 12.9×10^{-3} achieved with dynamics bootstrapping. The study also highlights challenges in maintaining incompressibility conditions and the tendency to learn excessive wall friction, which leads to undesired boundary layer development, particularly in out-of-distribution simulations such as the 'WaterVortex' example and flow over a backward-facing step.

Originality: This paper contributes to the field of graph network-based fluid flow modelling by facilitating the implementation of inlet velocity conditions through the PTR algorithm and evaluating the effectiveness of sequential training. The degree of compressibility is assessed using a newly proposed velocity divergence term, and a 'push particle' algorithm is introduced to improve the quality of particle distribution.

Keywords: Deep learning, Data-driven modelling, Physics-informed modelling, Particle-based fluid flow, Lagrangian fluid flow, Graph Neural Networks.

Nomenclature

GNS Training

Particles in a Scenario

- $\mathcal{P}_{s_i} = \{p_{i1}, p_{i2}, \ldots, p_{in_i}\}$: The set of n_i particles in i^{th} scenario s_i.
- p_{ij}: A single particle in scenario s_i, where j ranges from 1 to n_i.
- n_i: The number of particles in scenario s_i.

Properties of a Particle

- $p_{ij}(t) = (x_{ij}(t), y_{ij}(t))$: The properties of particle p_{ij} at time t.
- $x_{ij}(t)$: The x-position of particle p_{ij} at time t.
- $y_{ij}(t)$: The y-position of particle p_{ij} at time t.
- $v_{ij}(t)$: The velocity of particle p_{ij} at time t calculated using its history $H_{p_{ij}}(t)$.
- $a_{ij}(t)$: The acceleration of particle p_{ij} at time t calculated using its history $H_{p_{ij}}(t)$.
- $\hat{a}_{ij}(t)$: The true acceleration for particle p_{ij} at time t.

Time Steps

- $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$: The set of all time steps.
- t_k: A single time step in the set $\mathcal{T}$, where k ranges from 1 to T.
- T: The total number of time steps in a transient rollout.

Union Set $\mathcal{U}$

- $\mathcal{U} = \bigcup_{i=1}^{1000} \left(\bigcup_{j=1}^{n_i} \{p_{ij}(t_k) \mid t_k \in \mathcal{T}\} \right)$: The union set containing all scenarios, particles, and their properties over all time steps.
- $\mathcal{B}(t)$: A batch of all particles from a scenario at time t along with five previous time steps (history) selected from $\mathcal{U}$

PTR Algorithm

x_0^b, y_0^b**:** Coordinates of the start point of the inlet linear boundary.

x_1^b, y_1^b**:** Coordinates of the end point of the inlet linear boundary.

n_r**:** Number of reference points (default is 1000).

N**:** Total number of particles to be added in the computational domain.

T: Total number of time steps.

v_{in}: Inflow velocity.

Δt: Time step size.

L: Length of the inlet boundary.

A_p: Area of each particle.

μ: Mean number of particles added per time step.

σ: Standard deviation of the number of particles added per time step.

P_{inlet}: Array of 1000 predefined positions with noise along the inlet boundary.

U: Union set of all particles within the computational domain with the corresponding history of 5 previous steps.

M_{T}: Set of new particles m_{t} to be added per time step t.

$p_i(t)$: Particle i added at time step t.

$x_i(t), y_i(t)$: Coordinates of particle i added at time step t.

$H_{p_i(t)}$: History of particle $p_i(t)$.

1 Introduction

Recent advancements in machine learning, particularly in deep learning models, have made significant contributions to the simulation of fluid dynamics. Graph Network Simulators (GNS) have gained prominence as an effective approach for modelling complex particle-based fluid flows. By leveraging graph structures, these models capture the relationships between particles, making them highly suitable for representing the intricate dynamics involved in fluid interactions. However, despite their successes, GNS models face challenges in handling boundary conditions, such as inlet velocity, and in generalising to out-of-distribution (OOD) scenarios.

In this paper, we address these challenges by introducing the Particle Trickle Release (PTR) algorithm, which enables the implementation of an inlet velocity boundary condition within GNS models. The PTR algorithm ensures more accurate particle inflows by employing a binomial distribution to account for inflow velocity and boundary length. Additionally, we compare the effectiveness of two training methodologies—sequential training and the commonly used dynamics bootstrapping method—in capturing fluid dynamics, with the aim of improving the generalisation capacity of GNS models. The goal of this research is not to replace traditional CFD with deep learning methods but rather to pursue curiosity-driven research exploring how particle-based Graph Network Simulators (GNS) can augment and generalise flow predictions.

The following sections will provide a review of the relevant literature. We begin by discussing the role of data-driven and physics-driven deep learning models in fluid dynamics, followed by a review of OOD generalisation, which is essential for improving the applicability of GNS models beyond the training datasets. An overview of particle-based fluid simulations is then presented, highlighting key approaches and their relevance to this study. Finally, we discuss the motivation for improving boundary conditions and training methodologies in GNS models, which forms the basis for the contributions made in this paper.

1.1 Data-driven and Physics-driven Deep Learning Models:

Deep learning models for fluid flow applications typically employ three primary network architectures: deep feedforward neural networks, convolutional neural networks (CNNs), and graph neural networks (GNNs). When deep feedforward neural networks are used to minimise the error in satisfying boundary conditions and governing differential equations, they are referred to as physics-informed neural networks (PINNs). Karniadakis et al. (2021) provided a comprehensive review of physics-informed learning in both forward and inverse problems, while Ghalambaz et al. (2024) further reviewed the most used PINN models.

Deep learning has found extensive applications across a wide range of fields, from heat transfer (Hachem et al. 2024, Edalatifar et al. 2024, Suresh & Chakravarthy 2024), porous media (Shi et al. 2024), two phase flow (Khan et al. 2024) to fluid flow and turbulence modelling (McConkey et al. 2024, Zhou et al. 2024, Zeng et al. 2024, Kochkov et al. 2021, Ishize et al. 2024). Typically, deep neural networks are designed to interpolate within the space defined by their training data. For instance, Löhner et al. (2021) compared the performance of deep learning models with interpolation methods in inverse heat transfer problems, demonstrating their effectiveness for both linear and non-linear behaviours.

Brunton et al. (2020), Brunton (2021) provided an extensive review of the potential of machine learning in fluid mechanics, highlighting the fundamental methodologies and their applications for understanding, modelling, optimising, and controlling fluid flows. They discussed both the strengths and limitations of these methods from a scientific perspective, considering data as an intrinsic element of modelling, experiments, and simulations.

Lino et al. (2023) classified deep learning methods for simulating fluid dynamics into two categories: data-driven and physics-driven approaches. Physics-driven methods, like PINNs, aim to minimise the residuals of governing partial differential equations, while data-driven methods are trained to reflect physical properties based on observed data. Lino et al. (2023) further noted that purely data-driven flow simulators often struggle with poor extrapolation and tend to accumulate errors in time-dependent simulations, classifying Graph Network Simulators (GNS)

as data-driven models.

However, this binary classification may be oversimplified. We argue that GNS models exhibit characteristics of both data-driven and physics-informed approaches. While GNS models rely on data-driven training, the training data itself is generated from physical equations, and the positions, velocities, and accelerations of the particles in the ground truth data are governed by physical laws. Moreover, GNS models, as demonstrated by Sanchez-Gonzalez et al. (2020), minimise the error between ground truth and predicted accelerations, enabling particles to learn to simulate physical behaviours. This has shown to significantly improve generalisation, particularly for out-of-distribution (OOD) examples.

Shukla et al. (2022) have argued that both PINNs and graph neural networks (which they refer to as PIGNs) can be physics-informed. Many complex problems in computational engineering, particularly those involving parameterised differential equations in mechanics, cannot be easily solved using standard numerical methods. However, progress can be made by reformulating these problems as large-scale minimisation tasks. In PINNs, automatic differentiation is used to enforce physical laws by penalising residuals across random points in the space-time domain. Similarly, in PIGNs, external graph calculus is used to penalise residuals on the nodes of a graph, making both approaches inherently physics-informed.

1.2 Out-of-Distribution (OOD) Generalisation:

Out-of-distribution (OOD) generalisation refers to the ability of deep learning models to perform well on unseen data that differs from the distribution of the training set. Goyal & Bengio (2022) emphasise that the inductive biases of deep learning methods enhance higher-level cognition, enabling models to generalise based on fewer examples and to effectively transfer knowledge to new tasks. This inductive reasoning allows deep learning methods to excel in OOD scenarios where conventional methods might fail.

Battaglia et al. (2018) explored how graph networks can facilitate generalisation and provide a robust framework for manipulating structured knowledge and producing structured behaviours. Their research demonstrated how graph networks support relational reasoning and combinatorial generalisation, offering a foundation for more sophisticated, interpretable, and flexible patterns of reasoning. This relational structure is particularly beneficial for fluid simulations, where the interactions between particles or elements are key to predicting complex behaviour.

OOD generalisation methods have been widely studied across various domains, such as in mechanical regression problems where researchers have focused on addressing covariate shift, mechanism shift, and sampling bias Yuan et al. (2022). These challenges are also present in modelling complex coupled environments, where spatio-temporal invariant patterns are exploited to achieve more accurate predictions Yuan et al. (2023). Shen et al. (2021) provided a comprehensive review of OOD generalisation, covering various aspects such as problem definition, methodological development, evaluation procedures, and future directions, all of which are pertinent to the growing demand for models capable of handling OOD data in real-world applications.

Specifically within graph neural networks (GNNs), Li et al. (2021) introduced an out-of-distribution generalised graph neural network (OOD-GNN) that demonstrates strong performance on unseen graphs with different distributions from the training data. Similarly, Ju et al. (2024) provided a detailed survey of GNN models, highlighting how they tackle practical real-world challenges, including imbalance, noise, privacy, and OOD scenarios—issues often overlooked in existing reviews but crucial for the robust application of GNNs in complex domains.

1.3 Overview of Particle-Based Fluid Simulations:

Particle-based fluid simulations have seen significant advancements, particularly through the application of Graph Neural Networks (GNNs) and other deep learning approaches. Khemani et al. (2024) reviewed several GNN models, such as Graph Convolutional Networks (GCNs), GraphSAGE, and Graph Attention Networks (GATs), which are increasingly applied across various domains. The review particularly highlights the message-passing mechanism used by GNNs to

capture complex interactions between nodes, emphasising both the strengths and limitations of these models in different applications.

Li et al. (2018) introduced Dynamic Particle Interaction Networks (DPI-Nets), designed specifically for learning particle dynamics. DPI-Nets focus on capturing the dynamic, hierarchical, and long-range interactions between particles. This framework can be integrated with perception and gradient-based control algorithms, enabling effective manipulation of deformable objects in robotic applications. This approach represents a key advancement in modelling fluid and particle-based systems, offering greater adaptability in dynamic environments.

Yalan et al. (2020) proposed a Lagrangian fluid simulation model, which uses neural networks to synthesise velocity fields in an irregular Lagrangian data structure. The model employs symmetric functions to capture the spatial structure and particle interactions, while different network architectures are designed to learn hierarchical features of fluid behaviour.

Müller et al. (2007) presented a position-based approach to fluid simulation, which differs from traditional impulse-based methods by directly manipulating particle positions rather than velocities. This method avoids explicit integration issues common in force-based systems and enables efficient handling of collision constraints. By projecting points to valid locations, this approach resolves particle penetration completely, offering a robust method for simulating dynamic systems.

Ladický et al. (2015) framed physics-based fluid simulations as a regression problem, where the acceleration of each particle is estimated for every frame. The feature vector used directly models forces and constraints derived from the Navier-Stokes equations, significantly improving the generalisation capability of the model. This allows for reliable predictions of particle positions and velocities, even over large time steps, enhancing the efficiency of fluid simulations.

In contrast, Ummenhofer et al. (2020) advocated for the use of spatial convolutions as the primary differentiable operation to relate particles to their neighbours, rather than relying on a graph structure. This method enables effective modelling of local interactions in fluid systems. Meanwhile, Belbute-Peres et al. (2020) combined traditional graph convolutional networks with an embedded differentiable fluid dynamics simulator, providing a hybrid approach that leverages both GNNs and physical simulations.

He et al. (2022) further advanced this field by using graph convolutional attention networks to infer velocity flow fields and the forces acting on bodies from incomplete data, demonstrating the versatility of graph-based methods in fluid simulations. Liu et al. (2022) proposed a fluid flow simulator based on a graph attention network architecture, trained on simulation results for flow fields around cylinders at different Reynolds numbers. Their work showcased the ability of GNNs to handle complex flow conditions, particularly in fluid dynamics problems.

1.4 Motivation for Improving Boundary Conditions and Training Methodologies for GNS Models

Sanchez-Gonzalez et al. (2020) introduced a Graph Network-based Simulator (GNS) framework designed for modelling particle-based fluid flow. They provided a TensorFlow version of the code, along with datasets for training and testing. Kumar & Vantassel (2023) later adapted this code to PyTorch, comparing the performance of both frameworks. However, a critical limitation of both versions is the inability to define inlet velocity conditions. This restricts the ability of GNS to model out-of-distribution (OOD) scenarios, thereby reducing its capacity to generalize beyond the examples in its training dataset. The physics or the partial differential equations (PDEs) used to generate the training data has not been explicitly mentioned. However, it was observed that the pressure gradient within the fluid domain in both the training and testing data is negligible, with the fluid pressure being mostly atmospheric. It appears that the model has primarily learned the following incompressible fluid flow equation:

$$\frac{D\mathbf{v}}{Dt} = \nu\nabla^2\mathbf{v} + \mathbf{g} \tag{1}$$

where:

- $\mathbf{v}$ (m/s) – Velocity of fluid particles.
- $\frac{D\mathbf{v}}{Dt}$ (m/s^2) – Material derivative, representing the acceleration following a fluid particle.
- ν (m^2/s) – Kinematic viscosity of fluid.
- $\mathbf{g}$ (m/s^2) – External body force per unit mass, typically gravity.

Klimesch et al. (2022) also highlighted the generalisation limitations of the GNS architecture, reporting a tendency for the model to deform fluid during roll-out. Li & Farimani (2022) addressed this issue by proposing a new GNN architecture that explicitly models fluid field dynamics. Their method decomposes fluid simulation into separate components—advection, collision, and pressure projection.

While the GNS model presented by Sanchez-Gonzalez et al. (2020) emphasised its potential for generalisation through dynamics bootstrapping, some limitations remain. The bootstrapping technique involved randomly selecting two-time sequences from all training scenarios across different time steps. However, Ball et al. (2021) argued for "zero-shot dynamics generalisation," where models should be able to adapt to changing dynamics without being explicitly trained on diverse conditions. Their use of dynamics augmentation in model-based reinforcement learning demonstrates how such methodologies can enable generalisation to unseen dynamics.

Pfaff et al. (2021) proposed enhancing the GNS architecture by integrating adaptive mesh representations. This method involves encoding simulation states into a graph and performing computations in both mesh-space and Euclidean world space. By passing messages in mesh-space, the model can approximate differential operators critical to the internal dynamics of physical systems, while message passing in world space accounts for external dynamics like contact and collision. Revisiting fundamental concepts such as triangulation and three-dimensional unstructured mesh generation Zheng et al. (1996) may offer new insights into the accuracy of point creation techniques and the broader applicability of these methods.

To address these limitations, this paper sets two key research objectives:

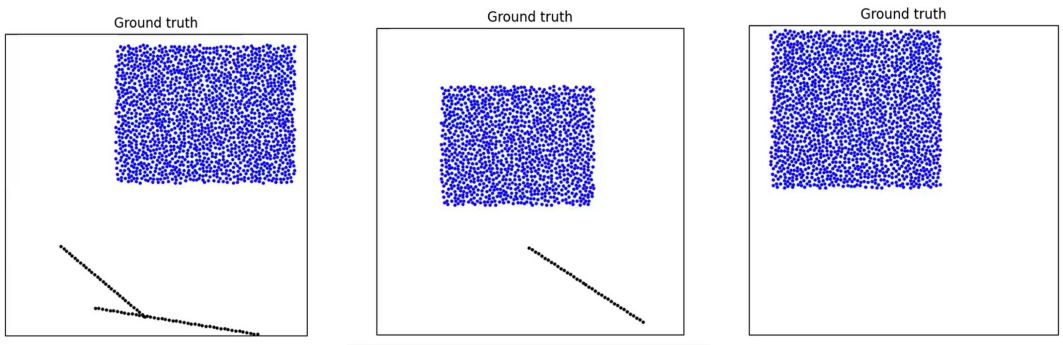
- Enhancing the GNS framework by implementing the PTR algorithm to manage inlet velocity inputs. This would enable the GNS to better handle boundary conditions and improve its generalisation to OOD examples.
- Comparing sequential training methods with traditional bootstrapping to assess their effectiveness in enhancing generalisation. Sequential training might offer a more structured way to expose the model to fluid dynamics over time, as opposed to the randomness inherent in bootstrapping.

The paper is structured to thoroughly examine the Graph Network Simulator (GNS) architecture, focusing on how it learns and processes training data, with particular emphasis on the implementation of the PTR algorithm as a means to introduce an inlet velocity boundary condition. Section 2 provides a description of the 'WaterRamps' dataset utilised to train the Graph Network Simulator (GNS) model, including a detailed analysis of particle characteristics such as mass, velocity, and cross-sectional area. It further explains how the dataset is employed to define both in-distribution and out-of-distribution (OOD) examples. Section 3 outlines the bootstrapping method referenced from Sanchez-Gonzalez et al. (2020) and introduces a sequential training approach to test its effectiveness. This section also includes a comparison of Mean Squared Error (MSE) results between bootstrapping and sequential training. The steps for implementing the divergence of velocity calculations and the 'push particle' algorithm are illustrated in this section. The PTR Algorithm is introduced in Section 4, detailing its role in enabling realistic particle inflows at boundaries. The mathematical formulation of the algorithm is presented, including calculations for binomial distribution and inflow velocity, along with an evaluation of its performance against theoretical values for particle release and area consistency. Section 5 highlights the primary challenges encountered, particularly the difficulty in maintaining incompressibility

during simulations on out-of-distribution (OOD) examples. The paper also compares implementations in PyTorch and TensorFlow in terms of particle density and consistency, identifying key pressure-related issues in the GNS model. The effect of learned wall friction on reattachment length prediction is examined in the flow over a backward-facing step example. Potential areas for future research are also discussed. Finally, the key findings—emphasising the role GNS can play in modelling fluid flow—are summarised in the conclusion section.

2 The 'WaterRamps' Training Dataset

Each fluid particle assumes a given average value of mass. The fluid particle is assumed to occupy a corresponding approximate volume or area (in two dimensions) to yield an average density value. For example, in the case of the 'WaterRamps' training dataset available on GitHub[1], which has been used to train the GNS in this study, the average initial area of each particle used among all 1000 training datasets is 1.0747 cm^2 with a standard deviation of 0.00431. The minimum and maximum values of areas used are 1.0586 cm^2 and 1.0879 cm^2, respectively. Assuming the density of water at 20 °C is 998.2 kg/m^3, the average mass of a water particle used in this study becomes 107.28 g. The 'WaterRamps' training dataset contains 1000 scenarios. Three sample scenarios are shown in Figure 1. The corresponding rollout video files are available as supplementary material.



(a) (b) (c)

Figure 1: Training data set sample scenarios. Number of particles (a) 1612, (b) 1148, (c) 1616; initial velocity (a)1.99 m/s, (b) 2.86 m/s, (c) 2.04 m/s

The initial velocity for all particles in each scenario is the same. The velocity vectors and the corresponding widths and heights of all particles in the 1000 scenarios are shown in Figure 2. The average speed is 1.9 m/s, with minimum and maximum speeds of 0.09 m/s and 5.31 m/s, respectively. The average number of water particles used per scenario is 1463, with minimum and maximum values of 842 and 2227 (Figure 3).

The initial configurations shown in Figures 1 and 2 were studied for 1,000 scenarios. The average area of particles A_p across all training scenarios at the initial time step was calculated to be 0.000107472 m^2. This resulted in the average height and width of a square-shaped imaginary fluid particle being $\sqrt{0.000107472} = 0.010366883$ m. Using this particle area, the total number of particles used across all 1,000 scenarios was recalculated. For instance, in Figure 1a, the original number of particles is 1,612. The total width and height of all particles are measured as 0.47 m and 0.37 m, respectively. The corresponding average number of particles along the width and height are $0.47/0.010366883 = 45.72$ and $0.37/0.010366883 = 35.35$. The predicted total number of particles is $45.72 \times 35.35 = 1,612$, which matches the number of particles used in the training dataset. These calculations were repeated for all 1,000 scenarios, yielding a prediction error of 0.03%. This confirmed that for every incompressible water particle, the average area needs to be 0.000107472 m^2. This value is then used to calculate the average number of particles released

[1] https://github.com/google-deepmind/deepmind-research/tree/master/learning_to_simulate

along a given input boundary length for the implementation of the PTR algorithm presented in the next section, ensuring that particle introduction remained consistent with the physical characteristics of the simulated system.

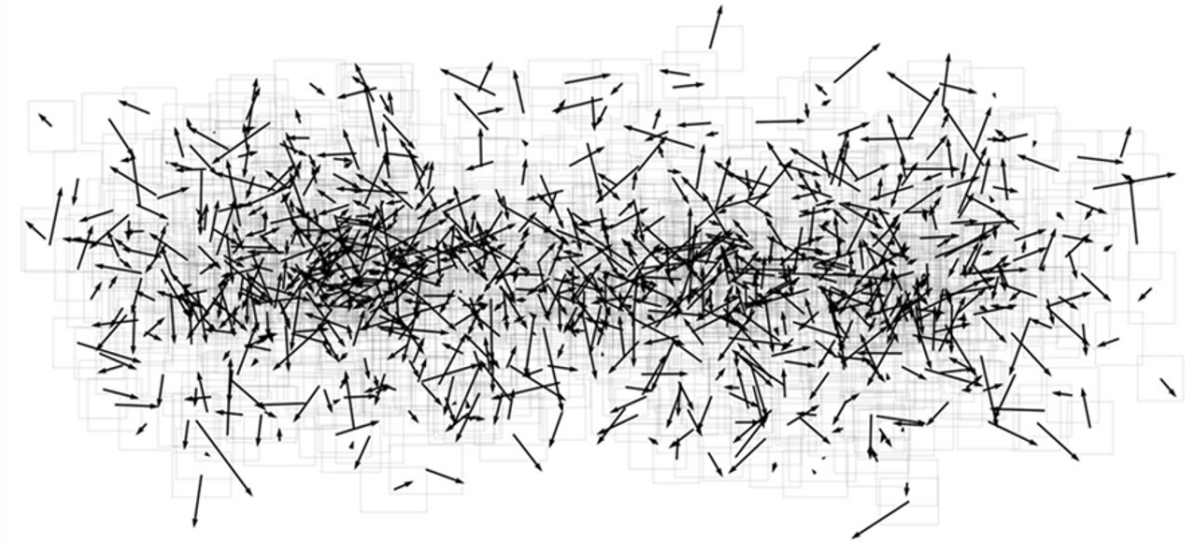


Figure 2: Initial locations of particles in each scenario are shown by a rectangle with corresponding initial velocity vector. The total number of training scenarios is 1000.

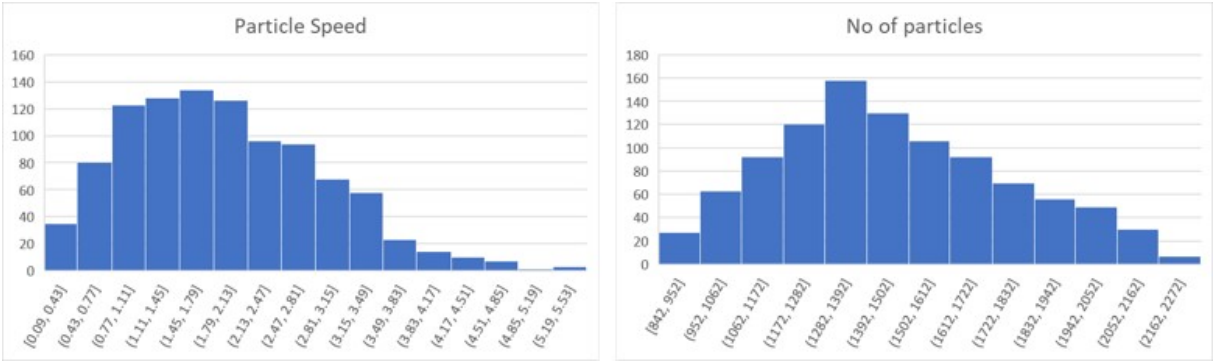


Figure 3: The speed and number of particle distribution for 1000 training scenarios.

This training dataset defines the scope of in-distribution examples and serves as a reference to define the out-of-distribution (OOD) examples presented in Section 5. It is observed that during the rollout, the gauge pressure is almost zero in all training examples, as most of the particles remain part of a free surface due to splashing and continuous mixing of water particles. In other words, it is important to observe that the hydrostatic pressure component on particles is negligible in all training examples.

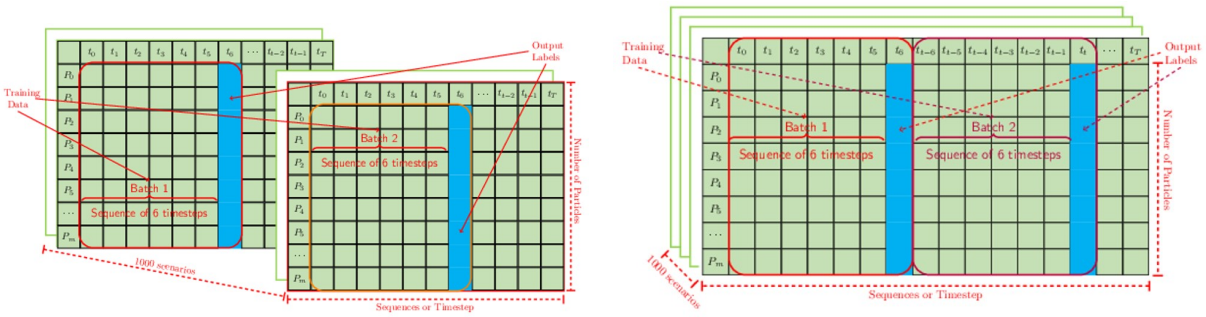
3 Training Methodology

For a given time step t, GNS predicts the next position for a set of particles $\mathcal{P}_t = p_{t1}, p_{t2}, \ldots, p_t$ based on the positions of the five previous time steps and the positions and types of neighbouring particles. It is able to predict the entire rollout of fluid particles from given initial positions of all particles. Initially, at $t = 0$, the history of five previous time steps for each particle must be provided. It is calculated based on the equations of motion for a given initial velocity and acceleration. The use of five previous timesteps helps capture temporal dependencies and inertial effects, but it is only one factor influencing the next particle position. A previous study by Sanchez-Gonzalez et al. (2020) identified five timesteps as an optimal choice for capturing long-range dependencies in unsteady flows. The current approach assumes a constant timestep value.

If the neighbourhood changes drastically over these five timesteps—due to phenomena such as droplet detachment or filamentation—a smaller timestep may be required to accurately resolve such rapid changes. Investigating adaptive timestep strategies to address this issue remains an open area for future research.

Sanchez-Gonzalez et al. (2020) also proposed dynamics bootstrapping for training GNS. They randomly selected two non-overlapping batches with the corresponding history of five previous steps across 1000 scenarios (Figure 5c). These two batches were presented to the GNS training algorithm at each epoch (see Algorithm 1). The algorithm was trained for 20 million epochs. The algorithm has been updated to implement sequential training, where a consecutive sequence of two batches with five previous steps were selected for training at each epoch (Figure 4b). The results are compared to assess whether dynamics bootstrapping offers better generalisation ability.

Each batch of particles is processed through an ENCODER, which generates a graph embedding with node and edge features. This graph is then passed through a PROCESS step consisting of ten message-passing iterations via feedforward neural networks. Each neural network has two hidden layers with 128 nodes per layer, while the input layer size is dynamically determined. The number of networks and their parameters have already been optimised by Sanchez-Gonzalez et al. (2020), and the same configuration was used in this study. Finally, the DECODE step outputs the updated particle positions. The full ENCODE-PROCESS-DECODE pipeline is explained by Sanchez-Gonzalez et al. (2020) in their published paper, and is available in the open-source code. Hence, these details have not been reproduced here. The implementation details of the sequential training approach, which has been explored in this study, are explained in Algorithm 1.



(a) Dynamics bootstrapping: Random selection of batches across scenarios and time steps.

(b) Sequential training: Sequential selection of two consecutive batches

Figure 4: Sequential training: Sequential selection of two consecutive batches.

Algorithm 1: GNS Training: Dynamics Bootstrapping or Sequential

Data :

1. Generate a set of all 1000 scenarios (training examples) with particles $p_{ij}(t)$:
   $\mathcal{U} = \bigcup_{i=1}^{1000} \left(\bigcup_{j=1}^{n_i} \{p_{ij}(t_k) \mid t_k \in \mathcal{T}\} \right)$
2. At initial time step t_0, $\forall p_{i,0} \in \mathcal{U}$, associate a history of 5 previous steps using the equations of motion and v_{in} as assumed for each training example.
   $\{\ p_{ij}(t_0), H_{p_{ij}}(t_0) \mid H_{p_{ij}}(t_0) = \{x_{ij}(t_0 - k\Delta t), y_{ij}(t_0 - k\Delta t) \mid k = 1, \ldots, 6; \}\ \}$
3. For each existing particle in the computational domain, update its history by dropping the oldest step and adding the new one

$$H_{p_{ij}}(t+1) = \{(x_{ij}(t-5), y_{ij}(t-5)), \ldots, x_{ij}(t), y_{ij}(t))\}$$

Result: A trained GNS with a minimised one step MSE over $\mathcal{U}$.

while $\epsilon \leq E$ **do**

`// The epoch` ϵ `is less than the total number of epochs` E

if *Dynamics Bootstrapping* **then**
Randomly select time steps t_1, t_2 from $\mathcal{U}$ for batches $\mathcal{B}_1(t_1)$ and $\mathcal{B}_2(t_2)$ from two scenarios with corresponding history of 5 previous steps.
else if *Sequential Training* **then**
Sequentially select a time step along with corresponding history of 5 previous time steps from $\mathcal{U}$ to create two contiguous batches $\mathcal{B}_1(t_1)$ and $\mathcal{B}_2(t_2 = t_1 + 6)$ for all scenarios.
end
$\mathcal{B}_1(t_1), \mathcal{B}_2(t_2) \leftarrow \mathcal{U}$
$\mathcal{B}_1(t_1+1), \mathcal{B}_2(t_2+1) \leftarrow \text{DECODE}(\text{PROCESS}(\text{ENCODE}(\mathcal{B}_1(t_1), \mathcal{B}_2(t_2))))$

Use the updated positions to determine acceleration and optimise all GNS parameters by minimising the acceleration-based one step loss function defined as:

$$\text{Loss} = \sum_{t=t_1}^{t_2} \sum_{\forall p_i \in \mathcal{B}(t)} \|\hat{a}_i(t) - a_i(t)\|^2 \tag{2}$$

where $\hat{a}_i(t)$ is the true acceleration and $a_i(t)$ is the predicted acceleration for particle p_i at time t
$\epsilon = \epsilon + 1$
end

The PyTorch implementation of the GNS algorithm with dynamics bootstrapping used in this study (Algorithm 1) yielded a mean squared error (MSE) value, averaged across time, particle, and spatial axes of all 100 test cases, of 12.9×10^{-3} after 20 million epochs. For the sequential training implementation, the error value was 13.9×10^{-3}. Sanchez-Gonzalez et al. (2020), using dynamics bootstrapping, reported an MSE value of 11.6×10^{-3}. In other words, the dynamics bootstrapping training algorithm achieved a 7% lower MSE across all 100 test cases compared to the sequential training algorithm. While this represents a notable performance improvement, it falls short of the order-of-magnitude reduction initially desired. To better understand this performance gap, a correlation analysis was conducted on particle acceleration values—represented as vectors— using a cosine similarity measure. This was computed for both sequential and randomly selected batches from the training data, with results summarised in Tables 1, 2, 3, 4. For instance, Tables 1 illustrates three such selections: the first example combines scenario

114 (time step 289) with scenario 264 (time step 296), while the second example pairs scenario 264 (time step 296) with scenario 405 (time step 589). The computed cosine similarity for the x- and y-components of acceleration vectors in the first case were -0.0146 and 0.185, respectively, indicating negligible correlation. In contrast, Tables 2, 3, 4 present results for sequential batch selection, where two consecutive batches with prior history were selected from scenarios 114, 405, and 70. Three timesteps—early, middle, and late in the sequence (20, 320, and 590, respectively)—were examined. While the y-component of acceleration tended to exhibit higher correlation than the x-component, the temporal correlation between consecutive batches was not consistently strong across all cases. This suggests that the 7% improvement in MSE with dynamics bootstrapping arises from its ability to learn from uncorrelated batches, rather than relying on potentially weak temporal correlations present in sequential training. Since each batch is treated as an independent learning instance, the bootstrapping approach enables the model to generalise better across diverse flow conditions. In contrast, sequential training, while still capturing some correlations, appears to be constrained by the inconsistent strength of temporal dependencies across different cases.

Table 1: Cosine similarity comparison of X and Y components across randomly chosen batches in Dynamics Bootstrapping

	Examples		
	Example 1	Example 2	Example 3
Scenario / Step for two batches	114 / 289 & 264 / 296	405 / 589 & 714 / 70	70 / 315 & 979 / 501
Cosine similarity: X component	-0.0146	0.0316	-0.0116
Cosine similarity: Y component	0.185	-0.2483	-0.0278

Table 2: Cosine similarity comparison of X and Y components across randomly chosen batches in Dynamics Bootstrapping

	Examples		
	Example 1	Example 2	Example 3
Scenario / Step for two batches	114 / 20 & 114 / 26	114 / 320 & 114 / 326	114 / 590 & 114 / 596
Cosine similarity: X component	0.0633	0.609	0.93
Cosine similarity: Y component	0.999	0.742	0.903

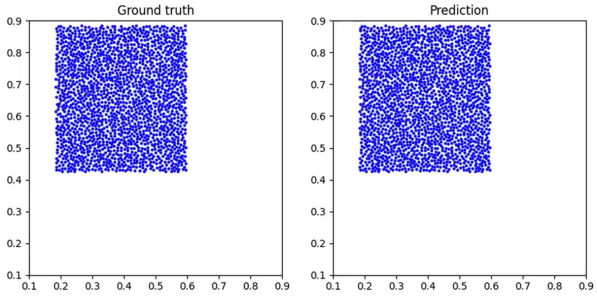
Table 3: Cosine similarity comparison of X and Y components across randomly chosen batches in Dynamics Bootstrapping

	Examples		
	Example 1	Example 2	Example 3
Scenario / Step for two batches	405 / 20 & 405 / 26	405 / 320 & 405 / 326	405 / 590 & 405 / 596
Cosine similarity: X component	-0.1	0.547	0.88
Cosine similarity: Y component	-0.02	0.626	0.362

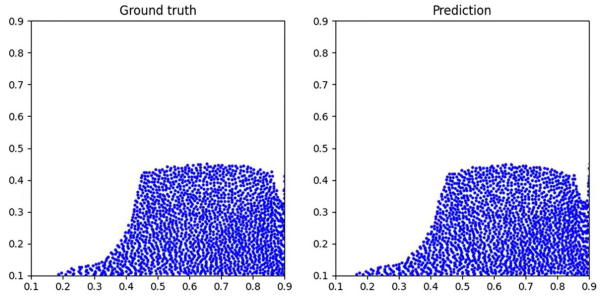
The visual prediction of the ground truth positions and the simulation on one sample test cases is shown in Figure 5, with the full animation video available as supplementary material. The particles were subjected to an initial downward velocity towards the lower right corner of the bounding box for this test example. Note that the total area occupied by the particles is visually similar to the ground truth, which is reflected in the low average MSE value, as discussed above. In later OOD (out-of-distribution) examples, where the ground truth is not available, the GNS fails to maintain the incompressibility condition of water.

Table 4: Cosine similarity comparison of X and Y components across randomly chosen batches in Dynamics Bootstrapping

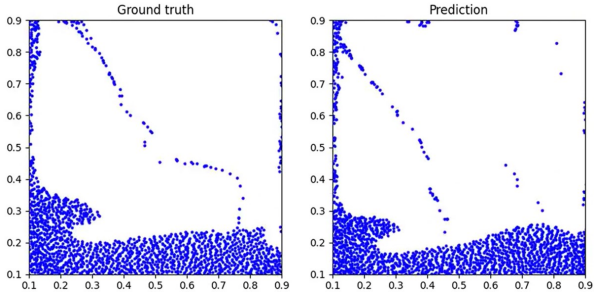
	Examples		
	Example 1	Example 2	Example 3
Scenario / Step for two batches	70 / 20 & 70 / 26	70 / 320 & 70 / 326	70 / 590 & 70 / 596
Cosine similarity: X component	-0.026	0.443	0.135
Cosine similarity: Y component	0.831	0.551	-0.0368



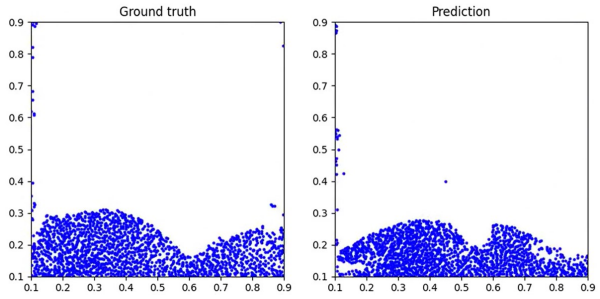
(a) The initial configuration of particles.



(b) Particle positions after first few time steps



(c) Particle positions at a much later time step during the rollout



(d) Particle positions towards the end of the rollout.

Figure 5: Comparison with the ground truth for one of the test examples using the PyTorch implementation of the GNS algorithm with dynamics bootstrapping training.

3.1 Incompressibility assessment with a velocity divergence term and physics based correction with the 'push particle' algorithm

The degree of compressibility is evaluated using a velocity divergence term and a 'push particle' algorithm to improve particle distribution quality.

3.1.1 Velocity divergence calculations

For a given particle i, the divergence of the velocity field is approximated using a discrete form of the continuous divergence operator:

$$\nabla \cdot \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}$$

Since the velocity field is represented by discrete particles, the spatial derivatives are computed using the contributions of neighbouring particles within a control radius r_c.

For each neighbouring pair (i, j), the relative position and velocity differences are defined as:

$$\mathbf{r}_{ij} = (x_j - x_i, y_j - y_i)$$

$$\mathbf{v}_{ij} = (v_{x,j} - v_{x,i}, v_{y,j} - v_{y,i})$$

The squared distance between particles is:

$$r^2 = r_{x,ij}^2 + r_{y,ij}^2$$

The weight function is defined as:

$$\text{weight}(r, r_c) = \begin{cases} 0, & r > r_c \text{ or } r < 10^{-8} \\ \frac{r_c}{r} - 1, & \text{otherwise} \end{cases}$$

For neighbours within the control radius r_c, we compute the weighted velocity divergence contribution:

$$w = \text{weight}(r, r_c)$$

$$\text{partial}_x = r_{x,ij}(v_{x,j} - v_{x,i})$$

$$\text{partial}_y = r_{y,ij}(v_{y,j} - v_{y,i})$$

The local divergence at particle i is given by:

$$\text{div}_i = \frac{w \cdot (\text{partial}_x + \text{partial}_y)}{r^2}$$

Aggregating contributions from all neighbours, the divergence is normalised by the particle density:

$$\text{divergence}_i = \sum_{j \in N(i)} \frac{\text{div}_i \times 2}{\text{density}}$$

The divergence was non-zero (as discussed below) on test examples, and hence a 'push particle' algorithm was implemented to prevent particle overlap. It iteratively adjusts particle positions by detecting neighbours within a predefined radius using a KD-tree (a space-partitioning data structure for organising points in a k-dimensional space) and applying a repulsion force if particles are closer than the minimum allowed distance. This ensures a physically plausible particle distribution and prevents unrealistic clustering.

3.1.2 the 'push particle' algorithm

Let $p_i = (x_i, y_i)$ be the position of the i-th particle and 'minDist' be the minimum distance between two particles. The particles within a radius r of particle i are determined using a K-D tree algorithm:

$$N(i) = \{j \mid \|p_j - p_i\| < r\} \tag{3}$$

The squared distance between particles i and j is:

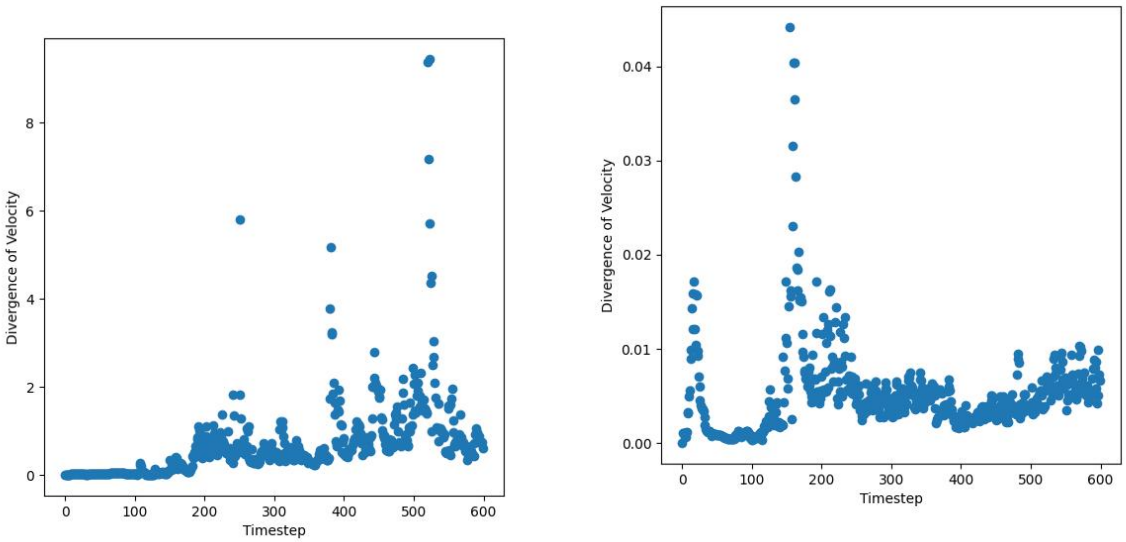
$$d^2 = (x_j - x_i)^2 + (y_j - y_i)^2 \tag{4}$$

The particles are considered too close if $d^2 < \text{minDist}^2$. If the particles ar too close, the new particle positions are calculated using the correction δ as follows:

$$\delta = \frac{\text{minDist} - d}{d} \tag{5}$$

$$x_i \leftarrow x_i - \delta(x_j - x_i) \tag{6}$$
$$y_i \leftarrow y_i - \delta(y_j - y_i) \tag{7}$$
$$x_j \leftarrow x_j + \delta(x_j - x_i) \tag{8}$$
$$y_j \leftarrow y_j + \delta(y_j - y_i) \tag{9}$$

The position history for each adjusted particle is recalculated while preserving the same particle velocity history. This adjustment led to an improvement in maintaining divergence-free flow for test cases (Figure 6).



(a) Original GNS results without push particle algorithm

(b) GNS results are corrected with push particle algorithm resulting in significantly lower maximum divergence of velocity of value.

Figure 6: The maximum divergence of the velocity value for a given timestep for the one of the test example.

4 Particle Trickle Release (PTR) Algorithm

Building upon the concepts presented by Sanchez-Gonzalez et al. (2020) for the 'WaterVortex' example as an out of distribution (OOD) example, we propose a general particle trickle release algorithm to define an inlet boundary condition for fluid flow simulations. As explained in Section 2, with the assumption of density of water as 998.2 kg/m3, the average area of each particle was calculated as 0.000107472 m^2 by studying initial configurations used in training data examples. For a given inlet velocity and inlet boundary length, the mean value for the binomial distribution or the average number of particles added per time step (μ) is calculated as shown in Equation 10. The actual number particles added are chosen from a binomial distribution The standard deviation (σ) is assumed to be 15% of the mean value. This allowed each time step to introduce a statistically fluctuating but controlled number of particles, maintaining realism in the simulation.

$$\mu = \frac{v_{\text{in}} \cdot \Delta t \cdot L}{A_p} \tag{10}$$

where:

- μ represents the mean or average number of particles added per time step.

- v_{in} is the inflow velocity perpendicular to the inlet boundary segment L,
- Δt is the time step size (0.0025),
- L is the length of the inlet boundary segment, and
- A_p is the average area of a particle across all training scenarios at the initial time step ($A_p = 0.000107472$).

In the schematic representation of the PTR algorithm as shown in Figure 7, the values 3, 3, 2 and 4 in the 'Time steps' table represents the number of particles added in the computational domain at time steps t_0, t_1, t_2 and t_3 respectively. The standard GNS implementation assumes that all particles are placed within the computational domain at the initial timestep, whereas the PTR algorithm calculates the number of particles to be added at each timestep a priori. This introduces negligible overhead compared to the computational cost required for each timestep in the GNS implementation.

In the PTR algorithm (Algorithm 2), the time steps set M_{T} representing number of particles added per time step t is generated as follows:

Let m_i be such that m_{it} is the number of particles to be added at time step t, sampled from the following binomial distribution:

$$m_{it} \sim \Big\{ \text{Binomial}(\mu, \sigma) \quad \text{for} \quad t = 1, 2, \ldots, T \quad \text{until} \sum_{i=1}^{t-1} m_i \leq N.$$

If the previous cumulative sum plus the new sampled value exceeds the total number of particles N:

$$m_t = \begin{cases} N - \sum_{i=1}^{t-1} m_i & \text{if } \sum_{i=1}^{t-1} m_i \leq N \\ 0 & \text{if } \sum_{i=1}^{t-1} m_i > N \end{cases}$$

$\forall t \in T$, the resulting set M_{T} is:

$$M_{\text{T}} = \left[m_1, m_2, m_3, \ldots, N - \sum_{i=1}^{t-1} m_{t-1}, 0, 0, 0 \right]$$

Note that each particle added has a history of positions from the previous five time steps, calculated using equations of motion. To ensure that the position at which a particle enters the computational domain fluctuates statistically, the inlet boundary segment L is subdivided with 1,000 equally spaced reference points ($n_r = 1000$) with noise and store in array P_{inlet}. For each of n_r particle reference positions in P_{inlet}, a history of five previous steps is calculated using the equations of motion based on v_{in} :

$$H_{p_i(t=0)} = \{x_i(-k\Delta t), y_i(-k\Delta t) \mid k = 5, \ldots, 1; i = 1, \ldots, n_r\}$$

where $t = 0$ is the initial time step.

Each particle history is schematically shown in the 'Particle history' column in Figure 7. In this way, as explained in Algorithm 2, new particles with the corresponding history are added at each time step, and the history of existing particles is updated based on newly predicted positions. The updating sequence of the union set of particles with history is shown schematically in Figure 7 and the corresponding steps are given in the Algorithm 2.

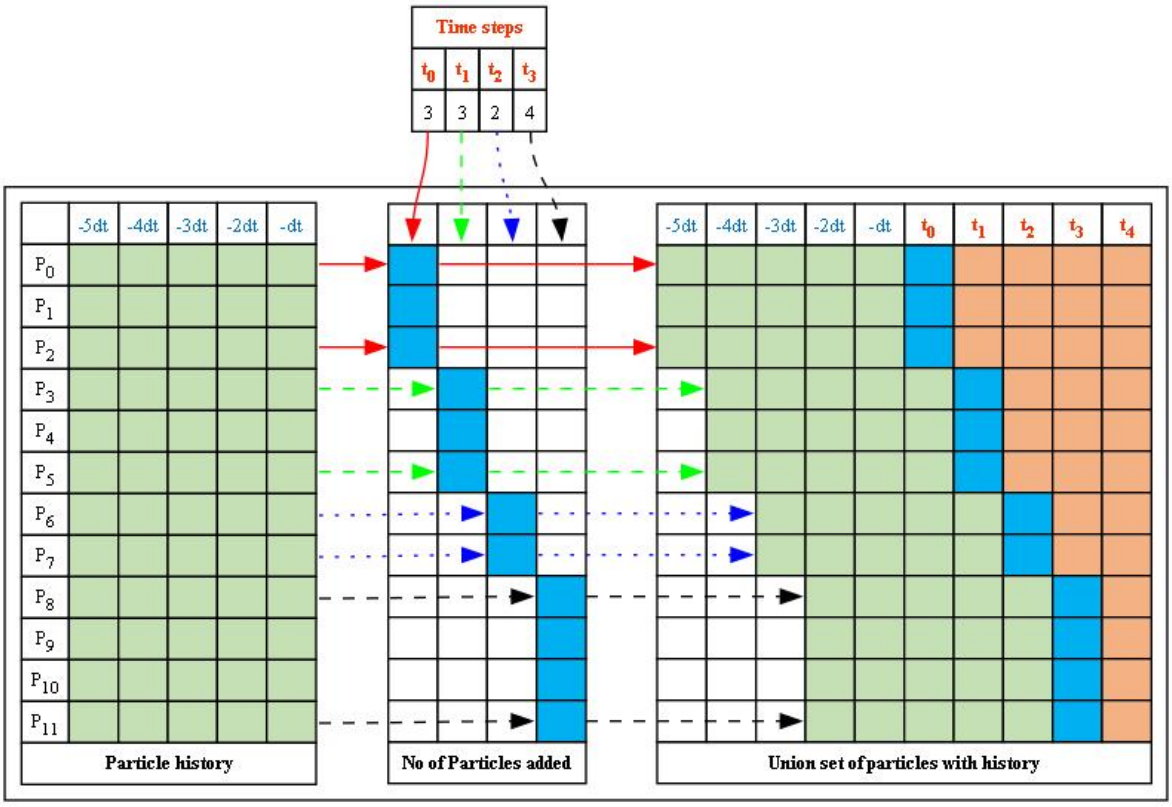


Figure 7: Schematic representation of the particle trickle release (PTR) algorithm.

Algorithm 2: Particle Trickle Release (PTR) Algorithm

Data :

1. Generate set M_{T}
2. Generate particle positions array P_{inlet} with corresponding history $H_{p_i(t=0)}$.

Result: A transient simulation (rollout) of all particles with randomly chosen initial particle positions, with replacement, from P_{inlet} added to the computational domain as per the sequence defined in M_{T}.

$U = \emptyset$ `// Initialise the union set` U `as an empty set.`

$t = 0$

while $t < T$ **do**

`// Time step` t `is less than total number of time steps`

$m_t \leftarrow M_T$

`// get number particles to be added for time step` t

- Divide the inlet boundary segment into m_t bins and map the bins to the 1000 predefined positions P_{inlet}
- Randomly select one position from P_{inlet} for each bin along with the corresponding history $H_{p(t=0)}$
- Generate m_t particle positions with corresponding history $H_{p(t=0)}$ and add the new m_t particles to U:

$$U = \{U \cup \{p_k(t=0), H_{p_k(t=0)} \mid k = 1, \ldots, m_t\}$$

- ENCODE-PROCESS-DECODE Scheme.

$$(p_i(t+1)) \leftarrow \text{DECODER}(\text{PROCESSOR}(\text{ENCODER}(U)))$$

- For each existing particle in the computational domain, update its history by dropping the oldest step and adding the new one

$$H_{p_i(t+1)} = \{(x_i(t-4), y_i(t-4)), \ldots, (x_i(t), y_i(t))\}$$

- Update positions and history $\forall p \in U$

$$U \leftarrow p_i(t+1), H_{p_i(t+1)}$$

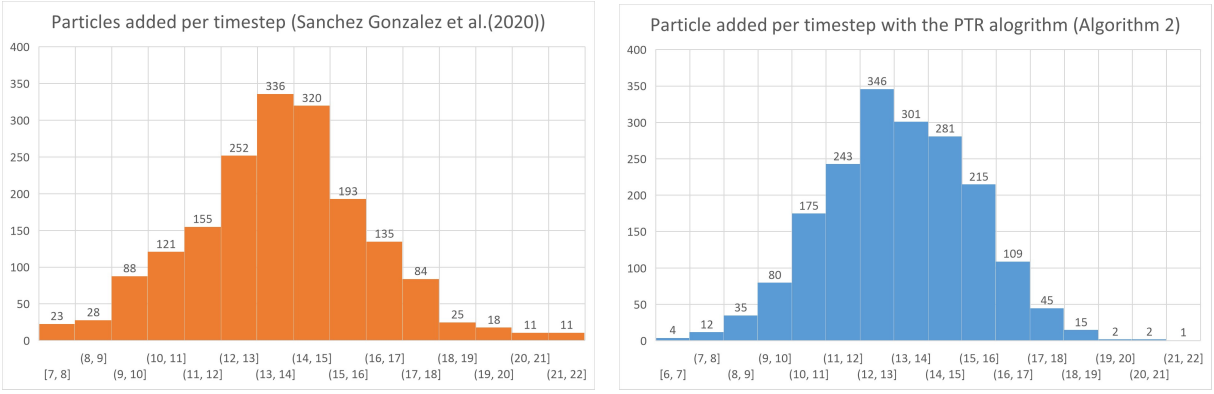
$t = t + 1$

end

5 Challenges & Results

The authors are grateful to Sanchez-Gonzalez et al. (2020) for providing the original array of the number of particles added per time step, which they used for generating results for the 'WaterVortex' example (Figure 8a). The number of particles added per time step are shown on the x-axis and the corresponding frequency of occurrence values are shown on the y-axis. The PTR algorithm utilised the inlet boundary curve's perimeter of 0.3047 m and an input velocity of 2 m/s. Using Equation 10, the target mean and standard deviation values were calculated as 14.17 and 2.13, respectively. However, the PTR algorithm (Algorithm 2) applied these target

values and added 25,498 particles over 1,866 steps, resulting in a mean number of particles added per step of 13.66, with a standard deviation of 2.18.

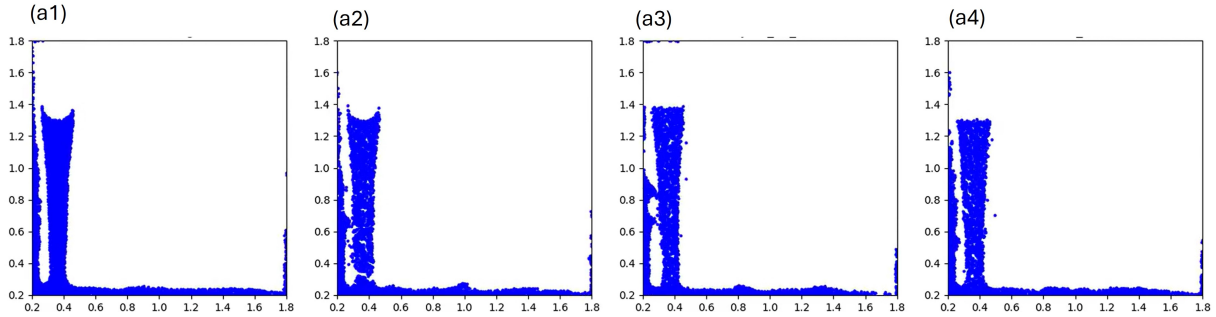


(a) Sanchez-Gonzalez et al. (2020) added 25,484 particles over 1,800 time steps, with the average number of particles added per step being 14.16 and a standard deviation of 2.48

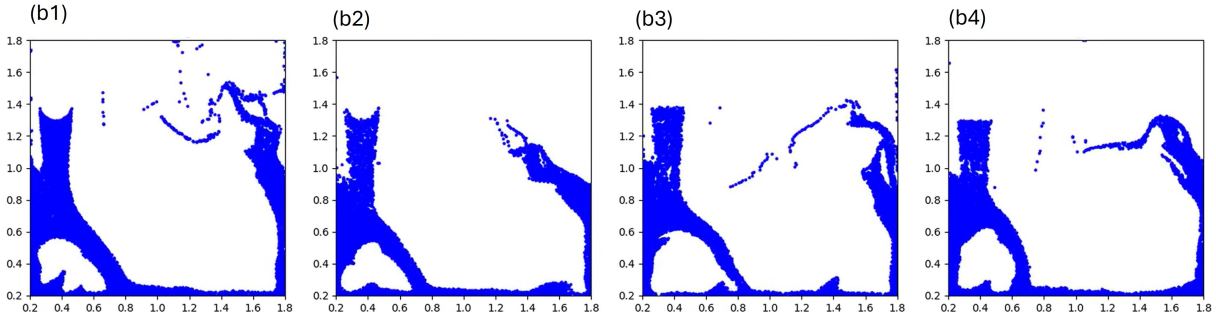
(b) The PTR algorithm added 25,498 particles over 1,866 steps, resulting in a mean number of particles added per step of 13.66 with a standard deviation of 2.18.

Figure 8: The distribution of set M_{T} of new particles m_{t} to be added per time step t .

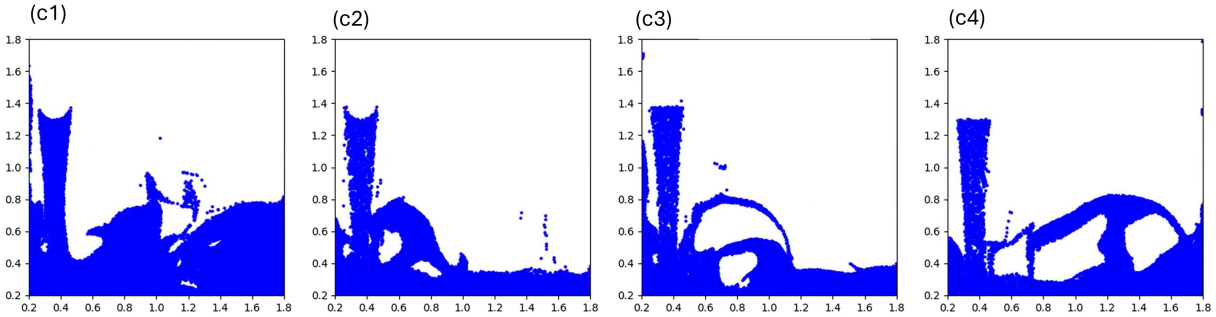
The results of the PyTorch implementation of the PTR algorithm with 20×10^6 training epochs (Figure 9, a2:e2) are compared with the 'WaterVortex' example published by Sanchez-Gonzalez et al. (2020) (Figure 9, a1:e1). The original TensorFlow version (Figure 9, a4:e4) is significantly slower than the PyTorch implementation (Figure 9, a3:e3), and as a result, the TensorFlow simulation was halted after 7.16×10^6 epochs. The results for all four cases are shown in Figure 9. Figures 9a and 9b are similar across all four cases. However, differences in the area occupied by the water particles are evident in Figures 9c and 9d. The TensorFlow version (e4) in Figure 9e performs slightly better than the PyTorch versions (e2 and e3), but the area occupied by particles (Figure 9e e2) is significantly lower than that observed by Sanchez-Gonzalez et al. (2020) (9e e1). The water particles have a noticeably lower density. Sanchez-Gonzalez et al. (2020) highlighted that the message-passing mechanism within the GNS framework mirrors the process of evaluating pressure using the density kernel in Smooth Particle Hydrodynamics (SPH) methods. However, as explained in Section 2, the average area of each particle A_p as computed to be 0.000107472 m². With 25,484 particles, the total area occupied should be approximately 2.739 m^2. Given the container width of 1.6 m, the particle height should be closer to 1.71 m, yet results from the GNS simulations showed heights of around 0.7 m and 0.3 m (Figure 9e). This discrepancy suggests that the message-passing mechanism within the GNS framework struggled to enforce incompressibility, likely due to the absence of pressure-driven constraints. In the majority of training rollouts, the gauge pressure was found to be zero, which indicates that the particles did not learn to manage hydrostatic pressure effectively, resulting in particle compression.



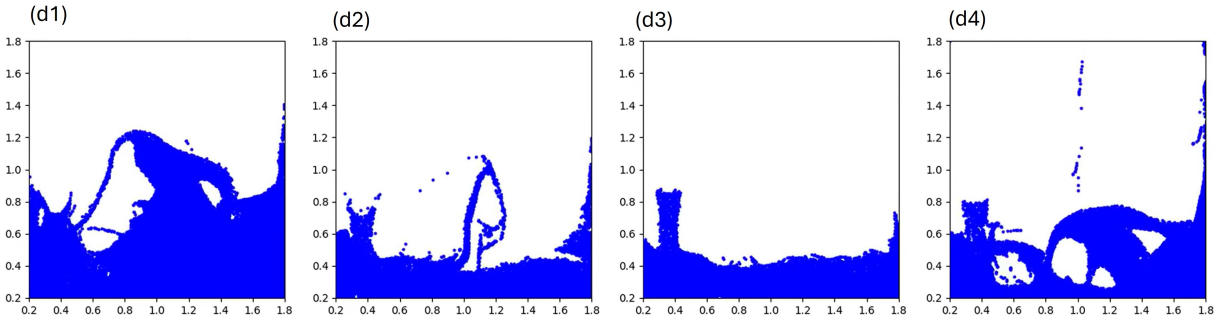
(a) 'WaterVortex' example results at step 249



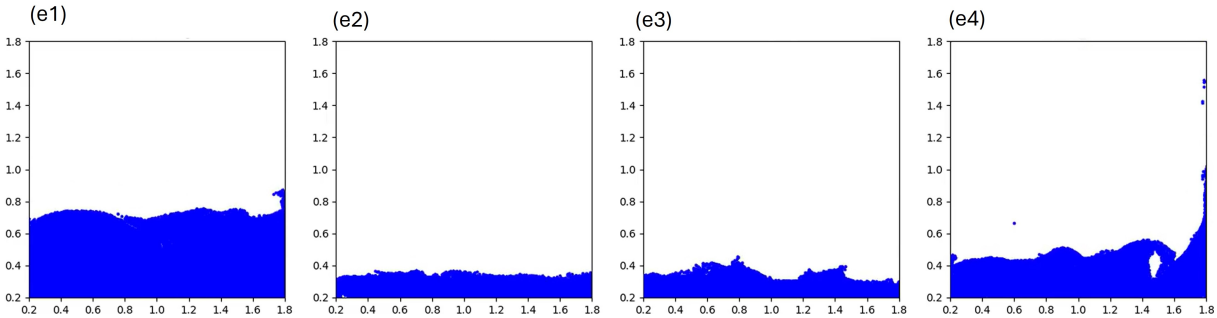
(b) 'WaterVortex' example results at step 522



(c) 'WaterVortex' example results at step 1272



(d) 'WaterVortex' example results at step 1872



(e) 'WaterVortex' example results at step 2424

Figure 9: Comparison of the 'WaterVortex' example - (a1:e1) Sanchez-Gonzalez et al. (2020); (a2:e2) PTR algorithm Pytorch 20×10^6 epochs; (a3:e3) PTR algorithm PyTorch 7.16×10^6 epochs; (a4:e4) PTR algorithm TensorFlow1 7.16×10^6 epochs

The GNS model has been trained on velocity updates and zero-gauge pressure values (Equation 1). As a result, it has not learned the concept of pressure recovery for enforcing incompressibility during training, which may lead to a failure in maintaining divergence-free flow when applied to unseen geometries with nonzero pressure gradients (e.g., the water vortex example).

Li & Farimani (2022) took an alternative approach, separately training the advection, collision, and pressure components of the MPS method to enforce incompressibility. However, it could be suggested that this level of over-supervision may not fully align with the fundamental principles of deep learning. Wessels et al. (2020) proposed a neural particle method for incompressible flow to compute pressure. Toshev et al. (2024) have demonstrated that insights gained from SPH solvers may be used to enhance training and inference of GNNs. For example, as commonly used in Smooth Particle Hydrodynamics (SPH) methods, it may be possible to inform the GNS training algorithm to incorporate pressure calculations through the Equation of State (EOS) (Shadloo et al. 2012, Wang et al. 2016) and correcting particle velocities based on pressure gradients.

5.1 Out-of-Distribution (OOD) Example: Backward-Facing Step Flow

The GNS code was modified to incorporate the boundary wall for the backward step as shown in Figure 10. The input flow height, indicated by the bottom left-right arrow, is 0.2 m, and the step height (h_t) is also 0.2 m. The PTR algorithm released water particles ensuring average constant inlet velocity of 3 m/s. The inlet flow has high Reynolds and Froude numbers, 599,280 and 2.14, respectively, indicating turbulent flow with dominant inertial forces.

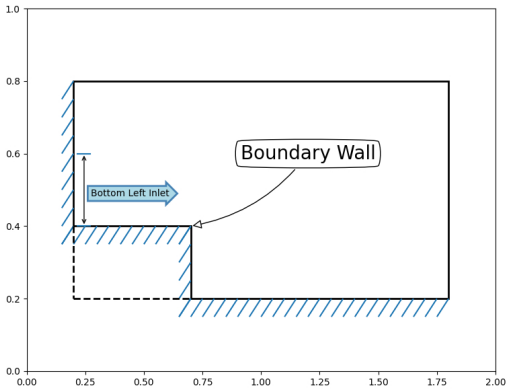


Figure 10: Schematic diagram of the backward-facing step geometry. The input flow height, indicated by the bottom left-right arrow, is 0.2 m, and the step height (h_t) is also 0.2 m.

The Graph Network Simulator (GNS) was tested this out-of-distribution (OOD) example where the flow exhibits upstream separation, an extended recirculation zone, and significant near-wall friction effects—phenomena absent from the training data. This scenario provides insight into the model's ability to generalise beyond the flow behaviours it has learned. However, as seen in Figure 11 for step 141, the model predicts a shorter-than-expected reattachment length (approximately $2.5h_t$ instead of 4–$6h_t$). The particles near the step wall, upstream of the step, were also seen to exhibit greater velocity reduction, suggesting a potential overestimation of near-wall friction effects. This, in turn, may weaken the flow separation strength, leading to an earlier-than-expected reattachment. The results also indicate that the model has not fully captured turbulence structures. However, it is important to note that the training data did not specifically include these physical phenomena. It is also seen that some particles are leaking through the step wall.

Although the model has not perfectly generalised to this unseen scenario, it is notable that GNS fails in a graceful manner—for example, by predicting a shorter-than-expected reattachment length. While the results fall short of expectations, they still represent a reasonable adaptation

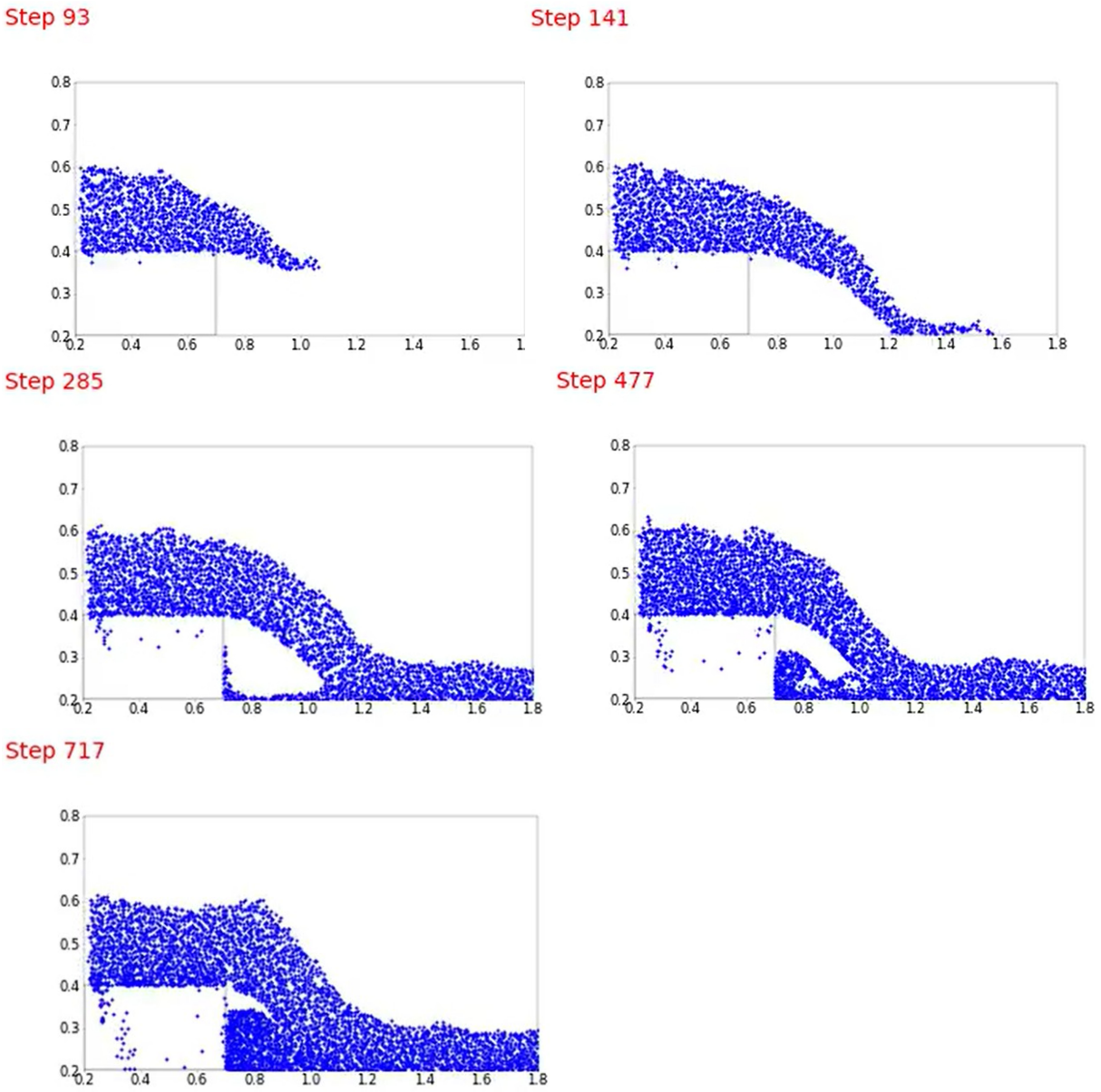


Figure 11: GNS flow over a backward-facing step. Particle positions are shown for steps 93, 141, 285, 477, and 717. Step 141 suggests a reattachment length much shorter than expected. The clustering of particles near the wall and within the recirculation zone results from a significant reduction in velocity due to wall friction effects, which the particles may have learned from training data.

to the new conditions. In contrast, the performance of interpolation-based deep learning models, such as Physics-Informed Neural Networks (PINNs), has not been reported when forced to extrapolate to OOD cases similar to the ones discussed in this paper.

5.2 Future work

While we explored implementing pressure recovery techniques inspired by the Moving Particle Semi-Implicit (MPS) method and the 'push particle' algorithm, these approaches worked on test cases but did not fully ensure divergence-free flow in all scenarios. This suggests that GNS, as currently trained, either lacks the necessary mechanisms for accurate pressure recovery or requires a new training and test dataset that explicitly incorporates pressure recovery. Although hybrid approaches, such as integrating pressure projection methods or modifying message-passing mechanisms, could potentially address this limitation, their development and validation require a

dedicated investigation. Furthermore, refining the error function to explicitly enforce divergence-free flow would be essential to fully resolving this issue. While these directions hold significant potential for future research, they extend beyond the scope of the present study.

At this stage, it is not claimed that the current GNS implementation is inherently faster or more accurate than traditional CFD methods. Instead, the objective is to investigate how GNS can learn physical interactions from data. This work represents an initial step in laying the foundation for future advancements in physics-informed deep learning. The GNS model updates particle states based on neighbour interactions, similar to traditional particle-based methods such as Smoothed Particle Hydrodynamics (SPH).

Based on the research presented in this paper, we view the GNS approach not as a replacement for, nor a complement to, SPH methods, which remain the more appropriate methodology for continuum-scale particle-based fluid mechanics. SPH methods are mature, physically grounded, and benefit from parallel computing and GPU acceleration. Although early SPH implementations were based on weakly compressible formulations that introduced pressure fluctuations (Lind et al. 2020), the method has since evolved into a highly robust framework. In contrast, GNS is still at a relatively early stage and does not yet demonstrate comparable accuracy, particularly in the modelling of pressure fields. Nevertheless, we regard GNS as a promising data-driven methodology with potential to complement molecular dynamics approaches—especially in multiscale simulation contexts where traditional molecular methods are constrained by extremely small time steps. As a natural extension of the current study, we intend to investigate how GNS represents fluid–wall interactions by comparing continuum boundary models with particle-based wall representations, as employed in molecular simulations. Such an analysis may help elucidate what GNS learns from training data and how it handles near-wall behaviour, thereby offering insights into its applicability in bridging time- and length-scale gaps in multiscale physics.

6 Conclusion

This study presents significant advancements in graph network-based fluid flow modelling through the implementation of the PTR algorithm and the introduction of an inlet velocity boundary condition within the Graph Network Simulator (GNS). The findings demonstrate that the PTR algorithm effectively addresses the limitations of previous models by enabling realistic particle inflows at boundaries, thereby enhancing the accuracy of fluid flow simulations.

The 'WaterRamps' Training Dataset has been shown to provide a robust understanding of particle dynamics by offering comprehensive insights into particle characteristics, such as mass, velocity, and cross-sectional area. Notably, its ability to define both in-distribution and out-of-distribution (OOD) examples contributes to the model's generalisation capabilities.

A comparison of bootstrapping and sequential training methodologies revealed that while both approaches yield valuable results, bootstrapping offers a slight advantage in terms of Mean Squared Error (MSE). This insight is particularly relevant for future developments in GNS models, guiding researchers in selecting optimal training strategies.

The divergence of velocity is calculated at the particle level, with the maximum divergence for a given timestep used to assess particle compressibility. A 'push particle' algorithm has been implemented to adjust particle distribution, leading to a significant reduction in the maximum divergence value across test cases.

Despite these advancements, challenges remain, particularly in maintaining incompressibility, boundary layer development, and turbulence modelling during out-of-distribution simulations, such as the 'WaterVortex' and flow over a 'backward-facing step' example. While the model has not fully generalised to these unseen scenarios, it is noteworthy that GNS fails in a controlled manner, often producing reasonable, albeit imperfect, adaptations—such as a shorter-than-expected reattachment length. In contrast, the performance of interpolation-based deep learning models, such as Physics-Informed Neural Networks (PINNs), under similar OOD conditions remains unreported.

Future work should focus on enhancing the robustness of GNS models to address a broader

range of fluid dynamics scenarios. This includes exploring how GNS can be integrated with molecular dynamics and other established simulation techniques. Continued refinement of GNS's capabilities could enable its application in advancing the simulation of complex fluid systems across various scales.

Acknowledgements

The authors gratefully acknowledge the support of the Supercomputing Wales project, which is part-funded by the European Regional Development Fund (ERDF) through the Welsh Government. They also extend their appreciation to the anonymous referees for their insightful comments.

References

Ball, P. J., Lu, C., Parker-Holder, J. & Roberts, S. J. (2021), 'Augmented world models facilitate zero-shot dynamics generalization', *Proceedings of the 38th International Conference on Machine Learning* .
URL: *10.48550/arXiv.2104.05632*

Battaglia, P., Chandler Hamrick, J. B., Bapst, V., Sanchez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K., Nash, C., Langston, V. J., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y. & Pascanu, R. (2018), 'Relational inductive biases, deep learning, and graph networks', *arXiv* .
URL: *https://doi.org/10.48550/arXiv.1806.01261*

Belbute-Peres, F. D. A., Economon, T. & Kolter, Z. (2020), 'Combining differentiable PDE solvers and graph neural networks for fluid flow prediction', *Proceedings of the 37th International Conference on Machine Learning* **119**, 2402–2411.
URL: *10.48550/arXiv.2007.04439*

Brunton, S. L. (2021), 'Applying machine learning to study fluid mechanics', *Acta Mechanica Sinica* **37**, 1718–1726.

Brunton, S. L., Noack, B. R. & Koumoutsakos, P. (2020), 'Machine learning for fluid mechanics', *The Annual Review of Fluid Mechanics* **52**, 477–508.

Edalatifar, M., Shafi, J., Khalid, M., Baro, M., Sheremet, M. A. & Ghalambaz, M. (2024), 'An artificial intelligence approach for the estimation of conduction heat transfer using deep neural networks', *International Journal of Numerical Methods for Heat & Fluid Flow* **34**(8), 3107–3130.

Ghalambaz, M., Sheremet, M., Khan, M., Raizah, Z. & Shafi, J. (2024), 'Physics-informed neural networks (PINNs): application categories, trends and impact', *International Journal of Numerical Methods for Heat & Fluid Flow* **34**(8), 3131–3165.

Goyal, A. & Bengio, Y. (2022), 'Inductive biases for deep learning of higher-level cognition', *Proceedings of the Royal Society A* **478**, 20210068.

Hachem, E., Vishwasrao, A., Renault, M., Viquerat, J. & Meliga, P. (2024), 'Reinforcement learning for cooling rate control during quenching', *International Journal of Numerical Methods for Heat & Fluid Flow* **34**(8), 3223–3252.

He, X., Wang, Y. & Li, J. (2022), 'Flow completion network: Inferring the fluid dynamics from incomplete flow information using graph neural networks', *Physics of Fluids* **34**(8).

Ishize, T., Omichi, H. & Fukagata, K. (2024), 'Flow control by a hybrid use of machine learning and control theory', *International Journal of Numerical Methods for Heat & Fluid Flow* **34**(8), 3253–3277.

Ju, W., Yi, S., Wang, Y., Xiao, Z., Mao, Z., Li, H., Gu, Y., Qin, Y., Yin, N., Wang, S., Liu, X., Luo, X., Yu, P. S. & Zhang, M. (2024), 'A survey of graph neural networks in real world: Imbalance, noise, privacy and ood challenges', *ArXiv* .
URL: *https://doi.org/10.48550/arXiv.2403.04468*

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S. & Yang, L. (2021), 'Physics-informed machine learning', *Nature Reviews Physics* **3**, 422–440.

Khan, U., Pao, W., Pilario, K., Sallih, N. & Khan, M. (2024), 'Two-phase flow regime identification using multi-method feature extraction and explainable kernel fisher discriminant analysis', *International Journal of Numerical Methods for Heat & Fluid Flow* **34**(8), 2836–2864.

Khemani, B., Patil, S., Kotecha, K. & Tanwar, S. (2024), 'A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions', *Journal of Big Data* **11**.
URL: *10.1186/s40537-023-00876-4*

Klimesch, J., Holl, P. & Thuerey, N. (2022), 'Simulating liquids with graph networks', *ArXiv* **abs/2203.07895**.
URL: *10.48550/arXiv.2203.07895*

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

Kumar, K. & Vantassel, J. (2023), 'GNS: A generalizable graph neural network-based simulator for particulate and fluid modeling', *The Journal of Open Source Software* **8**(88), 5025.

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

Li, H., Wang, X., Zhang, Z. & Zhu, W. (2021), 'Ood-gnn: Out-of-distribution generalized graph neural network', *ArXiv* .
URL: *https://doi.org/10.48550/arXiv.2112.03806*

Li, Y., Wu, J., Tedrake, R., Tenenbaum, J. B. & Torralba, A. (2018), 'Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids', *CoRR* **abs/1810.01566**.
URL: *http://arxiv.org/abs/1810.01566*

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

Lind, S. J., Rogers, B. D. & Stansby, P. K. (2020), 'Review of smoothed particle hydrodynamics: towards converged lagrangian flow modelling', *Proceedings of the Royal Society A* **476**, 20190801.

Lino, M., Fotiadis, S., Bharath, A. & Cantwell, C. (2023), 'Current and emerging deep-learning methods for the simulation of fluid dynamics', *Proceedings of the Royal Society A* **479**, 20230058.

Liu, Q., Zhu, W., Jia, X., Ma, F. & Gao, Y. (2022), 'Fluid simulation system based on graph neural network', *ArXiv* .
URL: *10.48550/arXiv.2202.12619*

Löhner, R., Antil, H., Tamaddon-Jahromi, H., Chakshu, N. K. & Nithiarasu, P. (2021), 'Deep learning or interpolation for inverse modelling of heat and fluid flow problems?', *International Journal of Numerical Methods for Heat & Fluid Flow* **31**, 3036–3046.

McConkey, R., Kalia, N., Yee, E. & Lien, F.-S. (2024), 'Turbo-RANS: straightforward and efficient Bayesian optimization of turbulence model coefficients', *International Journal of Numerical Methods for Heat & Fluid Flow* **34**(8), 2986–3016.

Müller, M., Heidelberger, B., Hennix, M. & Ratcliff, J. (2007), 'Position based dynamics', *Journal of Visual Communication and Image Representation* **18**, 109–118.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A. & Battaglia, P. W. (2021), 'Learning mesh-based simulation with graph networks', *International Conference on Learning Representations* .

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J. & Battaglia, P. W. (2020), 'Learning to simulate complex physics with graph networks', *Proceedings of the 37th International Conference on Machine Learning* .

Shadloo, M. S., Zainali1, A., Yildiz, M. & Suleman, A. (2012), 'A robust weakly compressible SPH method and its comparison with an incompressible SPH', *International Journal of Numerical Methods in Engineering* **89**, 939–956.

Shen, Z., Liu, J., He, Y., Zhang, X., Xu, R., Yu, H. & Cui, P. (2021), 'Towards out-of-distribution generalization: A survey', *ArXiv* **abs/2108.13624**.

Shi, K., Jin, G., Yan, W. & Xing, H. (2024), 'Permeability estimation for deformable porous media with convolutional neural network', *International Journal of Numerical Methods for Heat & Fluid Flow* **34**(8), 2943–2962.

Shukla, K., Xu, M., Trask, N. & Karniadakis, G. (2022), 'Scalable algorithms for physics-informed neural and graph networks', *Data-Centric Engineering* **3**.

Suresh, P. & Chakravarthy, B. (2024), 'Deep learning algorithms for temperature prediction in two-phase immersion-cooled data centres', *International Journal of Numerical Methods for Heat & Fluid Flow* **34**(8), 2917–2942.

Toshev, A. P., Erbesdobler, J. A., Adams, N. A. & Brandstetter, J. (2024), 'Neural SPH: Improved neural modeling of lagrangian fluid dynamics', *41st International Conference on Machine Learning - ML Research Press* **235**, 48428 – 48452.
URL: *10.48550/arXiv.2402.06275*

Ummenhofer, B., Prantl, L., Thürey, N. & Koltun, V. (2020), Lagrangian fluid simulation with continuous convolutions, *in* 'International Conference on Learning Representations'.
URL: *https://api.semanticscholar.org/CorpusID:211165482*

Wang, Z., Chen, R., Wang, H., Liao, Q., Zhu, X. & Li, S.-Z. (2016), 'An overview of smoothed particle hydrodynamics for simulating multiphase flow', *Applied Mathematical Modelling* **40**, 9625–9655.

Wessels, H., Weißenfels, C. & Wriggers, P. (2020), 'The neural particle method – an updated lagrangian physics informed neural network for computational fluid dynamics', *Computer Methods in Applied Mechanics and Engineering* **368**, 113127.

Yalan, Z., Ban, X., Du, F. & Wu, D. (2020), 'FluidsNet: End-to-end learning for lagrangian fluid simulation', *Expert Systems with Applications* **152**, 113410.

Yuan, H., Sun, Q., Fu, X., Zhang, Z., Ji, C., Peng, H. & Li, J. (2023), 'Environment-aware dynamic graph learning for Out-of-Distribution generalization', *Proceedings of the 37th International Conference on Neural Information Processing Systems* .
URL: *10.48550/arXiv.2311.11114*

Yuan, L., Park, H. S. & Lejeune, E. (2022), 'Towards out of distribution generalization for problems in mechanics', *Computer Methods in Applied Mechanics and Engineering* **400**, 115569.

Zeng, G.-Z., Chen, Z.-W., Ni, Y.-Q. & Rui, E.-Z. (2024), 'Investigating embedded data distribution strategy on reconstruction accuracy of flow field around the crosswind-affected train based on physics-informed neural networks', *International Journal of Numerical Methods for Heat & Fluid Flow* **34**(8), 2963–2985.

Zheng, Y., Lewis, R. W. & Gethin, D. T. (1996), 'Three-dimensional unstructured mesh generation: Part 1. Fundamental aspects of triangulation and point creation', *Computer Methods in Applied Mechanics and Engineering* **134**(3), 249–268.
URL: *https://www.sciencedirect.com/science/article/pii/0045782595009167*

Zhou, Z., Zhao, F. & Hung, D. (2024), 'Swirl-induced motion prediction with physics-guided machine learning utilizing spatiotemporal flow field structure', *International Journal of Numerical Methods for Heat & Fluid Flow* **34**(8), 2890–2916.