Machine Learning Model for Gas–Liquid Interface Reconstruction in CFD Numerical Simulations †

Tamon Nakano 1,* , Michele Alessandro Bucci 1, Jean-Marc Gratien 2 and Thibault Faney 2

1 Institute National de Recherche en Informatique et en Automatique (INRIA), LISN, bât 660, Université Paris-Saclay, 91405 Orsay, France; bucci.malessandro@gmail.com
2 IFP Énergies-Nouvelles, 1 et 4 av Bois Préau, 92852 Rueil-Malmaison, France; jean-marc.gratien@ifpen.fr (J.-M.G.); thibault.faney@ifpen.fr (T.F.)

* Correspondence: tamon.nakano@gmail.com
† This paper is an extended version of our paper published in the Proceedings of the 8th European Congress on Computational Methods in Applied Sciences and Engineering, Oslo, Norway, 5–9 June 2022.

Abstract: The volume of fluid (VoF) method is widely used in multiphase flow simulations to track and locate the interface between two immiscible fluids. The relative volume fraction in each cell is used to recover the interface properties (i.e., normal, location, and curvature). Accurate computation of the local interface curvature is essential for evaluation of the surface tension force at the interface. However, this interface reconstruction step is a major bottleneck of the VoF method due to its high computational cost and low accuracy on unstructured grids. Recent attempts to apply data-driven approaches to this problem have outperformed conventional methods in many test cases. However, these machine learning-based methods are restricted to computations on structured grids. In this work, we propose a machine learning-enhanced VoF method based on graph neural networks (GNNs) to accelerate interface reconstruction on general unstructured meshes. We first develop a methodology for generating a synthetic dataset based on paraboloid surfaces discretized on unstructured meshes to obtain a dataset akin to the configurations encountered in industrial settings. We then train an optimized GNN architecture on this dataset. Our approach is validated using analytical solutions and comparisons with conventional methods in the OpenFOAM framework on a canonical test. We present promising results for the efficiency of GNN-based approaches for interface reconstruction in multiphase flow simulations in the industrial context.

Keywords: graph neural network; interface reconstruction; VoF method; unstructured mesh

1. Introduction

Gas–liquid interfacial flows are found in a wide range of industrial applications, including cooling systems for electrical engines, chemical reactor modeling, and pore-scale flows in porous media. Among the numerous numerical methods developed to simulate such gas–liquid flows, an important issue is tracking the motion of the free interface. Two classes of methods have emerged in recent decades: front-tracking methods and interface-capturing methods. The latter are more suitable for industrial applications, as they can more easily handle topological changes in the free interface (e.g., breakup or coalescence) in complex flows. The volume of fluid (VoF) method [1,2] is an interface-capturing method that retrieves interface properties (i.e., normal, location, curvature) from the volume fraction of each fluid. However, a bottleneck in the VoF method is the cost of accurately computing
the local interface curvature, which is essential for evaluating the surface tension force at the interface. Additionally, while the VoF method is accurate on regular structured grids, it introduces unphysical fluid motions (Harvie et al. [3]) when applied to unstructured tetrahedral grids.

Several attempts to address these shortcomings based on machine learning algorithms have been made recently. Qi et al. [4] proposed a two-layer neural network to predict the interface curvature on a 2D Cartesian grid. The model was trained on a synthetic dataset to predict the curvature for a given volume fraction in neighboring cells. This model was then implemented in a CFD solver and evaluated on test cases with moving interfaces. The prediction was accurate enough to consider this approach a valid solution. Patel et al. [5] extended this approach to 3D Cartesian geometry using a two-layer neural network. They trained their model on a synthetic dataset generated from a spherical surface and evaluated its performance on analytical and CFD-based test cases. Their approach outperformed the convolution method, and even matched the accuracy of state-of-the-art VoF methods such as the height function method. Svyetlichnyy [6] also trained a neural network to predict the interface properties for 2D and 3D surface reconstruction. Their model showed good performance in predicting normals, but not in predicting curvature and surface locations. While these results are promising, they are limited to Cartesian grids due to the fixed size input restriction of the machine learning algorithms used by the authors.

Unfortunately, most industrial problems involving multiphase flow simulations are discretized on unstructured grids, as this enables efficient handling of complex geometries with irregular shapes. An overview of the algorithms used to generate such grids is provided in Owen [7]. As described in Katz and Sankaran [8], persistent challenges remain due to the poor quality of the generated meshes. Finally, due in part to the costs associated with interface reconstruction, the application of VoF methods is generally restricted to simulations within relatively small physical domains. For complex chemical processes that require simulating flows with a large number of bubbles, as described in [9,10], the VoF method is not feasible for the entire simulation. Due to its accuracy, the VoF method is sometimes employed in conjunction with other methods that are better suited for large-scale simulations. In multiscale coupled strategies, it typically helps to accurately determine local parameters that model the impact of bubbles (treated as particles) on the overall flow dynamics. Table 1 presents a comparative analysis of various interface reconstruction approaches, focusing on their accuracy, stability, and computational cost. Accuracy is assessed based on the ability to compute exact solutions, while stability is evaluated by the algorithms’ performance irrespective of mesh quality. The computational cost is considered in terms of the total expense incurred during each timestep calculation in the Volume of Fluid (VoF) method.

|                      | Structured Grid | Unstructured Grid | Computational Cost |
|----------------------|-----------------|-------------------|--------------------|
| VoF                  | Accurate        | Unstable          | Expensive          |
| Neural Network       | Accurate        | Not available     | Improved           |

In this work, we propose a graph neural network (GNN)-based algorithm designed to enhance the accuracy of interface reconstruction on unstructured meshes. To reach this objective, we also propose a novel data generation process that is able to generate surfaces in three dimensions that are more representative of CFD simulations, such as deformed grid cells intersecting highly curved surfaces. We subsequently train the GNN on our generated dataset and integrate the resulting interface reconstruction model in OpenFoam.
In Section 2, we provide a foundation for the volume of fluid (VoF) method. Next, in Section 3, we present a new GNN architecture for recovering the interface properties of each mesh element from the neighboring mesh geometry and volume fraction, generalizing the work presented in Patel et al. [5] to unstructured meshes. In Section 4, we describe how to generate a training dataset that enables the model to make accurate predictions on a wide range of unseen test data that are representative of industrial applications. Section 5 details model training and hyperparameter optimization. We then validate our methodology in Section 6 using several synthetic test cases and a canonical test simulated with the OpenFOAM framework, on which we benchmark our method to the conventional interface reconstruction method.

This paper is an extended version of our paper published in [11]

2. Volume of Fluid Method

There is a wide variety of numerical methods for solving multiphase flow problems. Most of these are based on the resolution of the Navier–Stokes (NS) equations based on the momentum and mass conservation (Equation (1)), which can be written as follows:

\[
\begin{align*}
\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) &= -\nabla p + \rho \mathbf{g} + \nabla \cdot (\mu \nabla \mathbf{U}) + \rho \mathbf{S} + \mathbf{F}_s \\
\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U}) &= 0
\end{align*}
\]

(1)

where \(\rho\), \(\mathbf{U}\), and \(p\) are the density, velocity, and pressure unknowns, \(\mu\) and \(\mathbf{g}\) are the dynamic viscosity and gravity, and \(\mathbf{S}\) is an extra force acting on the fluid (e.g., due to a moving object). The interface between the two fluids induces a surface tension force \(\mathbf{F}_s\), modeled as follows:

\[
\mathbf{F}_s = \sigma_s \kappa \mathbf{n} \delta(x - x_s)
\]

(2)

where \(x_s\) represents the coordinates of the interface, \(\sigma_s\), \(\kappa\), and \(\mathbf{n}\) are the surface tension coefficient, mean curvature of the interface, and normal direction to the interface, and \(\delta\) is the Kronecker \(\delta\), which ensures that surface tension is only considered for points on the interface.

The volume of fluid (VoF) method allows for the resolution of two-phase flow problems through a single-field formulation of the NS equations. Each fluid is represented in the VoF method with the characteristic function \(\chi(x)\), defined as \(\chi(x) = 1\) if \(x\) is in the fluid and \(\chi(x) = 0\) otherwise. Let us consider two fluids related to a primary phase \(A\) and secondary phase \(B\) partitioning the simulation domain \(\Omega\). The VoF method is an Eulerian–Eulerian approach that is capable of accurately capturing the interface from the volume fraction of the primary phase in each cell, denoted by \(\alpha = \frac{\int \chi_A}{V}\), where \(\tau\) is a cell with volume \(V\) of a mesh discretization \(\Omega_h\) of \(\Omega\). Figure 1a illustrates a bubble defined by a characteristic function \(\chi_A\) in blue and the discretization on a 2D tetrahedral mesh with cells colored depending on the value of the volume fraction \(\alpha\). Physical quantities such as fluid density, viscosity, and velocity are expressed as characteristic function-weighted sums. For instance, letting \(\rho_A\) and \(\rho_B\) be the respective densities of the primary and secondary phases, the fluid density is defined as \(\rho = \chi_A \rho_A + (1 - \chi_A) \rho_B\).

The evolution of the interface implicitly defined by \(\alpha(x) = 0\) is tracked with the additional advection equation shown below as Equation (3).

\[
\frac{\partial \alpha}{\partial t} + \nabla \cdot \alpha \mathbf{U} = 0
\]

(3)

In the VoF method, the main NS equations in (1) are solved with the PIMPLE algorithm, a combination of the PISO (Pressure Implicit with Splitting of Operator) and SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) methods (more details can be found in
The advection equation in (3) is solved with a numerical scheme, either MULES (Deshpande et al. [13]) or IsoAdvector (Roenby et al. [14]). MULES is designed to ensure boundedness and stability in the face of sharp gradients such as those found at fluid interfaces. It employs a flux limiter that adjusts the interface to prevent the overshoots and undershoots that can lead to non-physical values in volume fraction fields. On the other hand, the IsoAdvector method focuses on achieving a more accurate representation of the interface geometry by using an isosurface-based advection approach. This method tends to provide better interface sharpness and reduced numerical diffusion compared to traditional VoF methods.

![Figure 1. Gas–liquid interface representation with volume fraction: (a) discretization of continuous interfaces and (b) interface reconstruction.](image)

The interface reconstruction step is carried out with the piecewise linear interface construction (PLIC) method; the interface is then uniquely determined in each cell by the interface normal $\mathbf{n}$ and the interface center $M$ as illustrated in Figure 2. The interface normal $\mathbf{n}$ is usually calculated as the gradient of the volume fraction $\alpha$:

$$\mathbf{n} = -\frac{\nabla \alpha}{\|\nabla \alpha\|}$$

while the interface center $M$ is determined such that the interface truncates the cell with a volume that matches the volume fraction.

Various procedures have then been developed for the estimation of the interface curvature $\kappa$ required to compute the surface tension force $F_\sigma$. $\kappa$ is usually defined as

$$\kappa = \nabla \cdot \mathbf{n}.$$ 

Currently, the convolution (CV) (Williams et al. [15], Williams et al. [16]), height function (HF) (Lörstad et al. [2]) and radial distance function (RDF) (Cummins et al. [17]) methods are the most standard and accurate approaches.

![Figure 2. Example of PLIC; the colored polyhedral cell corresponds to the volume fraction $\alpha$ of this cell.](image)

The objective of our work is to replace computation of the interface reconstruction and interface curvature with a machine learning algorithm that directly predicts these surface properties from the volume fraction in the neighboring cells and the coordinates of these cells.
3. Graph Neural Network Model

Interface reconstruction in the VOF method consists of geometric algorithms that cut each diphasic cell into two monophasic subcells and provide the center, normal, area, and mean curvature of the subcells’ interface. To tackle the computational complexities inherent in these geometric algorithms, recent efforts have been made to develop deep learning-based surrogate models. These methodologies typically involve the creation of a dataset where input features (i.e., geometric configurations defined by the local mesh and values of the volume fraction) are paired with corresponding output labels (i.e., interface normal, curvature, and position). Subsequently, a neural network model architecture is designed and the model is trained and validated to ensure its accuracy and efficiency. The objective is to produce a surrogate model that can rival the performance of the original geometric algorithms, offering a more computationally efficient alternative. To date, the methods put forth have predominantly relied on multilayer perceptrons (MLP) and convolutional neural networks (CNN); however, these architectures face limitations when confronted with irregularly-organized data such as unstructured meshes. In this section, we present an innovative approach utilizing graph neural networks (GNN). The rationale behind this choice lies in the ability to represent unstructured meshes as graphs.

3.1. GNN Architecture Design

The work by Zhou et al. [18] provides a comprehensive review of how GNNs can be used for unstructured meshes. A succinct summary of the fundamental principles underlying these GNN methods is presented in Appendix A.

3.1.1. Data Modeling

The objective of the proposed GNN approach is to predict the interface normal, curvature, center, and area for each interface cell (i.e., where $\alpha \in [0, 1]$) from a graph containing the information from all neighboring tetrahedral cells. Therefore, each interface cell is associated with its own graph and interface properties, representing a single data point in the training process. In each graph, nodes correspond to neighboring cells, while edges represent cell connections through a shared face. The dimension of the graph is defined by all tetrahedral elements sharing at least one vertex with the current cell. Node features consist of the cell vertices coordinates and volume fraction. Figure 3b sketches the reconstructed surface properties for a three-dimensional tetrahedral cell, while Figure 3c shows the graph for a given interface cell in two dimensions.

![Figure 3](image)

**Figure 3.** Gas–liquid interfaces representation with GNN graphs: (a) schematic diagram of a graph, (b) surface properties, and (c) graph structure.

In the training algorithm, all coordinates are normalized by $L = \max(L_x, L_y, L_z)$, where $L_x$, $L_y$, and $L_z$ are the maximum dimensions in the $x$, $y$, and $z$ directions over all mesh cells, ensuring that each tetrahedral cell belongs to the unitary cube. The corresponding
dimensionless curvature $H$ is $H = H' * L$, where $H'$ is the original curvature and the corresponding dimensionless area of the reconstructed surface $S$ is $S = S'/L^2$, where $S'$ is the original area. The coordinates of the interface center $M$ are expressed in the barycentric system as $M = p_1P_1 + p_2P_2 + p_3P_3 + p_4P_4$, where $P_1, P_2, P_3, P_4$ are the coordinates of each vertex of the tetrahedral cell (see Figure 3b) and $p_1 + p_2 + p_3 + p_4 = 1$.

3.1.2. Neural Network Architecture

The complete architecture is illustrated in Figure 4d. It consists of three interconnected submodels, designated as NN-1, NN-2, and NN-3. The inputs to these submodels are depicted by the respective arrows directed toward each model, which outline the flow of information within the architecture. The predictive process unfolds as follows:

- NN-1 determines the interface normal and curvature.
- NN-2 identifies the center of the plane within the VOF approximation.
- NN-3 computes the area of the plane intersecting the target element.

The predictive sequence is inherently hierarchical: NN-2 relies on the outputs generated by NN-1; in turn, NN-3 depends on the results provided by NN-2. The entire architecture shown in Figure 4d is simultaneously trained by backpropagation during the training process.

NN-1 (Figure 4a) shows a graph neural network that predicts both the normal and the curvature from the input graph associated with a given interface cell. The node features are the coordinates of the cell vertices and the volume fraction $\alpha$ of the cells. More specifically, at each node we have the three-dimensional coordinates for the four vertices of a tetrahedron $\{P_v \in \mathbb{R}^3, \forall \nu \in \mathcal{V}\}$. Here, the coordinates of the vertices are flattened in a manner such that $P_v = (P_1, P_2, ..., P_4) = (x_1, y_1, z_1, x_2, y_2, ..., z_4)$, where $1,2,3,4$ are the number of the vertices of the tetrahedron. We then concatenate $P_v$ with the volume fraction of the same node as $x_v = (\alpha, P_v) \in \mathbb{R}^3$. Therefore, an input feature of NN-1 for a graph having $n$ nodes can be denoted as $X_1 \in \mathbb{R}^{13 \times n}$. The first two layers of NN-1 are SAGEConv layers that process the input graph and return a new graph with updated features. Node features are further processed by two linear layers. Skip connections are employed to prevent the vanishing gradient problem. The following global mean pool layer returns the graph-averaged node features. A latent representation of the initial graph is then recovered, then the model splits into two branches of linear layers for prediction of the normal $n \in \mathbb{R}^3$ and curvature $H$.

NN-2 (Figure 4b) shows a simple MLP network that predicts the interface center $M$ from the local information available in the current interface cell, including the normal $n$ predicted by NN-1. We concatenate the volume fraction $\alpha$, coordinates of the vertices $P_{cell} \in \mathbb{R}^{3 \times 4}$, and normal $n$, which is fed in the form of the dot product with $P_{cell}$, that is, $n \cdot P_{cell} \in \mathbb{R}^4$. This helps the neural network to discriminate vertices that are above or below the interface. Finally, a concatenated form $X_2 = (\alpha, n \cdot P_{cell}, P_{cell}) \in \mathbb{R}^{17}$ is used as input, where $P_{cell} = \{P_{cell1}, P_{cell2}, ..., P_{cell4}\} \in \mathbb{R}^{12}$ is the flattened $P_{cell}$ and $P_{cell1}, P_{cell2}, ..., P_{cell4}$ are the coordinates of the vertices. At the output, the barycentric coordinates $p_j = 1,2,3,4 \in \mathbb{R}^4$ are recovered, then a final $L1$ normalization is applied on the output to enforce $\sum_{j=1}^{4} P_j = 1$.

NN-3 (Figure 4c) shows another simple MLP that predicts the interface area from the available cell quantities $X_3 = (n \cdot P_{cell}, \hat{P}_{cell}, p_{1,2,3,4}) \in \mathbb{R}^{19}$. A sigmoid function is used after the final layer. Due to the nondimensionalization by $L$ of the coordinates, the maximum possible area is $\sqrt{2}/2$, which is less than 1, allowing us to skip normalization of the target area.
Figure 4. Detailed schematic diagram of the architecture. The numbers in the figure represent the size of the node features (Input graph), the size of the current layer (Linear or SageConv), and the dropout probability (Dropout): (a) NN-1: Normal/Curvature; (b) NN-2: Center; (c) NN-3: Area. (d) Entire architecture.

4. Dataset

Patel et al. [5] employed spherical surfaces with varying radii to create their synthetic dataset and train their model. We can observe their prediction of the normal on a sinusoidal surface, as shown in Figure 5. We observe that the largest errors are located on saddle point regions; it is likely that the model trained on a sphere-based dataset is biased and cannot generalize to non-spherical surfaces. Because a surface in three dimensions is locally characterized by its principal curvatures $\kappa_1$ and $\kappa_2$, we created a new dataset derived from paraboloidal surfaces, defined by the equation

$$\frac{x^2\kappa_1}{2} + \frac{y^2\kappa_2}{2} = z. \quad (6)$$

Figure 6 illustrates an example of such a paraboloidal surface with curvatures of opposite signs. The dataset creation process was then adapted from Patel et al. [5] to account for sampling principal curvatures. The exact data generation process is detailed in Appendix A.2 and illustrated in Figure 7.

Figure 5. Predicted curvature by Patel et al. [5].
Figure 6. Paraboloid surface locally defined by its principal curvature $\kappa_1$ and $\kappa_2$.

Figure 7. Interface and normal at the surface of interest (top left), voxels containing the graph (top right), and interfaces found in the voxels containing the graph (bottom).

With this procedure, we generated a dataset with $N_{all} = 700,000$ paraboloids. The distribution of the graph features of the dataset elements is shown in Figure 8. We observe that $\kappa_1$ and $\kappa_2$ follow a uniform distribution, as designed. Figure 8b shows the histogram of the volume fraction $\alpha$ of the cell corresponding to the first node of each graph. Even if this distribution were not controllable a priori in our generation algorithm, the figure indicates that quasi-uniformity was reached by our dataset generation process. Figure 8c displays the distributions of the rotations in each direction, as explained in step 6 of the data generation process. Figure 8d shows the normals at the center point of the rotated paraboloid, which appear to be evenly distributed, confirming the 3D uniformity of the rotation. The mean curvature, defined as $H = \frac{\kappa_1 + \kappa_2}{2}$, follows the expected triangular shape in the histogram shown in Figure 8e, as it results from the sum of two independent uniform distributions. The area histogram in Figure 8f exhibits a unimodal distribution with a long tail, which is not controllable a priori. As observed, most properties exhibit a uniform
distribution; therefore, we expect such a dataset generation procedure to better represent general surfaces than a dataset solely based on spherical surfaces.

Figure 8. Histograms on the generated dataset: (a) $k_1$, $k_2$; (b) $\alpha$; (c) rotation on $x, y, z$; (d) normal (only 1 k graphs are displayed for visibility reasons); (e) curvature $H$; (f) area $A$.

5. Model Training and Optimization

5.1. Training

We utilized a dataset comprising 500,000 graphs selected from a total of $N_{\text{all}} = 700,000$ to train our model. The choice of this specific data size aligns with the considerations outlined in Section 5.3. The 500,000 graph dataset was partitioned into a 7:2:1 ratio for training, validation, and testing, respectively.

In a deep architecture such as the one shown in Figure 4, the backpropagation of gradients becomes less effective due to the depth of the entire architecture. At the same time, the errors from intermediate predictions are propagated to the subsequent models.

To address challenges associated with ineffective training due to model depth and mitigate error propagation from intermediate predictions, auxiliary loss functions are introduced. These functions incorporate ground truth labels (“wl”, with-label) as inputs to models responsible for computing the center and area. By addressing the vanishing gradient
problem, these auxiliary losses help to prevent error compounding from intermediate predictions. In NN-2, the input normal can either be the predicted value from NN-1 (“wol”, without-label) or the ground truth label (“wl”, with-label), necessitating two distinct loss functions. Similarly, this principle applies to the input center in NN-3. Each loss function employs an L2-normalized mean squared error (MSE). The overall loss is then defined as follows:

\[
\text{Loss}_{\text{total}} = \text{Loss}_{\text{normal}} + \text{Loss}_{\text{curv}} + \text{Loss}^{\text{wl}}_{\text{center}} + \text{Loss}^{\text{wol}}_{\text{center}} + \text{Loss}^{\text{wl}}_{\text{area}} + \text{Loss}^{\text{wol}}_{\text{area}}
\]

where, \( \text{Loss}_{\text{normal}} \), \( \text{Loss}_{\text{curv}} \), \( \text{Loss}^{\text{wl}}_{\text{center}} \), and \( \text{Loss}^{\text{wol}}_{\text{area}} \) represent the loss functions associated with the normal, curvature, center, and area predictions, respectively.

The Pytorch-integrated adaptive learning rate was employed and configured within the range of \( 1 \times 10^{-2} \) to \( 1 \times 10^{-6} \). The learning rate was reduced by a factor of 0.9 when the validation loss function failed to improve for five consecutive epochs. The batch size was set to 512. To prevent overfitting, the early stopping patience parameter was established at 80 epochs. Additionally, gradient clipping was implemented with a threshold of \( 1 \times 10^{-2} \), and the optimization was conducted using the Adam optimizer.

Figure 9 illustrates the learning curves for both total and individual train/validation losses during the training process. Notably, no improvement was discerned for the total validation loss beyond the 588th epoch, leading to the termination of training at the 588th epoch. It is observed that all types of losses converge without evident signs of major overfitting.

In terms of the center and area, the with-label loss yields a lower loss function compared to the without-label scenario. This outcome aligns with expectations, as in the absence of label input the input variables are predicted variables from another neural network, potentially introducing errors. The total and individual loss functions at the 508th epoch for both training and validation are summarized in Table 2. Furthermore, the table includes predictions on the test dataset generated by the trained model. All types of losses on the test dataset remain within the same range as those observed during training and validation. This outcome substantiates the model’s generalization capability beyond the training and validation datasets.

**Figure 9.** Learning curves: total, normal, curvature (left); center (middle); area (right).

To conduct a detailed analysis of the results, the errors in the \( L_1 \) norm for each predicted property outlined in Table 2 (right) are depicted in Figure 10. The error histograms display a unimodal distribution centered around zero. Large errors (errors \( \geq 0.2 \)) occur significantly less frequently than those near zero. These large errors typically arise in configurations where the volume fractions \( \alpha \) are close to 0 or 1. In such cases, the center of the interface is positioned very near to one of the nodes that define the corners of the grid cells. This proximity can lead to inaccuracies in the interface representation, affecting the results. The median of the errors is also presented in Table 2.
In addition, Figure 11 displays the $R^2$ score between the predictions and their corresponding labels in the test dataset for the scalar variables, namely, the curvature and the area. The scattered points are color-coded based on the volume fraction $\alpha$ of the cell where the prediction was made. The $R^2$ value exceeds 0.9 for both variables. Interestingly, the value of $\alpha$ does not appear to exhibit a clear correlation with the error; high and low $\alpha$ values are distributed uniformly in the figure, irrespective of whether the prediction underestimates or overestimates the label.

Table 2. Model performance at the 508th epoch in training (top) and on the test set (bottom).

| Type of Loss  | Training  | Validation | MSE       |
|--------------|-----------|------------|-----------|
| Loss$_{total}$ | $7.50 \times 10^{-3}$ | $9.07 \times 10^{-3}$ | $9.30 \times 10^{-3}$ |
| Loss$_{normal}$ | $2.30 \times 10^{-3}$ | $2.39 \times 10^{-3}$ | $2.41 \times 10^{-3}$ |
| Loss$_{curv}$ | $4.13 \times 10^{-3}$ | $5.26 \times 10^{-3}$ | $5.40 \times 10^{-3}$ |
| Loss$_{center}$ | $1.10 \times 10^{-4}$ | $2.01 \times 10^{-4}$ | $1.84 \times 10^{-4}$ |
| Loss$_{vol}$ | $6.66 \times 10^{-4}$ | $9.76 \times 10^{-4}$ | $1.05 \times 10^{-3}$ |
| Loss$_{area}$ | $1.03 \times 10^{-4}$ | $5.53 \times 10^{-5}$ | $5.51 \times 10^{-5}$ |
| Loss$_{wol}$ | $1.99 \times 10^{-4}$ | $1.95 \times 10^{-4}$ | $2.03 \times 10^{-4}$ |

| Predicted Variable | Median |
|---------------------|--------|
| $n_{error} = |n_{label} - n_{pred}|$ | $5.61 \times 10^{-2}$ |
| $H_{error} = |H_{label} - H_{pred}|$ | $4.16 \times 10^{-2}$ |
| $M_{error} = |M_{label} - M_{pred}|$ | $1.98 \times 10^{-2}$ |
| $A_{error} = |A_{label} - A_{pred}|$ | $7.57 \times 10^{-3}$ |

Figure 10. Histogram of the errors: normal, curvature, center, and area.

Figure 11. $R^2$ score on the prediction: curvature (left) and area (right).

5.2. Hyperparameter Optimization

The complex architecture employed in our study is characterized by numerous hyperparameters, and optimizing these parameters is crucial for effective training; to facilitate this process, we utilized an automated tuning tool called Optuna Akiba et al. [19], which is a free library based on Bayesian optimization. The hyperparameter study was conducted independently for the three neural networks illustrated in Figure 12.
In Optuna, optimal hyperparameters are determined through a series of “trials”, each representing a training iteration with a specific set of hyperparameters chosen from suggested values, as detailed in Tables 3–5. Each trial’s effectiveness is gauged by the final validation loss, which serves as its score. Hyperparameters for subsequent trials are automatically selected based on these scores using a Bayesian optimization approach. In our study, we conducted 100 trials, with each trial capped at a maximum of 50 epochs. This cap represented a practical balance between optimizing performance and efficiently managing computational resources. The tables further highlight the parameters that achieved the lowest validation loss.

For NN-1, the node numbers for the first two SAGEConv layers were set to $2^s_1$ and $2^s_2$, with $2^s_2$ maintained in the subsequent $n_1$ linear layers to preserve the skip connection. Another $n_2$ linear layers followed with node numbers $2^{l_2(i)}$, where $i = 1, 2, \ldots, n_2$; here, $2^{l_2(i)}$ could vary at each layer $i$ under the condition $l_2(i) \geq l_2(i+1)$. After a global max pooling operation, an additional $n_3$ linear layers followed with node numbers $2^{l_3(j)}$ ($j = 1, 2, \ldots, n_3$ with $l_3(j) \geq l_3(j+1)$). Following a divergence in the latent space into normal and curvature predictions, the curvature side included another block of layers, beginning with a linear layer with node number $2^{l_3(n)} + 3$ and succeeded by $n_4$ layers with node numbers $2^{l_4(k)}$ ($k = 1, 2, \ldots, n_4$ with $l_4(k) \geq l_4(k+1)$).

![Schematic diagram of the architecture for the hyperparameter study by Optuna](image)

**Figure 12.** Schematic diagram of the architecture for the hyperparameter study by Optuna (to avoid using a small font, the power of two is written as $2^x$): (a) normal and curvature; (b) center; (c) area.
Fluids 2025, 10, 20

Table 3. Hyperparameter study for NN-1.

| Hyperparameters | Suggested Values | Best Parameter |
|-----------------|------------------|---------------|
| $s_1$           | $\{5, 6, 7, 8\}$ | 6             |
| $s_2$           | $\{s_1, s_1 + 1, \ldots, 8\}$ | 6             |
| $l_2(i)$        | $\{5, 6, \ldots, s_2\} \land (l_2(i) \geq l_2(i+1))$ | $l_2(1) = 6$ |
| $l_3(i)$        | $\{5, 6, \ldots, l_2(n_2)\} \land (l_3(i) \geq l_3(i+1))$ | $l_3(1) = 6$ |
| $l_4(k)$        | $\{5, 6, \ldots, l_3(n_3)\} \land (l_4(k) \geq l_4(k+1))$ | $l_4(k) = 6$ for $k = 1, 2, 3$ |
| $n_1$           | $\{0, 1, 2, 3, 4\}$ | 1             |
| $n_2$           | $\{0, 1\}$       | 1             |
| $n_3$           | $\{0, 1, 2, 3, 4\}$ | 1             |
| $n_4$           | $\{1, 2, 3, 4\}$ | 3             |
| $n_b$           | $\{8, 9, 10, 11, 12, 13\}$ | 9             |

For NN-2, the constant node number $2^h$ was established for the first layer and the subsequent $n_1$ linear layers preceding the dropout layer. Following the dropout operation, an additional block of $n_2$ linear layers was introduced, where the node number $2^{h(i)}$ could vary at each layer under the condition $l_2(i) \geq l_2(i+1)$.

Table 4. Hyperparameter study for NN-2.

| Hyperparameters | Suggested Values | Best Parameter |
|-----------------|------------------|---------------|
| $l_1$           | $\{7, 8, 9, 10, 11\}$ | 11            |
| $l_2(i)$ ($i = 1, 2, \ldots, n_2$) | $\{4, 5, \ldots, l_1\} \land (l_2(i) \geq l_2(i+1))$ | $l_2(1) = 5, l_2(2) = 5$ |
| $n_1$           | $\{1, 2, 3\}$   | 3             |
| $n_2$           | $\{0, 1, 2\}$   | 2             |
| $n_b$           | $\{8, 9, 10, 11, 12\}$ | 8             |

For NN-3, the constant node number $2^h$ was assigned to the first layer and the subsequent $n_1$ linear layers, followed another block of $n_2$ linear layers, where the node number $2^{h(i)}$ could vary at each layer under the condition $l_2(i) \geq l_2(i+1)$.

Table 5. Hyperparameter study for NN-3.

| Hyperparameters | Suggested Values | Best Parameter |
|-----------------|------------------|---------------|
| $l_1$           | $\{5, 6, 7, 8, 9\}$ | 7             |
| $l_2(i)$        | $\{4, 5, \ldots, l_2(i)\} \land (l_2(i) \geq l_2(i+1)), l_2(0) = l_1$ | $l_2(1, 2, 3, 4) = 6$ |
| $n_1$           | $\{1, 2, 3, 4\}$ | 1             |
| $n_2$           | $\{1, 2, 3, 4\}$ | 4             |
| $n_b$           | $\{8, 9, 10, 11\}$ | 9             |

The impact of batch size, denoted as $N_b = 2^n$, was investigated for all three neural networks; the results indicated $n_b = 9$ to be the optimal parameter for two of the networks. Consequently, a batch size of $N_b = 512$ was selected for training.

Considering all the optimizations discussed above, the architecture of our model was finalized as presented in Section 3.1.2.

5.3. Data Size

The size of the training dataset significantly influences the quality of the model. Generally, a larger dataset tends to result in a better model, albeit at the expense of longer training times. From an engineering standpoint, understanding the tradeoff between model quality and training time is crucial. In this study, we investigated the impact of dataset size on training by examining the results of two generalization tests. The predictive
performance comparison across all variables in the two test cases is depicted in Figure 13a,b. The use of colors in these figures aids in swiftly identifying discrepancies in errors among different scenarios. The training was conducted using 50, 100, 200, 300, 400, 500, 600, and 700 thousand graphs. In both tests, we observe the expected downward trend in error with an increase in the number of graphs. However, the reduction in losses becomes less pronounced for datasets larger than 500,000 graphs, especially in test-No.2.

Table 6 provides a summary of training time versus dataset size. As anticipated, larger datasets result in longer training times. Considering the tradeoff between time and accuracy, the model was ultimately trained with a dataset size of 500,000 graphs.

Table 6. Training times of different data sizes.

| Data size (in thousand) | 50   | 100  | 200  | 300  | 400  | 500  | 600  | 700  |
|------------------------|------|------|------|------|------|------|------|------|
| Time (mins.)           | 9.50×10^1 | 1.25×10^2 | 2.79×10^2 | 4.90×10^2 | 7.71×10^2 | 1.03×10^3 | 1.23×10^3 | 1.41×10^3 |

Figure 13. Data size and prediction performance: (a) MSE of test-No.1 and (b) median of the error in test-No.2.

6. Model Validation

6.1. Generalization Test on a Sinusoidal Surface

To evaluate the generalization of the model, we conducted predictions on a 3D sinusoidal surface, as illustrated in Figure 14. The surface was defined by \( f(x) = 0.1 \sin 9x \cos 9y \). The surrounding space was voxelized (generating a tetrahedral mesh) with a constant discretization of \( \Delta = 0.05 \). Graphs were then generated within each of these voxels, with each voxel containing the center point of a graph, and its neighboring voxels containing other nodes of the same graph. Therefore, a voxel can act as both a container for a center point and a neighbor point.

The maximum absolute nondimensionalized curvature of the test surface was \( |H|_{\text{max}} = 0.703 \), falling within the range of the training dataset (\( H_{\text{training}} \approx [-0.75, 0.75] \)). With the volume fraction in the range \( 0.01 < \alpha < 0.99 \), each interface cell was then input to the model, while interface cells with very high or low volume fractions were filtered out, as we considered these to be outside the range of applicability for our model. Figure 15 displays the errors of the prediction for the four variables and their corresponding histograms. The errors for each variable are defined in Table 7. It is observed that errors do not exhibit a clear correlation with the geometry for any variable. The histograms display a unimodal shape, and no abnormal behavior is observed. Table 7 also presents the median error values.
Figure 14. Test surface colored by the mean curvature (left) and test surface voxelized and colored by $\alpha$ (right).

Figure 15. Error plotted on the test surface (left and middle) and its histogram (right) of $n_{\text{error}}$, $H_{\text{error}}$, $M_{\text{error}}$, $A_{\text{error}}$ (top to bottom); the visualized range in the 3D surface is indicated in Table 7.

The test surface utilized in this study exhibited a nonuniform distribution about $\alpha$, as depicted in Figure 16. A higher concentration of samples is observed at $\alpha$ values close to 0 and 1, referred to as “marginal $\alpha$”. This uneven distribution may impact the prediction results. In Figure 16, the prediction is compared to the label and colored by $\alpha$. Notably, cases with marginal $\alpha$ values exhibit larger errors. For instances with small marginal $\alpha$, the model tends to predict values smaller than the label, whereas it tends to predict larger
values for instances with large marginal $\alpha$. This influence of $\alpha$ was not observed in the test case during training, where $\alpha$ was quasi-uniform. This suggests limited capacity of the model for marginal $\alpha$, and should be taken into consideration when implementing the model in a real CFD solver and applying it to real-world use cases.

Figure 16. Histogram of the volume fraction $\alpha$ on the test surface and $R^2$ score on the curvature (middle) and area (right) prediction colored by $\alpha$.

Table 7. Median of the error for the entire surface.

| Predicted Variable | Median Color Scale |
|--------------------|--------------------|
| $n_{\text{error}} = |n_{\text{label}} - n_{\text{pred}}|$ | $6.05 \times 10^{-2}$ | $[\min(n_{\text{error}}), \max(0.3n_{\text{error}})]$ |
| $H_{\text{error}} = |H_{\text{label}} - H_{\text{pred}}|$ | $4.32 \times 10^{-2}$ | $[\min(H_{\text{error}}), \max(0.2H_{\text{error}})]$ |
| $M_{\text{error}} = |M_{\text{label}} - M_{\text{pred}}|$ | $1.84 \times 10^{-2}$ | $[\min(M_{\text{error}}), \max(0.1M_{\text{error}})]$ |
| $A_{\text{error}} = |A_{\text{label}} - A_{\text{pred}}|$ | $6.84 \times 10^{-3}$ | $[\min(A_{\text{error}}), \max(0.2A_{\text{error}})]$ |

6.2. Validation Simulating a Two-Phase Flow Model with OpenFOAM

To validate our approach, we implemented our methodology in OpenFOAM (“Open-source Field Operation And Manipulation”), the C++ toolbox for the development of customized numerical solvers in computational fluid dynamics (CFD). We extended the IsoAdvector solver of OpenFOAM described in Roenby et al. [14] with the implemented geometrical interface reconstruction method, denoted $\text{IsoAlpha}$. To compute the geometrical features of interfaces, we introduced a new reconstruction method, denoted $\text{IsoAlphaML}$, based on the GNN model described in Section 3.

We benchmarked the two methods using a 3D study case inspired by the original 2D rising bubble study case published in 2009 by Hysing et al. [20]. This study case involves simulating a two-phase flow within a cylindrical domain. The simulation domain is discretized with a tetrahedral mesh using the open-source GMSH tool presented in Geuzaine and Remacle [21]. Three meshes, Mesh$_{1}$, Mesh$_{2}$, and Mesh$_{3}$, are built. The original mesh Mesh$_{1}$ of size $\Delta_{1} = 0.1$ is refined twice to obtain Mesh$_{2}$ of size $\Delta_{2} = \frac{\Delta_{1}}{2} = 0.05$ and Mesh$_{3}$ of size $\Delta_{3} = \frac{\Delta_{2}}{2} = 0.025$. The primary goal of this study is to assess the impact of the mesh discretization size on the finite volume schemes employed across various models. Additionally, we seek to illustrate the performance of our method with different mesh sizes. Notably, as the grid cell size diminishes, the relative curvature of interfaces tends to decrease to lower values after normalization to the reference root cell of unit size. This significant reduction may trigger the emergence of unexpected phenomena.

The two phases are modeled as Fluid1 and Fluid2 with their specific dynamical properties (density and viscosity). Initially, the entire domain is occupied by Fluid1 except for the inside of a spherical bubble occupied by Fluid2. We impose boundary conditions as described in Figure 17.

We denote the height of the domain $\Omega$ by $L$ and the radius of the initial spherical bubble by $r_{0}$. We performed different simulations with a constant advected velocity $U_{0}$ along the $z$-axis imposed within the domain and with a surface tension coefficient denoted $\sigma$. These simulations can be classified and characterized by the scaling factors $\frac{L}{r_{0}} = 2$, $\frac{L}{U_{0}}$, where $U_{g} = \sqrt{gL}$ is the gravitational velocity, the Reynolds number $Re = \frac{\rho_{1}U_{0}L}{\mu_{1}}$ describes
the ratio of inertial to viscous effects, and the Eötvös number $E_0 = \frac{\Delta \rho g L^2}{\sigma} = \frac{\Delta \rho U_s^2 L}{\sigma}$ describes the ratio of gravitational forces to surface tension effects.

The surface tension force plays a crucial role in these models, making its correct discretization necessary for achieving accurate simulation results. As seen in Section 2, computation of the surface tension force based on the continuous surface force model (2) relies on approximation of the mean curvature $M = \frac{\kappa_1 + \kappa_2}{2}$. We compared our IsoAlphaML approach, where $M$ is predicted with our GNN-based model, to the IsoAlpha method, where $M$ is computed by discretizing Equations (5) with a finite volume scheme, as explained in Section 2.

The study case was designed in three progressive test cases:

- **Test-1:** The Stationary Droplet test case, in which Fluid1 and Fluid2 have the same dynamic properties and $U_0 = 0$. Nothing should happen. This test case aims to detect residual streams that may appear due to errors in the curvature computation.
- **Test-2:** The Translating Droplet test case, in which the two fluids have the same properties. The imposed velocity is different from 0. The bubble should be translated without deformation.
- **Test-3:** The Original 3D Hysing test case, in which the properties of the fluids are different. Fluid1 represents the liquid phase, while Fluid2 represents the gas phase. No velocity is imposed, and $U_0 = 0$. The bubble is supposed to rise with the gravity effects. The bubble is deformed under the effect of surface tension at the gas–liquid interface, which depends on the interface curvature.

In Table 8, we gather the fluids properties and simulation parameters of these three test-cases.

**Table 8.** Property of Fluid1 and 2.

| Test | Fluid1 Density | Fluid1 Viscosity | Fluid2 Density | Fluid2 Viscosity | $\sigma$ | $U_0$ | $U_s$ | Re | $E_0$ |
|------|----------------|------------------|----------------|------------------|--------|------|------|----|------|
| Test-1 | 1000           | 10               | 1000           | 10               | 1      | 0    | 0    | 0  | 0    |
| Test-2 | 1000           | 10               | 1000           | 10               | 1      | 1    | 0    | 50 | 0    |
| Test-3 | 1000           | 10               | 1              | 0.1              | 1.96   | 0.7  | 0.7  | 35 | 125  |
We first explore the distribution of the volume fraction generated by the OpenFOAM simulations to check whether it conforms to one of the datasets used to train our GNN model. Figure 18 shows the histogram of the volume fraction $\alpha$ of the diphasic cells computed during the simulation of Test-1 on the left, while the right shows the histogram of the predicted normal error $n_{\text{error}}$ on the spherical interface regarding the expected normal values, which can be easily computed. We can observe that many samples are found in the “marginal $\alpha$” range. The distribution of the $n_{\text{error}}$ is mono-modal and its peak is located near its median. These tendencies are similar to those observed in the previous generalization sinusoidal test case.

![Figure 18. Histogram of the volume fraction $\alpha$ on the spherical surface and $n_{\text{error}}$.](image)

Figure 19 compares the evolution of bubble circularity over time for each test case when using the IsoAlphaML method and the IsoAlpha method. The similarity of the curves at first glance indicates that the simulation results are very similar. As expected, the bubble circularity does not change over time for Test-1 and Test-2. In Test-3, the evolution of bubble circularity using the two methods is very similar.

To better analyze the difference between the mean curvature computation of the two methods, we focus on the results of Test-1. In this test, the droplet should remain stationary. The mean curvature should also remain constant and equal to the reference value $M_{\text{ref}} = \frac{1}{R}$, where $R$ denotes the initial radius of the droplet. We have introduced two variants IsoAlphaML$_{M_{\text{ref}}}$ and IsoAlpha$_{M_{\text{ref}}}$ of the previous methods, setting the mean curvature to the reference value $M_{\text{ref}}$. The reference normal value of the interface in each diphasic voxel is also easy to evaluate using the voxel center and the stationary droplet center. We evaluate the errors in the curvature computation and use the fact that these errors impact the value of the surface tension force, which leads to the appearance of nonzero residual velocities. Then, computing the overall kinetic energy is a simple way to evaluate the accuracy of the curvature computation method. The errors regarding the reference solution of the two approaches IsoAlphaML and IsoAlpha are gathered in Table 9. The histogram of curvature values, the normal and curvature errors are plotted in Figures 20 and 21. Figure 22 plots the time evolution of the global residual kinetic energy for the IsoAlphaML and the IsoAlpha methods along with their variants IsoAlphaML$_{M_{\text{ref}}}$ and IsoAlpha$_{M_{\text{ref}}}$. As expected, we see that the kinetic energy of the IsoAlphaML$_{M_{\text{ref}}}$ and IsoAlpha$_{M_{\text{ref}}}$ simulations remains very low. For IsoAlphaML and IsoAlpha, the kinetic energy increases with time, highlighting the effect of the curvature error on the surface tension computations. For the simulation with Mesh$_1$ and Mesh$_2$, this effect is less important for the IsoAlphaML method than for the original IsoAlpha method, while for the simulation with Mesh$_3$ the effect is inverted.
Table 9. Median of the predicted/calcualted value.

| Variable | IsoAlphaML | IsoAlpha |
|----------|------------|----------|
| $\|\|n_{\text{label}} - n_{\text{pred}}\|\|$ | $5.94 \times 10^{-2}$ | $1.33 \times 10^{-1}$ |
| $\|\|M_{\text{label}} - M_{\text{pred}}\|\|$ | $1.64 \times 10^{-2}$ | $2.38 \times 10^{-2}$ |

Test 3.1: Stationary droplet

Test 3.2: Translating droplet

Test 3.3: 3D Hysing test case

Figure 19. Bubble circularity time evolution.

Figure 20. Test-1: IsoAlphaML vs. IsoAlpha curvature values histograms.

Figure 21. Test-1: IsoAlphaML vs. IsoAlpha curvature and normal error histograms.
The errors of our GNN-based approach are reasonable. Similar to the previous generalization sinusoidal test, the results prove the capability of our method to generalize on unseen surfaces. This makes the proposed method interesting in the context of real CFD simulations, where complex unstructured meshes are required.

Notably, our approach outperforms the IsoAlpha method on both variables.

**Figure 22.** Test-1: Residual kinetic energy time evolution.

### 7. Conclusions and Future Perspectives

In this study, we have proposed an enhanced volume of fluid (VoF) method utilizing machine learning, specifically graph neural networks (GNNs), to expedite interface reconstruction on general unstructured meshes. Our approach involves the generation of a synthetic dataset comprising paraboloid surfaces discretized on unstructured meshes. Comprehensive data exploration was conducted to understand the characteristics of the dataset. Subsequently, we introduced a GNN-based method for computing interface reconstruction on general unstructured meshes. Extensive hyperparameter optimization was performed, focusing on the parameters of our GNN architecture and the dataset size.

We validated the efficacy of our new method through generalization tests. The results demonstrate that GNNs provide a viable alternative to conventional surface reconstruction methods, proving effective even on unstructured grids. This is in contrast to other approaches, which are limited to structured meshes. Importantly, our method showcases superior accuracy compared to traditional geometrical methods on tetrahedral meshes, making it particularly appealing in industrial contexts.

Future work will focus on efficient integration of GNN models in CFD simulators such as OpenFoam. To observe significant computational gains, extensive work is required in order to couple traditional numerical simulators, which are usually parallelized on CPU cores, with ML models, which are most efficient when evaluated using GPU accelerators. This endeavor will contribute to a more comprehensive understanding and application of our proposed GNN-enhanced VoF method in practical engineering scenarios.

**Author Contributions:** Conceptualization, M.A.B., J.-M.G. and T.F.; Methodology, T.N., M.A.B., J.-M.G. and T.F.; Software, T.N., M.A.B. and J.-M.G.; Validation, M.A.B., J.-M.G. and T.F.; Formal analysis, T.N. and J.-M.G.; Investigation, T.N. and J.-M.G.; Data curation, T.N.; Writing—original draft, T.N. and J.-M.G.; Writing—review & editing, M.A.B. and T.F.; Supervision, M.A.B., J.-M.G. and T.F.; Project administration, T.F. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the DATAIA Convergence Institute as part of the “Programme d’Investissement d’Avenir” (ANR-17-CONV-0003) operated by INRIA and IFPEN.

**Data Availability Statement:** The raw data supporting the conclusions of this article will be made available by the authors on request.

**Conflicts of Interest:** The authors declare no conflicts of interest.
Appendix A

Appendix A.1. GNN Models

A graph \( G = (V, E) \) consists of a set of \( n \) nodes \( V \) and a set of edges \( E \) connecting the nodes, as illustrated in Figure 3a. We denote a node in the graph by \( v_i \) and an edge between nodes \( v_i \) and \( v_j \) by \( e_{ij} \). The graph connectivity can be represented by the adjacency matrix \( A \), which is an \( n \times n \) matrix with \( A_{ij} = 1 \) if \( e_{ij} \in E \) and \( A_{ij} = 0 \) if \( e_{ij} \notin E \). Nodes and edges can be associated with feature vectors \( x_v \in \mathbb{R}^d \) and \( x_e \in \mathbb{R}^q \), respectively.

GNNs are designed to handle data in the form of graph structures. A GNN takes a graph as its input and returns the same graph with updated features as the output. Message passing (MP) is the main mechanism that allows the nodes to communicate through edges to update the node features. The MP mechanism on a node \( i \) can be summarized in the following three steps:

1. Message computation: \( \phi_e(x_{e_{ij}}) \rightarrow \tilde{x}_{e_{ij}} \) for all nodes \( j \) in the neighbour of the \( i \) node.
2. Message propagation: \( \phi_v(\tilde{x}_{e_{ij}}, x_{v_i}) \rightarrow \tilde{x}_{v_i} \).
3. Message aggregation: \( \phi_a(\tilde{x}_{v_i}, \square_j(\tilde{x}_{e_{ij}})) \rightarrow x_{v_i} \), where \( \square_j \) is any aggregation function with permutation-invariant properties.

Here, \( \phi_v \), \( \phi_e \), and \( \phi_a \) are learnable functions, generally MLPs. The same steps are computed for all nodes in the graph in an operation similar to convolution. Many types of GNN have been proposed; for examples, see [18,22]. In our work, we adopt the SAGEConv architecture proposed by [23].

Most frameworks proposed prior to SAGEConv are transductive; such approaches directly optimize the embeddings for each node using matrix factorization-based objectives, and do not naturally generalize to unseen data, as they make predictions on nodes in a single fixed graph. On the other hand, SAGEConv is an inductive framework which learns a function that generates embeddings by sampling and aggregating features from a node’s local neighborhood. This mechanism allows SAGEConv to efficiently generate node embeddings for unseen data. SAGEConv learns the topological structure of each node’s neighborhood as well as the distribution of node features in the neighborhood. It is based on MP mechanism.

Below, we show the forward propagation algorithm of SAGEConv (Algorithm A1). The intuition behind it is that at each iteration (or search depth), nodes aggregate information from their local neighbors; iteration then allows nodes to obtain more information from further nodes. The entire graph \( G = (V, E) \) and the input features (dimension \( d \)) for all nodes \( x_v \in \mathbb{R}^d \) and \( v \in V \) are provided as input. The outer ‘for’ loop performs the iteration on \( k \), while the depth of the search and the \( h^k \) denote a node’s representation at this step. The inner ‘for’ loop iterates on nodes \( v \in V \), in which the aggregation of the representation from the nodes in its immediate neighborhood is \( h^{k-1}_u \) for \( u \in N(v) \) into \( h^k_{N(v)} \), where \( N(v) \) indicates all the neighboring nodes of \( v \). The aggregation step refers to the previous step \( k - 1 \). Input node features are defined as the \( k = 0 \) representations. After aggregation, the node’s current representation \( h^k_v \) and the aggregated neighbor \( h^{k-1}_{N(v)} \) are concatenated. This concatenated vector is fed through a fully connected layer with a nonlinear activation function \( \sigma \), which transforms the representations to \( h^k_{N(v)} \) for the next step \( k + 1 \). The final representations output at depth \( K \) is \( z_v \), \( \forall v \in V \).

A variety of aggregate architectures have been proposed, including mean, LSTM, and pooling aggregators. We employ mean aggregation in our work. Figure A1 shows the concept of message passing with a mean aggregator, which uses the following steps. We refer the reader to [18,22,23] for a thorough review of the topic.

1. mean \( \varepsilon_{u \in N(v)} h^{k-1}_u \): Calculate a mean of features stocked at the neighbors \( u \) of the node \( v \). This step corresponds to aggregation.
2. \( W_2 \cdot \text{mean}_{u \in N(v)} h_{u}^{k-1} \): Multiply a weight \((W_2)\) to this mean.

3. \( x_v' = W_1 h_{v}^{k-1} + W_2 \cdot \text{mean}_{u \in N(v)} h_{u}^{k-1} \): Add the transformed (weighted) features of \( v \) to the weighted mean of \( x \) and define a new feature on \( v \). This step corresponds to updating.

4. \( h_v^k = \sigma \left( W_1 h_{v}^{k-1} + W_2 \cdot \text{mean}_{u \in N(v)} h_{u}^{k-1} \right) \): Add an activation function to pass the new features to the next layer.

Algorithm A1 SAGEConv [24] embedding generation (forward propagation) algorithm.

**Input:** Graph \( G = (V, E) \); input features \( \{x_v, \forall v \in V\} \); depth \( K \); weight matrices \( W_k \), \( \forall k \in \{1, ..., K\} \); activation function \( \sigma \); differentiable aggregator functions \( \text{AGGREGATE}_k \), \( \forall k \in \{1, ..., K\} \); neighborhood function \( N : v \to 2^V \).

**Output:** Vector representations \( z_v \) for all \( v \in V \)

\[
\begin{align*}
h_0^v & \leftarrow x_v, \forall v \in V; \\
\text{for } k = 1 \ldots K \text{ do} \\
& \text{for } v \in V \text{ do} \\
& \quad h_{N(v)}^k \leftarrow \text{AGGREGATE}_k \left( \left\{ h_{u}^{k-1}, \forall u \in N(v) \right\} \right); \\
& \quad h_v^k \leftarrow \sigma \left( W_k \cdot \text{CONCAT} \left( h_{N(v)}^{k-1}, h_v^{k-1} \right) \right) \\
& \text{end for} \\
& h_v^k \leftarrow h_v^k / \left\| h_v^k \right\|_2, \forall v \in V \\
\text{end for} \\
z_v & \leftarrow h_K^v, \forall v \in V
\end{align*}
\]

Figure A1. Multilayer perceptron (MLP) (left) and message-passing around a node \( v \) (right).

We used Pytorch and Pytorch-Geometric as platforms to develop our GNN-based interface reconstruction algorithm through the Pytorch-Lightning interface. For hyperparameter optimization, we used Optuna. Originally developed by Facebook’s AI Research lab (FAIR), Pytorch [25] is an open-source deep learning framework based on Python programs. It provides Numpy-like tensor computing and GPU acceleration. Programs are built on a type-based automatic differentiation system. Its flexibility and ease of use make it one of the most popular among similar frameworks. Pytorch-Geometric [26] is a library built upon PyTorch that allows for easy writing and training of GNNs. In addition, it consists of easy-to-use minibatch loaders for operating on many small and single giant graphs, as well as multi-GPU support. Pytorch-Lightning is a lightweight wrapper (high-level interface) for Pytorch and Pytorch-Geometric that simplifies usage and helps to abstract the details of training [27]. Optuna is a library for automatic hyperparameter optimization based on a Bayesian algorithm called the Tree-Structured Parzen Estimator. It can perform searches based on numerical variables such as number of layers/nodes and learning rate as well as methodological variables such as optimization method (SGD, Adam, etc.). Optuna is widely used due to its ease of implementation and user-friendly interface.

Appendix A.2. Dataset Generation Algorithm

To generate the dataset based on paraboloidal surfaces, we employed an algorithm that relies on the maximum curvature value \( K_{\text{max}} \) of the surface samples. This algorithm defines \( \kappa \) within the range \([-K_{\text{max}}, \infty[\) and \( u \sim U[-1,1) \) and \( v \sim U[0,1) \). The procedure is as follows:
1. Select $\kappa_1, \kappa_2$ from $k$ and $u_1, u_2, u_3, u_4$ from $u$ and $v_1$ from $v$.
2. Generate the paraboloidal surface defined by Equation (6) with $\kappa_1, \kappa_2$. This surface will represent the interface between two fluids.
3. Define the translation $T(u_1, u_2, u_3)$ and rotation $R(\cos^{-1}(2v_1 - 1), \pi v_1, 2\pi v_1)$.
4. Form the cubic domain $(x\mathbf{C}, L)$ around this interface, where $x\mathbf{C} = (u_1, u_2, u_3)$ and $L = 12 + u_4$ represent the center and dimension $L$ of the cube, respectively.
5. Generate a tetrahedral mesh with a constant grid resolution $\Delta = 1$, discretizing the domain using the open-source GMSH tool [21].
6. Compute the intersection between the interface and grid cell $C_0$.
7. Randomly rotate the graphs by $(\text{rot}_x, \text{rot}_y, \text{rot}_z) = (\cos^{-1}(2v_1 - 1), \pi v_1, 2\pi v_1)$ to ensure a uniform distribution of 3D rotations.
8. Collect the values of the normal, curvature, center, and area of this intersection as the label values for the training process.
9. Collect the graph data as features on the graph nodes, including the coordinates of the vertices (center of the neighboring tetrahedral cells), edges (connectivity between cells), and volume fraction of the neighboring tetrahedral cells.

These steps are repeated $N_{\text{all}}$ times to generate a dataset with $N_{\text{all}}$ data samples.

References

1. Kothe, D.; Rider, W.; Mosso, S.; Brock, J.; Hochstein, J. Volume tracking of interfaces having surface tension in two and three dimensions. In Proceedings of the 34th Aerospace Sciences Meeting and Exhibit, Reno, NV, USA, 15–18 January 1996; p. 859.
2. Lörstad, D.; Francois, M.; Shyy, W.; Fuchs, L. Assessment of volume of fluid and immersed boundary methods for droplet computations. Int. J. Numer. Methods Fluids 2004, 46, 109–125. [CrossRef]
3. Harvie, D.; Davidson, M.; Rudman, M. An analysis of parasitic current generation in Volume of Fluid simulations. Appl. Math. Model. 2006, 30, 1056–1066. [CrossRef]
4. Qi, Y.; Lu, J.; Scardovelli, R.; Zaleski, S.; Tryggvason, G. Computing curvature for volume of fluid methods using machine learning. J. Comput. Phys. 2019, 377, 155–161. [CrossRef]
5. Patel, H.; Panda, A.; Kuipers, J.; Peters, E. Computing interface curvature from volume fractions: A machine learning approach. Comput. Fluids 2019, 193, 104263. [CrossRef]
6. Svyetlichnyy, D. Neural networks for determining the vector normal to the surface in CFD, LBM and CA applications. Int. J. Numer. Methods Heat Fluid Flow 2018, 28, 1754–1773. [CrossRef]
7. Owen, S.J. A survey of unstructured mesh generation technology. IMR 1998, 239, 15.
8. Katz, A.; Sankaran, V. Mesh quality effects on the accuracy of CFD solutions on unstructured meshes. J. Comput. Phys. 2011, 230, 7670–7686. [CrossRef]
9. Ngo, S.I.; Lim, Y.I. Multiscale Eulerian CFD of chemical processes: A review. ChemEngineering 2020, 4, 23. [CrossRef]
10. Abidi, A.; Ahmadi, A.; Enayati, M.; Sajjadi, S.M.; Yarmand, H.; Ahmed, A.; Cheraghian, G. A review of the methods of modeling multi-phase flows within different microchannels shapes and their applications. Micromachines 2021, 12, 1113. [CrossRef]
11. Bucci, M.; Gratien, J.; Faney, T.; Nakano, T.; Charpiat, G. Using Graph Neural Network for gas-liquid interface reconstruction in Volume of Fluid methods. In Proceedings of the 8th European Congress on Computational Methods in Applied Sciences and Engineering, Oslo, Norway, 5–9 June 2022. [CrossRef]
12. Devolder, B.; Schmitt, P.; Rauwoen, P.; Elsaesser, B.; Troch, P. A review of the implicit motion solver algorithm in OpenFOAM® to simulate a heaving buoy. In Proceedings of the NUTTS Conference, Marstrand, Sweden, 28–30 September 2015; Volume 2015, p. 18.
13. Deshpande, S.S.; Anumolu, L.; Trujillo, M.F. Evaluating the performance of the two-phase flow solver interFoam. Comput. Sci. Discov. 2012, 5, 014016. [CrossRef]
14. Roenby, J.; Bredmose, H.; Jasak, H. A computational method for sharp interface advection. R. Soc. Open Sci. 2016, 3, 160405, [CrossRef] [PubMed]
15. Williams, M.W.; Kothe, D.; Korzekwa, D.; Tubesing, P. Numerical methods for tracking interfaces with surface tension in 3-D mold filling processes. In Proceedings of the Fluids Engineering Division Summer Meeting, Montreal, QC, Canada, 14–18 July 2002; Volume 36150, pp. 751–759.
16. Williams, M.; Kothe, D.; Puckett, E. Accuracy and convergence of continuum surface tension models. *Fluid Dyn. Interfaces* 1998, 294–305.
17. Cummins, S.J.; Francois, M.M.; Kothe, D.B. Estimating curvature from volume fractions. *Comput. Struct.* 2005, 83, 425–434. [CrossRef]
18. Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; Sun, M. Graph neural networks: A review of methods and applications. *AI Open* 2020, 1, 57–81. [CrossRef]
19. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A Next-generation Hyperparameter Optimization Framework. *arXiv* 2019, arXiv:1907.10902. [CrossRef]
20. Hysing, S.R.; Turek, S.; Kuzmin, D.; Parolini, N.; Burman, E.; Ganesan, S.; Tobiska, L. Quantitative benchmark computations of two-dimensional bubble dynamics. *Int. J. Numer. Methods Fluids* 2009, 60, 1259–1288. [CrossRef]
21. Geuzaine, C.; Remacle, J.F. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *Int. J. Numer. Methods Eng.* 2009, 79, 1309–1331. [CrossRef]
22. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Philip, S.Y. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 2020, 32, 4–24. [CrossRef] [PubMed]
23. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. *Adv. Neural Inf. Process. Syst.* 2017, 30. Available online: https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html (accessed on 28 October 2024).
24. PyTorch-Geometric. SAGEConv. Available online: https://pytorch-geometric.readthedocs.io/en/2.6.0/generated/torch_geometric.nn.conv.SAGEConv.html (accessed on 28 October 2024).
25. PyTorch. PyTorch. Available online: https://pytorch.org/ (accessed on 28 October 2024).
26. PyTorch-Geometric. PyTorch Geometric. Available online: https://pytorch-geometric.readthedocs.io/en/latest/ (accessed on 28 October 2024).
27. PyTorch-Lightning. PyTorch Lightning. Available online: https://www.pytorchlightning.ai/ (accessed on 28 October 2024).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.