Proceedings of the Thirty-sixth (2026) International Ocean and Polar Engineering Conference **www.isope.org**
Orlando, Florida, USA, May 31-June 5, 2026

ISBN 978-1-880653-72-2; ISSN 1098-6189

Memory-Efficient GNN Training to solve PPE in 3D MPS simulations

*Congyi Huang[1], Decheng Wan[2], Zhe Sun[1], Guiyong Zhang[1]**
[1]School of Naval Architecture, Dalian University of Technology
Dalian, Liaoning, China
[2]Computational Marine Hydrodynamics Lab (CMHL), School of Naval Architecture, Ocean and Civil Engineering,
Shanghai Jiao Tong University, Shanghai, China
*Corresponding Author

ABSTRACT

This paper investigates edge sampling strategies for memory-efficient training of Graph Neural Networks (GNNs) to solve the Pressure Poisson Equation (PPE) in three-dimensional Moving Particle Semi-implicit (MPS) simulations. We compare three approaches: random edge sampling preserving full influence radius topology, and two reduced influence radius methods with different kernel configurations. Through experiments on a 3D liquid sloshing benchmark, we find that random edge sampling at 15% retention achieves $R^2 > 0.96$, while radius truncation methods fail to converge regardless of kernel choice. We provide a physical interpretation linking this observation to the elliptic nature of PPE, which requires global information propagation. Based on these findings, we propose the sparse but complete principle for GNN-based PDE solvers.

KEY WORDS: Graph Neural Network; Pressure Poisson Equation; MPS method; edge sampling; liquid sloshing.

INTRODUCTION

The numerical simulation of fluid dynamics plays a crucial role in various engineering applications. Among the numerous computational methods developed for fluid simulation, particle-based approaches such as the Moving Particle Semi-implicit (MPS) method and Smoothed Particle Hydrodynamics (SPH) have gained significant attention due to their inherent advantages in handling complex free-surface flows, large deformations, and fluid-structure interactions (Koshizuka and Oka 1996; Gingold and Monaghan 1977). Unlike traditional mesh-based methods, these Lagrangian approaches represent the fluid domain as a collection of discrete particles, naturally accommodating topological changes without the need for mesh regeneration.

Despite their flexibility, particle-based methods face substantial computational challenges, particularly in the pressure calculation step. In semi-implicit formulations, the pressure field is obtained by solving a Pressure Poisson Equation (PPE) at each time step to enforce the incompressibility constraint (Khayyer and Gotoh 2009). This process typically involves assembling a large sparse linear system and solving it iteratively using methods such as the Preconditioned Conjugate Gradient (PCG) or Incomplete Cholesky Conjugate Gradient (ICCG) algorithms (Chen, Xi, and Sun 2014). For large-scale three-dimensional simulations involving hundreds of thousands of particles, the PPE solving process can consume over 90% of the total computational time, severely limiting the practical applicability of these methods in time-critical scenarios.

The emergence of deep learning has opened new avenues for accelerating scientific computing, with neural networks demonstrating remarkable capabilities in approximating complex nonlinear mappings (LeCun, Bengio, and Hinton 2015). Physics-Informed Neural Networks (PINNs), introduced by Raissi et al. (2019), have shown promise in solving partial differential equations by embedding physical constraints directly into the loss function. However, PINNs are typically designed for continuous domains and may not naturally accommodate the discrete, irregular structure inherent in particle-based methods. Graph Neural Networks (GNNs), on the other hand, provide a natural framework for learning on non-Euclidean data structures, making them particularly well-suited for particle-based simulations where interactions occur between neighboring particles within a specified influence radius (Kipf 2016; Battaglia et al. 2018).

Recent advances in GNN-based physics simulation have demonstrated impressive results in learning complex dynamics directly from data. Sanchez-Gonzalez et al. (2020) proposed Graph Network-based Simulators (GNS) that successfully learned to simulate various materials including fluids, sand, and deformable solids, showing generalization capabilities to systems with orders of magnitude more particles than those seen during training. Pfaff et al. (2020) extended this approach to mesh-based simulations with MeshGraphNets, demonstrating the versatility of graph-based learning for physical systems. Li and Farimani (2022) applied GNNs to accelerate Lagrangian fluid simulation, achieving significant speedups over traditional SPH solvers while maintaining reasonable accuracy. More recently, Toshev et al. (2024) introduced Neural SPH, combining learned simulators with classical SPH relaxation steps to improve stability and physical consistency.

While these developments are encouraging, a critical examination of the literature reveals that the vast majority of GNN-based fluid simulation studies focus on two-dimensional problems (Li and Farimani 2022; Lino et al. 2023; Bentivoglio et al. 2023). The extension to three dimensions presents fundamental challenges that have not been adequately addressed. In 2D simulations, each particle typically interacts with approximately 20-30 neighbors within its influence radius. However, in

3D, this number increases dramatically to 200-300 neighbors due to the cubic scaling of the interaction volume. For a simulation with N particles, the total number of edges in the graph scales as O(N × k), where k is the average number of neighbors. Consequently, a 3D simulation with 50,000 particles can easily generate over 10 million edges, far exceeding the memory capacity of typical GPU devices used for deep learning.

This memory bottleneck represents a significant barrier to the practical application of GNN-based methods for 3D particle simulations. Yao et al. (2023) proposed the NN-MPS method using fully connected neural networks (FCNNs) to predict pressure, achieving promising results but without exploiting the graph structure of particle interactions. However, FCNNs treat all particles uniformly without explicit spatial relationships, and for large-scale 3D problems with tens of thousands of particles, this approach may lose the local interaction physics essential for PPE solving. In contrast, GNNs naturally exploit the sparse, local connectivity inherent in particle methods.

In this paper, we address this challenge by systematically investigating edge sampling strategies for memory-efficient GNN training on 3D PPE problems. We compare three distinct approaches: (1) random edge sampling that preserves the full influence radius topology but retains only a fraction of edges, (2) reduced influence radius with consistent kernel functions, and (3) reduced influence radius with the original kernel functions.

The remainder of this paper is organized as follows. Section 2 presents the methodology, including the MPS formulation, GNN architecture, and the three edge sampling strategies under investigation. Section 3 describes the experimental setup, including the test case, data preparation, and evaluation metrics. Section 4 presents the results and provides a detailed analysis of the observed phenomena. Finally, Section 5 concludes the paper.

METHODOLOGY

Moving Particle Semi-implicit Method and Pressure Poisson Equation

The MPS method, originally proposed by Koshizuka and Oka (1996), solves the incompressible Navier-Stokes equations in a Lagrangian framework. The governing equations consist of the continuity equation and momentum equation:

$$\frac{D\rho}{Dt} + \rho\nabla \cdot \mathbf{v} = 0$$
$$\frac{Dv}{Dt} = -\frac{1}{\rho}\nabla \boldsymbol{P} + \nu\nabla^2\mathbf{v} + g$$

where ρ is the fluid density, $\mathbf{v}$ is the velocity vector, $\boldsymbol{P}$ is the pressure, ν is the kinematic viscosity, and $\mathbf{g}$ is the gravitational acceleration. The continuity equation is transformed into the incompressibility condition $\nabla \cdot \mathrm{v} = 0$ for constant density flows.

The MPS method employs a fractional step approach, splitting each time step into prediction and correction stages. In the prediction step, an intermediate velocity v* is computed explicitly from viscous and gravitational forces. In the correction step, the pressure field is determined by solving the PPE to satisfy the incompressibility constraint:

$$\nabla^2 P^{n+1} = \frac{\rho}{\Delta t}\nabla \cdot v^*$$

The spatial derivatives in MPS are approximated using particle interaction models based on a kernel function. For a scalar field ϕ, the Laplacian operator is discretized as:

$$\langle\nabla^2\phi\rangle_i = \frac{2d}{n_0\lambda_0}\sum_{j\neq i}(\phi_j - \phi_i)w(|r_j - r_i|)$$

where d is the number of spatial dimensions, n_0 is the initial particle number density, λ_0 is a correction factor, and w(r) is the kernel function. The summation extends over all neighboring particles j within the influence radius r_e. In this work, we employ the standard MPS kernel function:

$$w(r) = \frac{20}{17(r/r_e) + 3} - 1$$

The kernel function vanishes at the influence radius boundary, ensuring compact support for particle interactions. The discretized PPE forms a sparse linear system A**x** = **b**, where **x** contains the unknown pressures at each particle, **b** is the source term related to velocity divergence and density deviation, and A is a sparse coefficient matrix with structure determined by the particle neighborhood topology.

The coefficient matrix A is stored in Compressed Sparse Row (CSR) format, consisting of three arrays: row offsets indicating the starting position of each row's data, column indices identifying the non-zero element positions, and values containing the actual coefficients computed from the kernel function. For a typical 3D simulation with Laplacian influence radius $r_{e\_lap} = 4.01 \times dp$ (where dp is the particle spacing), each particle has approximately 200-250 neighbors, resulting in a highly sparse but large matrix that requires iterative methods for efficient solution.

Boundary Treatment in PPE: In MPS simulations, boundary conditions are essential for the proper formulation of the PPE. We employ three types of particles with different roles in the PPE system:

(1) Interior fluid particles: These particles are fully surrounded by neighboring fluid particles and participate directly in the PPE linear system. Their pressures are computed by solving the PPE.
(2) Free-surface particles: Particles detected at the free surface are assigned a Dirichlet boundary condition with pressure P = 0. These particles are included in the PPE system, but their pressure values are directly set rather than solved.
(3) Wall boundary particles: We employ a two-type wall boundary model. The first type of boundary particles directly contacts fluid particles and participates in the PPE system; their pressures are computed through the PPE solution. The second type also known as buffer particles, is positioned outside the type to complete the support domain of the first type particles. The pressures of the second type particles are interpolated from neighboring first type particles and do not participate in the PPE computation.

Consequently, the PPE linear system includes all interior fluid particles, free-surface particles, and the first type wall boundary particles.

Graph Neural Network Architecture

The goal of our GNN is to learn a direct mapping from the PPE system configuration to its solution:

$$f_\theta: (X, B, A) \rightarrow P$$

where X represents particle positions, B is the RHS vector of the PPE, A encodes the sparse matrix structure, P is the pressure solution, and θ denotes the learnable network parameters. This supervised learning approach treats the GNN as a fast surrogate for the iterative PCG solver.

The natural correspondence between particle interactions in MPS and message passing in GNNs motivates our approach. We construct a graph G = (V, E) where each node v ∈ V represents a particle, and each edge e ∈ E represents a pairwise interaction between neighboring particles within the influence radius.

Node Features: For each particle *i*, we define a feature vector $\mathbf{x}_i$ consisting of:

Normalized position coordinates
Normalized right-hand side term B_i from the PPE
Normalized neighbor count

The total node feature dimension is 5. Position coordinates are normalized to [0, 1] based on the domain extent, while the right-hand side term and neighbor count are standardized using z-score normalization computed from the training data.

Edge Features: For each edge connecting particles i and j, we define a feature vector e_{ij} consisting of:

Normalized relative position $(\mathbf{r}_j - \mathbf{r}_i)/\| \mathbf{r}_j - \mathbf{r}_i \|$
Euclidean distance $\| \mathbf{r}_j - \mathbf{r}_i \|$ (1 dimension)
Kernel function coefficient from the CSR matrix

The total edge feature dimension is 5. The inclusion of the CSR coefficient as an edge feature embeds the physics of the MPS discretization directly into the graph structure.

Boundary Treatment:
The graph structure exactly mirrors the sparse structure of the PPE coefficient matrix A. For free-surface particles, the Dirichlet boundary condition (P = 0) is directly reflected in the training labels. For Type-I wall boundary particles, their pressures are computed by the PPE solver and serve as ground-truth targets. Type-II buffer particles and dummy particles are not included as GNN nodes, as they do not participate in the PPE system.

The boundary influence is implicitly encoded in the GNN through: (1) the RHS term B_i, which accounts for density deviation caused by boundary neighbor configurations; (2) the neighbor count feature, which is naturally reduced for particles near boundaries; and (3) the CSR coefficients, which reflect the actual PPE matrix structure including boundary effects. This design ensures mathematical consistency between the GNN learning target and the PPE linear system.

Network Architecture: We employ a message-passing neural network (MPNN) following the encode-process-decode paradigm (Sanchez-Gonzalez et al. 2020). The encoder transforms input features into latent representations using multilayer perceptrons (MLPs). The processor consists of L stacked graph network layers, each performing message passing:

$$m_{ij}^{(l)} = \phi_e^{(l)}\left(\left[h_i^{(l)}, h_j^{(l)}, e_{ij}\right]\right)$$

$$h_i^{(l+1)} = \phi_v^{(l)}\left(\left[h_i^{(l)}, \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}^{(l)}\right]\right)$$

where $\boldsymbol{h}_i^{(l)}$ is the latent representation of node i at layer *l*, $\mathbf{m}_{ij}$ is the message from node *j* to node *i*, $\mathcal{N}(i)$ denotes the neighbors of node *i*, and ϕ_e and ϕ_v are learnable MLPs for edge and node updates, respectively. The decoder maps the final latent representations to the predicted pressure values.

We use a hidden dimension of 64 throughout the network. Each MLP consists of two layers with ReLU activation and layer normalization. Residual connections are employed in the processor to facilitate gradient flow. The number of message-passing layers L is a key hyperparameter that determines the effective receptive field of the network. We use L = 4 for full-radius graphs, following common practice in GNN-based physics simulation (Sanchez-Gonzalez et al., 2020), providing a receptive field of 4 × 4.01×dp = 16.04×dp. For reduced-radius graphs, we increase to L = 6 to ensure the theoretical receptive field (6 × 2.1×dp = 12.6×dp) exceeds the original influence radius. Residual connections are employed to mitigate potential over-smoothing issues.

Training Configuration: The network is trained to minimize the mean squared error between predicted and ground-truth pressures:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{P}_i - P_i\right)^2$$

where $\hat{P}_i$ is the predicted pressure and P_i is the ground-truth pressure obtained from the converged PCG solution. We use the AdamW optimizer with an initial learning rate of 10^{-3} and weight decay of 10^{-5}. A ReduceLROnPlateau scheduler reduces the learning rate by a factor of 0.5 when the validation loss plateaus for 5 consecutive epochs. Gradient clipping with a maximum norm of 1.0 is applied to stabilize training.

The GNN predicts normalized pressure values. The normalization procedure is as follows:

Normalization: P_norm = (P - μ) / σ, where μ and σ are the mean and standard deviation of pressure values computed from the training dataset
Network output: The GNN outputs normalized pressure P_norm for each particle
Denormalization: Physical pressure is recovered as P = P_norm × σ + μ

Edge Sampling Strategies

The primary challenge in applying GNNs to 3D particle simulations is the prohibitive memory requirement for storing and processing millions of edges. We investigate three distinct strategies for reducing the graph size while attempting to preserve the essential information for PPE prediction.

Strategy 1: Random Edge Sampling

In this approach, we retain the full influence radius $r_e\_lap = 4.01 \times dp$ for defining the graph topology but randomly sample only a fraction p of the edges. Specifically, for each particle pair (*i, j*) within the influence radius, we include the edge in the graph with probability p and discard it with probability 1-p. This process is applied independently to each edge during data loading. The key characteristics of this strategy are:

Graph topology spans the complete influence radius.
Edges are uniformly distributed across all distances from 0 to r_e_lap
Edge features are computed using the original kernel function
Expected memory reduction factor: 1/p

The random sampling preserves the statistical properties of the neighborhood distribution. Although individual particles may have varying numbers of retained neighbors, the ensemble average maintains representativeness across all interaction distances.

Table 1. Comparison of edge sampling strategies

Aspect	Strategy 1 (Random)	Strategy 2 (Small Radius, Small Kernel)	Strategy 3 (Small Radius, Large Kernel)
Graph topology radius	4.01 × dp	2.1 × dp	2.1 × dp
Kernel function radius	4.01 × dp	2.1 × dp	4.01 × dp
Avg. neighbors	~250	~40	~40
Memory reduction	~85%	~86%	~86%
Physical consistency	Full	Inconsistent	Partial
GNN layers	4	6	6

Strategy 2: Reduced Influence Radius with Consistent Kernel

This strategy reduces the influence radius from $r_e = 4.01 \times dp$ to $r_e = 2.1 \times dp$, which corresponds to the radius used for other MPS operators such as the gradient calculation. Only particle pairs within this smaller radius are included as edges.

The edge features, including the kernel function coefficient, are computed using the reduced radius:

$$w(r) = \frac{20}{17(r/r_e) + 3} - 1$$

where $r_e = 2.1 \times dp$. This ensures consistency between the graph structure and the edge features but introduces a discrepancy with the actual PPE coefficients used in the solver.

The expected neighbor count reduction follows from the volume ratio:

$$\left(r_e/r_{e\_lap}\right)^3 = (2.1/4.01)^3 \approx 0.14$$

This yields approximately 86% reduction in edge count. To compensate for the reduced receptive field per layer, we increase the number of GNN layers from 4 to 6, providing a theoretical receptive field of $6 \times 2.1 \times dp = 12.6 \times dp$, which exceeds the original influence radius.

Strategy 3: Reduced Influence Radius with Original Kernel

This hybrid strategy uses the same reduced influence radius $r_e = 2.1 \times dp$ for defining the graph topology but computes the edge features using the original larger radius in the kernel function:

$$w(r) = \frac{20}{17\left(r/r_{e\_lap}\right) + 3} - 1$$

The motivation is to maintain physical consistency with the PPE coefficients while benefiting from the reduced graph size. The graph structure is identical to Strategy 2, but the numerical values of the edge features differ. Physical consistency in this context refers to the alignment between graph edge features (kernel function coefficients) and the actual PPE matrix coefficients used by the MPS solver. Strategy 1 maintains full consistency by using the same kernel radius ($re\_lap = 4.01 \times dp$) for both graph construction and edge feature computation. Strategy 2 is inconsistent as it uses a different kernel radius ($re = 2.1 \times dp$). Strategy 3 achieves partial consistency by using the original kernel radius for coefficient computation but with truncated graph topology.

The fundamental difference lies in the graph topology: Strategy 1 maintains sparse connections across the full influence radius, while Strategies 2 and 3 completely eliminate connections beyond the reduced radius. As we demonstrate in the results section, this topological distinction has profound implications for the network's ability to learn the PPE solution.

EXPERIMENTAL SETUP

Test Case: 3D Sloshing in a Rectangular Tank

The numerical experiments are conducted on a three-dimensional liquid sloshing problem, which represents a challenging benchmark for particle-based methods due to its violent free-surface dynamics and complex pressure distributions. The computational domain consists of a rectangular tank with dimensions of 0.2 m × 0.1 m × 0.2 m. The tank is partially filled with water to an initial depth of 0.05 m, representing a filling ratio of 25%.

The tank undergoes harmonic oscillation along the longitudinal direction following a sinusoidal motion profile:

$$x(t) = A\sin(\omega t)$$

where A = 0.02 m is the oscillation amplitude and ω = 9.27 rad/s is the angular frequency. This frequency corresponds to the first natural frequency of the liquid in the tank, calculated from the linear sloshing theory:

$$\omega_n = \sqrt{\frac{\pi g}{L}\tanh\left(\frac{\pi h}{L}\right)}$$

where g is gravitational acceleration, L is the tank length, and h is the liquid depth. Operating at the resonant frequency induces large-amplitude sloshing with significant free-surface deformation and wave breaking, providing a demanding test case for the pressure prediction model.

The simulation employs a uniform particle spacing of dp = 0.004 m, resulting in a total of 42,348 particles, of which 12,397 are fluid particles and the remainder constitute the tank walls and dummy particles for boundary treatment. The time step is set to $\Delta t = 0.0002$ s, and the simulation is advanced for 100,000 steps, covering a physical duration of 20 seconds, which corresponds to approximately 29.5 complete oscillation cycles.

Data Generation and Preparation

Training data are generated by exporting the particle states and PPE solution at regular intervals during the MPS simulation. Specifically, a snapshot is saved every 500 time-steps, yielding 200 data files spanning the entire simulation period. Each data file contains the particle positions, the source term of the PPE, the converged pressure solution from the PCG solver, and the CSR-format sparse matrix coefficients representing

particle interactions. The data are processed differently by the three edge smapling strategies as introduced above. Table 2 summarizes the graph statistics for each strategy. Due to varying graph sizes across different frames (caused by changing fluid particle counts during sloshing), we use a batch size of 1, processing one frame at a time. Training data frames are loaded sequentially using PyTorch's DataLoader with on-the-fly graph construction from stored CSR matrices. The memory consumption reported in Table 2 refers to GPU memory during forward and backward passes.

Table 2. Graph statistics for different sampling strategies

Strategy	Avg. Neighbors	Total Edges (per frame)	Memory per Frame
Full graph	~250	~3.1 million	~300 MB
Strategy 1 (15% sampling)	~38	~470,000	~45 MB
Strategy 2 & 3 (small radius)	~40	~500,000	~48 MB

Training Configuration

From the 200 available data files, we select 100 files for the training experiments. The selected files are randomly shuffled and split into training and validation sets with an 4:1 ratio, resulting in 80 training samples and 20 validation samples. The GNN model architecture follows the specifications described before. All experiments and the trainings are conducted on a workstation equipped with an NVIDIA NVIDIA GeForce RTX 5060 GPU with 8 GB of memory. The GPU has 3840 CUDA cores and a compute capability of 12.0. The software environment includes Python 3.10.19, PyTorch 2.11.0.dev 20260104 + cu 128, and CUDA 12.8. Each model is trained for 150 epochs, which is sufficient for convergence as observed from the training curves. The model checkpoint with the lowest validation loss is saved for subsequent evaluation.

Evaluation Metrics

The prediction accuracy is evaluated using the following metrics:

Coefficient of Determination (R^2): Measures the proportion of variance in the true pressure explained by the predicted values:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(P_i - \hat{P}_i)^2}{\sum_{i=1}^{N}(P_i - \bar{P})^2}$$

where P_i is the true pressure, $\hat{P}_i$ is the predicted pressure, $\bar{P}$ is the mean true pressure, and N is the number of particles. An R^2 value close to 1 indicates excellent prediction accuracy.

Root Mean Square Error (RMSE): Quantifies the average magnitude of prediction errors:

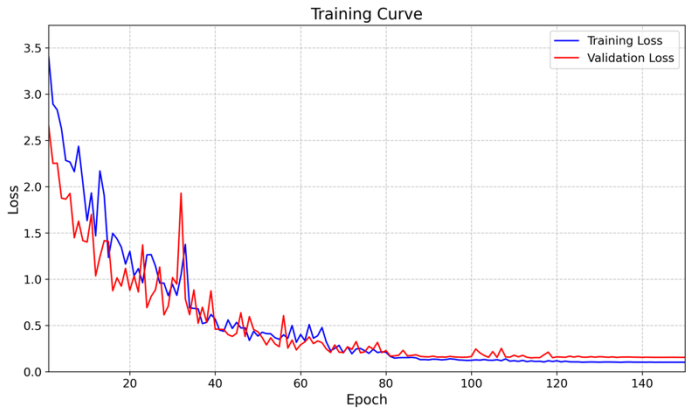
$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_i - \hat{P}_i)^2}$$

Mean Absolute Error (MAE): Provides a linear measure of average error magnitude:

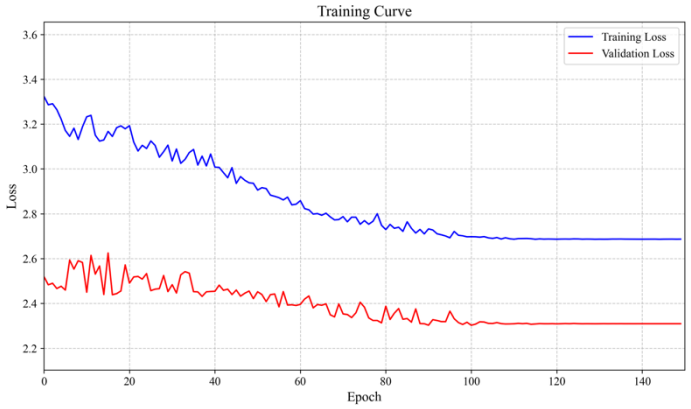
$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N} | P_i - \hat{P}_i |$$

Relative Error: Expresses the prediction error as a percentage of the true pressure range:

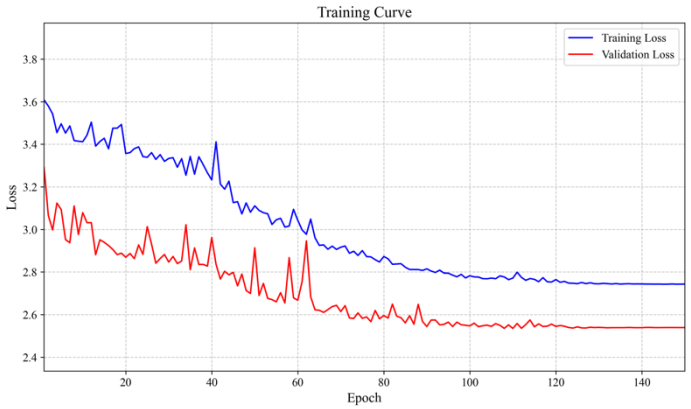
$$\text{Relative Error} = \frac{\text{RMSE}}{P_{\max} - P_{\min}} \times 100\%$$



(a)



(b)



(c)

Figure 2. Training curves for three different strategies(a) strategy 1 (random edge sampling);(b) strategy 2 (reduced radius with consistent kernel); (c) strategy 3 (reduced radius with original kernel)

RESULTS AND DISCUSSION

Training Convergence Analysis

Figure 2 presents the training curves for all three edge sampling strategies over 150 epochs. The contrast in convergence behavior reveals differences in the learning capability of the GNN under different graph topologies.

For Strategy 1, both training and validation losses exhibit consistent decrease throughout the training process. The training loss reduces from an initial value of approximately 3.5 to below 0.1 by epoch 150, while the validation loss follows a similar trajectory, stabilizing around 0.15. The close tracking between training and validation curves indicates good generalization without significant overfitting. The smooth convergence suggests that the randomly sampled graph, despite containing only 15% of the original edges, retains sufficient information for the network to learn the underlying pressure-velocity relationship.

In contrast, Strategies 2 and 3 demonstrate severely impaired learning. For Strategy 2, the training loss decreases slowly from 3.2 to approximately 2.4 over 150 epochs, while the validation loss plateaus around 3.2 with high variance. The large gap between training and validation losses, combined with the high absolute values, indicates that the model fails to capture the essential physics. Strategy 3 shows similar behavior, with the training loss stabilizing around 2.7 and validation loss around 2.5. Despite using physically consistent kernel coefficients, the truncated graph topology prevents effective learning.

The training curves alone provide compelling evidence for the failure of radius truncation strategies. Given that the losses for Strategies 2 and 3 remain an order of magnitude higher than Strategy 1, we focus our detailed prediction analysis on Strategy 1, while providing summary statistics for comparison.

Table 3 presents the training time comparison among the three strategies. Despite similar computational costs per epoch (due to comparable edge counts after sampling), only Strategy 1 achieves convergence, demonstrating that the performance difference lies in learning effectiveness rather than computational efficiency.

Table 3. Training time comparison among the three strategies

Strategy	Time per Epoch	Total training time (150 epochs)	Convergence
Strategy 1	~45s	~112min	Converged (~80 epochs)
Strategy 2	~50s	~125min	Did not converge
Strategy 3	~50s	~125min	Did not converge

Prediction Accuracy for Random Edge Sampling

The trained model from Strategy 1 is evaluated on all 200 data frames to assess its prediction capability across different sloshing phases. Table 4 summarizes the overall prediction accuracy:

The model achieves an average R^2 of 0.968, indicating that approximately 97% of the pressure variance is explained by the GNN predictions. The low standard deviation (0.015) demonstrates consistent performance across different sloshing phases.

Table 4. Prediction accuracy metrics for random edge sampling

Metric	Mean	Std. Dev.	Min	Max
R^2	0.968	0.015	0.912	0.989
RMSE (Pa)	38.2	8.5	22.1	67.3
MAE (Pa)	28.6	6.2	17.8	48.5
Relative Error (%)	6.8	1.5	3.9	11.8

Figure 3 presents scatter plots comparing predicted versus true pressures for six representative test frames spanning different simulation times. The data points cluster tightly around the perfect prediction line ($y = x$) across all frames, with R^2 values ranging from 0.980 to 0.989. The scatter is slightly larger in the high-pressure region ($P > 400$ Pa), which corresponds to impact events where pressure gradients are steeper and more challenging to predict accurately.

To visualize the spatial distribution of prediction accuracy, Figure 4 displays the pressure fields for selected frames, showing the PCG ground truth, GNN prediction, and absolute error. The GNN successfully captures the overall pressure distribution, including the hydrostatic gradient, dynamic pressure variations due to sloshing, and localized high-pressure regions near impact zones. The error field reveals that the largest discrepancies occur near the free surface and at wall-fluid interfaces, where particle configurations are most irregular. Nevertheless, the maximum local errors remain within acceptable bounds for engineering applications.

Physical Interpretation

The striking performance difference between random edge sampling and radius truncation can be understood through the mathematical properties of the Pressure Poisson Equation. The PPE is an elliptic partial differential equation, characterized by global information propagation: the pressure at any point depends on the entire domain through the boundary conditions and source term distribution. In the MPS discretization, this global dependence is realized through the sparse matrix A, where each particle is directly coupled to neighbors within the influence radius r_{e_lap}. The iterative PCG solver propagates information across the domain through successive matrix-vector multiplications, with the number of iterations reflecting the global coupling strength. When we construct a GNN on this particle system, the message-passing mechanism mimics the information propagation in the iterative solver. Each GNN layer allows information to travel one "hop" through the graph, with L layers providing a receptive field of $L \times r_e$. For the GNN to approximate the PPE solution, it must be able to propagate information across the entire domain—or at least capture the essential long-range correlations.

Random edge sampling at 15% retains the full influence radius topology while sparsifying the graph. Crucially, edges at all distances from 0 to r_{e_lap} have equal probability of being retained. This means that even after sampling, there exist pathways for information to propagate between particles separated by distances up to r_{e_lap} in each layer. The statistical representativeness of the sampled edges allows the GNN to learn an approximate message-passing scheme that captures the essential physics.

In contrast, reducing the influence radius to $r_e = 2.1 \times dp$ completely eliminates all edges between particles separated by distances greater than r_e. No amount of additional GNN layers can recover these missing connections because the underlying graph simply does not contain them. The theoretical receptive field of $6 \times 2.1 \times dp = 12.6 \times dp$ exceeds the original influence radius, but this is misleading: information can only

propagate through paths that exist in the graph, and all long-range edges are absent. In the randomly sampled graph, a sparse but complete network of edges connects near and distant neighbors, enabling multi-hop information flow. In the truncated graph, a dense local neighborhood exists, but particles beyond r_e are completely disconnected, creating information barriers that the GNN cannot overcome.

This analysis reveals a fundamental principle for GNN-based PDE solvers: for elliptic equations requiring global information propagation, preserving the graph topology is more important than maintaining edge density. We term this the "sparse but complete" principle, which states that a sparse graph with edges spanning all relevant length scales will outperform a dense graph with truncated connectivity.

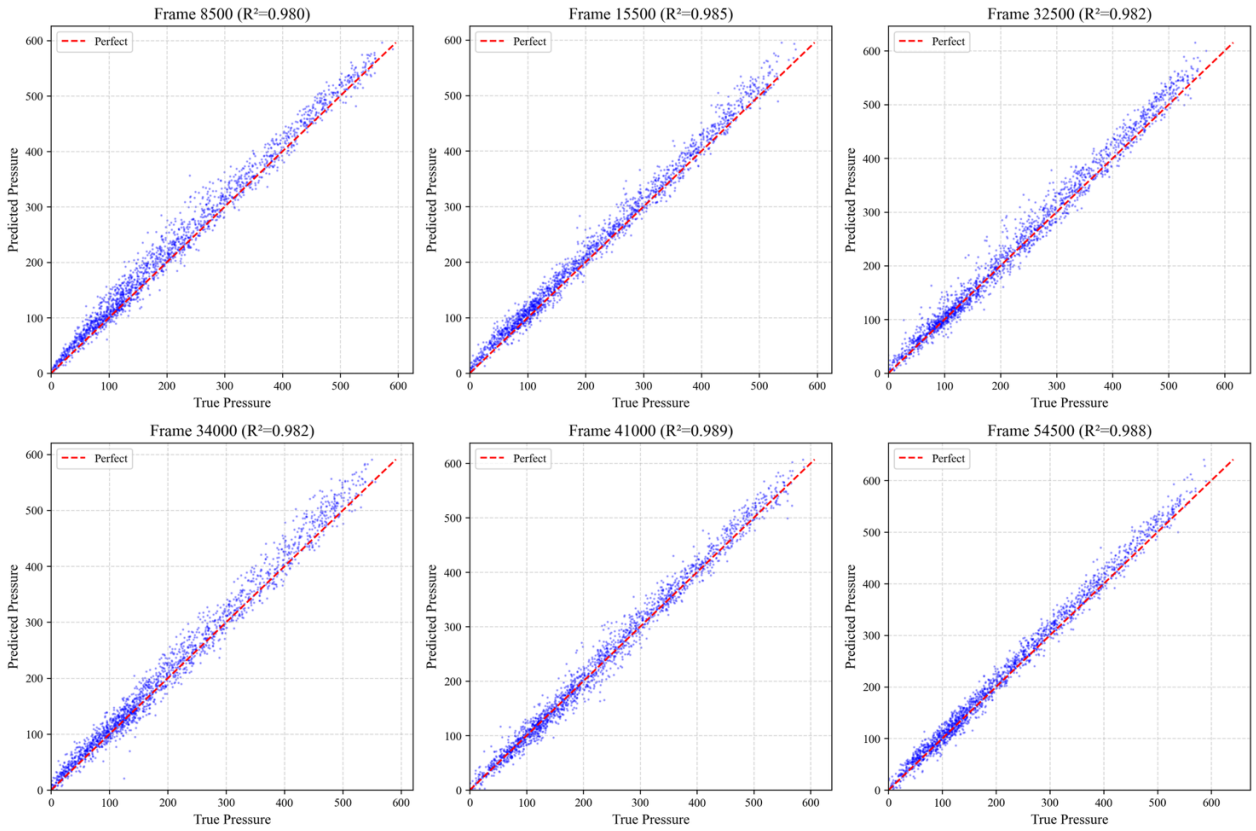


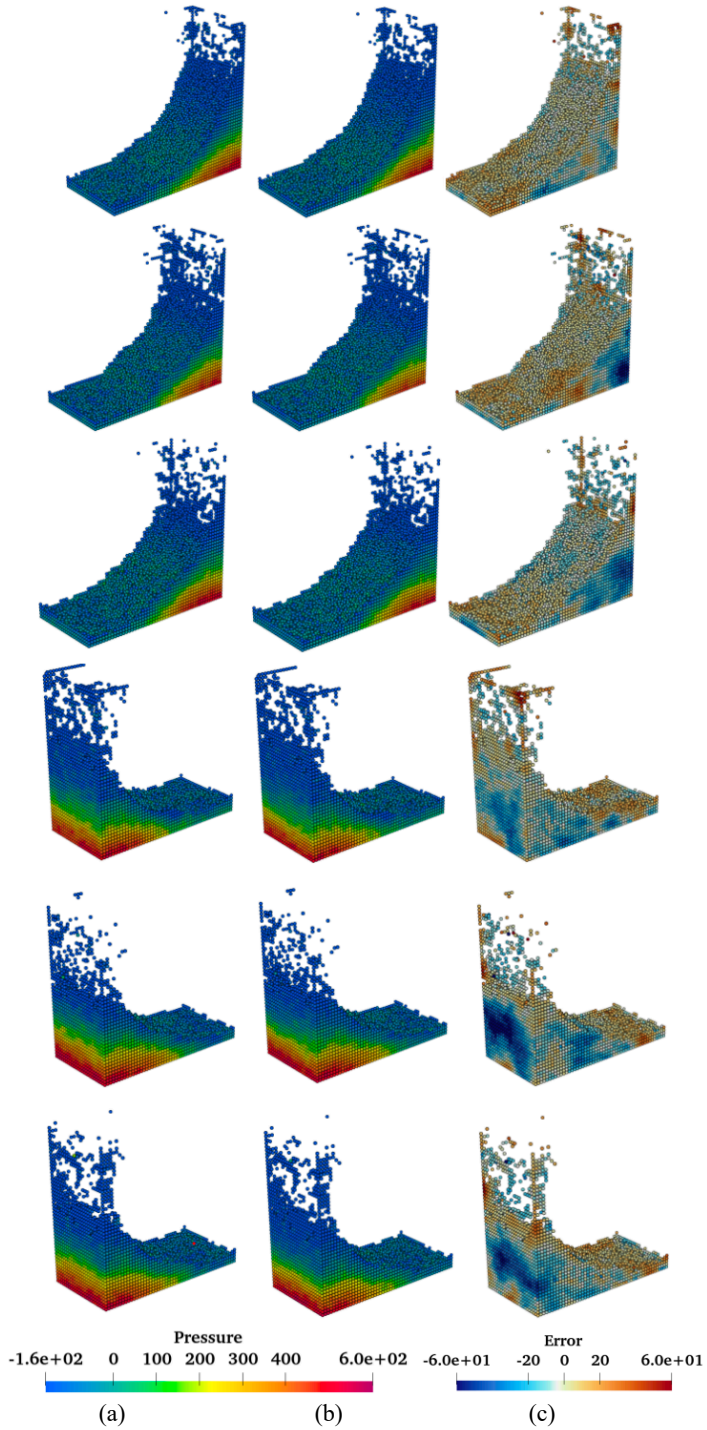
Figure 3. Scatter plots of predicted vs. true pressure for representative test frames

CONCLUSIONS

This paper investigated edge sampling strategies for memory-efficient training of Graph Neural Networks to predict pressure fields in MPS simulations. Through experiments on a 3D sloshing benchmark using the MPS method, we compared three strategies: (1) random edge sampling preserving the full influence radius topology, (2) reduced influence radius with consistent kernel function, and (3) reduced influence radius with original kernel function.

Our experiments reveal a important finding that random edge sampling significantly outperforms radius truncation methods, achieving $R^2 > 0.96$ with only 15% of edges retained, while radius truncation methods fail to converge regardless of kernel function choice. We provide a physical interpretation linking this observation to the elliptic nature of the Pressure Poisson Equation, which requires global information propagation. Random sampling preserves sparse but complete pathways for information flow across the full influence radius, while radius truncation creates insurmountable information barriers. This leads us to propose the "sparse but complete" principle: for GNN-based solvers of elliptic PDEs, graph topology preservation is more critical than edge density.

Our findings may have some practical implications for researchers and engineers seeking to apply GNN-based acceleration techniques to large-scale 3D particle simulations. First of all, for memory-constrained training, random edge sampling at 10-20% provides an effective approach to reduce GPU memory requirements by 80-90% while maintaining high prediction accuracy. Second, for architecture design, increasing GNN depth cannot compensate for truncated graph topology. Resources are better spent on maintaining topological completeness than on deeper networks. It should be noted that the current work focuses on demonstrating the effectiveness of edge sampling strategies for GNN-based PPE prediction as a supervised learning task. Integration into a complete MPS time-marching solver, which involves additional considerations such as error accumulation and numerical stability, remains as future work.



(a) (b) (c)

Figure 4. Pressure field visualization: (a)PCG ground truth, (b)GNN prediction, and (c)absolute error

REFERENCES

Battaglia, PW, Hamrick, JB, Bapst, V, Sanchez-Gonzalez, A, Zambaldi, V, Malinowski, M, et al. (2018). "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:*1806.01261.

Bentivoglio, R, Isufi, E, Jonkman, SN, and Taormina, R. (2023). "Rapid spatio-temporal flood modelling via hydraulics-based graph neural networks", *Hydrology and Earth System Sciences*, 27: 4227-46.

Chen, X, Xi, G, and Sun, ZG. (2014). "Improving stability of MPS method by a computational scheme based on conceptual particles", *Computer Methods in Applied Mechanics and Engineering*, 278: 254-71.

Gingold, RA, and Monaghan, JJ. (1977). "Smoothed particle hydrodynamics: theory and application to non-spherical stars", *Monthly Notices of the Royal Astronomical Society*, 181: 375-89.

Khayyer, A and Gotoh, H. (2009). "Modified Moving Particle Semi-implicit methods for the prediction of 2D wave impact pressure", *Coastal Engineering*, 56: 419-40.

Kipf, T (2016). "Semi-supervised classification with graph convolutional networks", *arXiv preprint arXiv:*1609.02907.

Koshizuka, S and Oka Y. (1996). "Moving-Particle Semi-Implicit Method for Fragmentation of Incompressible Fluid", *Nuclear Science and Engineering*, 123: 421-34.

LeCun, Y, Bengio, Y, and Hinton G. (2015). "Deep learning", *Nature*, 521: 436-44.

Li, Z and Farimani, AB. (2022). "Graph neural network-accelerated Lagrangian fluid simulation," *Computers & Graphics*, 103: 201-11.

Lino, M, Fotiadis, S, Bharath, AA and Cantwell, CD. (2023). "Current and emerging deep-learning methods for the simulation of fluid dynamics," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 479.2275.

Pfaff, T, Fortunato, M, Sanchez-Gonzalez, A and Battaglia, P. (2020). "Learning mesh-based simulation with graph networks," *International conference on learning representations* , 2010.03409.

Raissi, M, Perdikaris, P and Karniadakis GE. (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, 378: 686-707.

Sanchez-Gonzalez, A, Godwin, J, Pfaff, T, Ying, R, Leskovec, J and Battaglia, P. (2020). "Learning to simulate complex physics with graph networks." In *International conference on machine learning*, 8459-68..

Toshev, AP, Erbesdobler, JA, Adams, NA, and Brandstetter J. (2024). "Neural sph: Improved neural modeling of lagrangian fluid dynamics," *arXiv preprint arXiv:*2402.06275.

Yao, Q, Wang, Z, Zhang, Y, Li, Z and Jiang, J. (2023). "Towards real-time fluid dynamics simulation: a data-driven NN-MPS method and its implementation", *Mathematical and Computer Modelling of Dynamical Systems*, 29: 95-115.