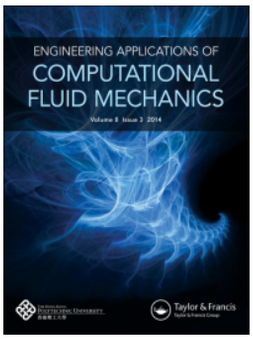


Engineering Applications of Computational Fluid Mechanics

ISSN: (Print) (Online) Journal homepage: www.tandfonline.com/journals/tcfm20

Efficient and accurate machine learning models for volume of fluid (VOF) simulation on uniform Cartesian mesh

Mostafa A. Rushdi, Shigeo Yoshida, Changhong Hu, Tarek N. Dief, Abdulrahman E. Salem & Mohamed M. Kamra

To cite this article: Mostafa A. Rushdi, Shigeo Yoshida, Changhong Hu, Tarek N. Dief, Abdulrahman E. Salem & Mohamed M. Kamra (2025) Efficient and accurate machine learning models for volume of fluid (VOF) simulation on uniform Cartesian mesh, Engineering Applications of Computational Fluid Mechanics, 19:1, 2451774, DOI: 10.1080/19942060.2025.2451774

To link to this article: https://doi.org/10.1080/19942060.2025.2451774

Published online: 16 Jan 2025.

ENGINEERING APPLICATIONS OF COMPUTATIONAL FLUID MECHANICS
2025, VOL. 19, NO. 1, 2451774
https://doi.org/10.1080/19942060.2025.2451774

Efficient and accurate machine learning models for volume of fluid (VOF) simulation on uniform Cartesian mesh

Mostafa A. Rushdi[a], Shigeo Yoshida[a,b], Changhong Hu[a], Tarek N. Dief[c], Abdulrahman E. Salem[c] and Mohamed M. Kamra[c]

[a]Research Institute for Applied Mechanics (RIAM), Kyushu University, Fukuoka, Japan; [b]Institute of Ocean Energy (IOES), Saga University, Saga, Honjo-machi, Japan; [c]College of Engineering, UAE University, Al-Ain, United Arab Emirates

ABSTRACT

In this paper, we describe a machine learning (ML) approach for estimating interface orientation in multiphase flow using the volume of fluid (VOF) method on a uniform Cartesian mesh. By using complex shapes generated with the parametric radial star formula during training, we significantly improve prediction accuracy without increasing the network's structural complexity or processing cost. Our key contribution is the development of a robust ML model capable of reliably predicting interface orientation angles on uniform Cartesian grids. To enhance performance and robustness, we conducted a parametric study of the ML models' hyperparameters. The proposed method produced two models: a full augmented (9-cell) stencil and a compact (5-cell) stencil. Both were compared against popular finite difference/volume methods commonly used in VOF schemes. The results show that our approach is more accurate while remaining computationally efficient, particularly when employing a small stencil. Numerical studies, including challenging flow scenarios, demonstrate that the technique can predict interface orientation with an absolute mean error of less than 1 degree. Implemented in the OpenFOAM isoAdvector, our technique reliably produces accurate interface tracking with minimal deviation from the exact solution. These findings highlight the potential for incorporating machine learning approaches into classical numerical methods to improve the accuracy and reliability of the VOF method in a variety of challenging applications using uniform Cartesian meshes.

ARTICLE HISTORY
Received 25 September 2024
Accepted 21 December 2024

KEYWORDS
Machine learning; volume of fluid (VOF) method; interface orientation; multiphase flow; Cartesian mesh

1. Introduction

1.1. ML/DL applications with CFD

Deep learning is a rapidly growing field of artificial intelligence that has revolutionized many areas of research and industry. It involves training multiple-layered artificial neural networks to learn from input data and produce predictions or classifications. With recent increases in computer processing power and the availability of vast amounts of data, deep learning has become a potent tool for addressing complicated issues in a variety of disciplines.

In the field of computational fluid dynamics (CFD), specifically in the context of interface capturing with volume-of-fluid techniques, deep learning has demonstrated tremendous potential. Interface capture is the process of correctly capturing/tracking the position of a fluid interface separating two immiscible fluids in a multiphase CFD simulation. Commonly, volume of fluid (VOF) techniques are used to model the fluid interface as a sharp discontinuity between the two fluids. However, these approaches can be computationally expensive and require a substantial amount of memory to hold VOF data.

Recent research has revealed that deep learning has the ability to enhance the precision and performance of VOF algorithms. Without the need for sophisticated numerical approaches, it is possible to construct models that can accurately predict the position and behaviour of fluid interfaces by training neural networks on massive datasets of simulated fluid flow data. These models can greatly lower the processing and complexity requirements of VOF algorithms, making them more efficient for practical engineering applications (Ataei, Pirmorad, et al., 2021; Després & Jourdren, 2020; Önder & Liu, 2022; Raissi et al., 2019).

1.2. Problem statement

For multi-phase fluid flow simulations, the volume of fluid methodology (Chorin, 1980; DeBar, 1974) is a well-known interface-tracking technique in which the follow-

CONTACT Mohamed M. Kamra mohamed.kamra@uaeu.ac.ae College of Engineering, UAE University, P.O. Box 15551, Al-Ain, United Arab Emirates

ing transport equation of volume fraction is solved to model interfacial dynamic phenomena:

$$\frac{\partial \phi}{\partial t} + \boldsymbol{v} \cdot \nabla \phi = 0 \tag{1}$$

where ϕ represents the volume fraction ranging from 0.0 to 1.0 and v is the velocity vector. Each computational cell's volume fraction specifies the ratio of the heavier phase, therefore when ϕ is between zero and unity, an interface is present in that cell.

Interfacial flow simulations in science and industry frequently employ the volume of fluid method. Popular approaches include PLIC (Harvie & Fletcher, 2000; Jofre et al., 2014; Scheufler & Roenby, 2019), Youngs (Youngs, 1982), LVIRA (Puckett, 1991), and ELVIRA (López et al., 2004; López et al., 2008; Pilliod & Puckett, 2004). In the PLIC method, the interface orientation angle is fundamental for defining the interface normal vector, which is used to reconstruct the linear interface segment within each computational cell. The accuracy of the PLIC method relies on the precise calculation of this vector, as it determines the interface's geometry and position.

The interface normal vector, $\mathbf{n}$, is obtained by normalizing the gradient of the volume fraction field, ϕ, and is defined as:

$$\mathbf{n} = \frac{\nabla \phi}{\|\nabla \phi\|}, \tag{2}$$

where the gradient of ϕ is represented as:

$$\nabla \phi = \begin{bmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial y} \\ \frac{\partial \phi}{\partial z} \end{bmatrix}. \tag{3}$$

In two dimensions, the interface orientation angle, θ, which represents the direction of the normal vector, can be calculated as:

$$\theta = \arctan\left(\frac{n_y}{n_x}\right),$$

where n_x and n_y are the x- and y-components of the interface normal vector. In this study, the terms 'interface normal' and 'interface orientation' are used interchangeably, as they describe the same geometric property of the interface and serve complementary roles in interface reconstruction and evolution.

The computation of the interface orientation angle is often performed using finite difference formulas or least-squares techniques with large stencil sizes (Ito et al., 2014). While these methods are well-established, they are prone to numerical errors, particularly near sharp interfaces.

In this work, we explore an alternative, data-driven approach using machine learning (ML) and deep learning (DL) techniques to predict the interface orientation based on local volume fraction inputs. As shown in Figure 1, this approach achieves lower mean orientation angle errors compared to traditional methods, thereby improving the accuracy and robustness of the VOF method for multiphase flow simulations.

ML approaches have their own concepts and methods because they are not based on solving mathematical formulae (associated with physical model) but rather on the creation of massive datasets and the algorithmic learning of key features contained in these datasets. We believe that interface reconstruction and VOF processes fit well within the ML paradigm because this class of approaches has been shown to be useful for image recognition and object detection and tracking. It should also be highlighted that one-sided differencing problems have been reported to affect the volume of fluid method's accuracy close to boundaries (Chen et al., 2012; Kothe & Mjolsness, 1992; Park et al., 2009), demanding future development to enhance predictions of wall adhesion problems.

Performance of the developed solvers is not extensively compared in this work to the literature; nonetheless, a quantitative comparison to standard finite difference/volume techniques is given here, with further information and analysis to be given in a later publication. We will focus on this new paradigm of approaches' viability in the subsequent sections.

The machine learning algorithms that we use fall into the category of supervised machine learning (Goodfellow et al., 2016). In more mathematical terms, it equates to the creation of an approximation or regression of a function specified by a set of given points (representing ground truth); the collection of these points is the dataset. Building a specific dataset (or datasets) and building the interpolating function are the two key tasks (this is called the training session). The effectiveness of both processes determines how well the outcomes turn out. In this study, we emphasize the significance of selecting the appropriate dataset in order to achieve reliable predictions, even when using a simple neural network architecture with few layers.

1.3. Related work/attempts

Ataei, Bussmann, et al. (2021) have presented a machine learning approach to perform Piece-wise Linear Interface Construction on square, cubic, and arbitrarily-shaped triangular and tetrahedral meshes. Each mesh type was

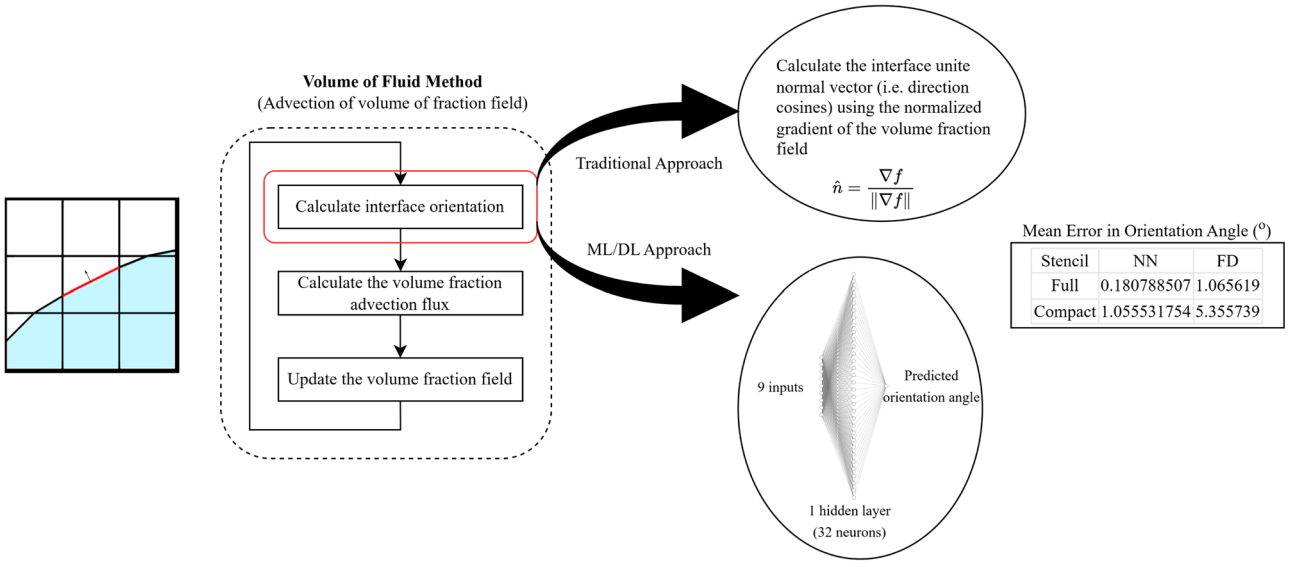


Figure 1. Problem statement and summary of the main findings.

normalized to reduce the number of inputs to the neural network. The authors have shown that their neural networks are capable of performing up to 100 times faster than available PLIC algorithms with minimal loss of accuracy.

Qi et al. (2019) aimed to find a relationship between volume fractions and curvature in VOF methods using machine learning. It was found that this approach can capture the curvature of a more complex interface with a spatially varying curvature reasonably well. The authors used a dataset based on one simple shape, which are circles of variable sizes. Their results suggest that using a neural network (with a 1-hidden layer of 100 neurons) to generate the relationship is a viable approach that results in reasonably accurate predictions. Although, the efficiency of this method compared to more traditional ones has not been tested, the study highlighted that a specific grid convergence order was not observed in the computed cases.

Compared to the aforementioned research work, our study considered a wider variety of shapes that enrich the dataset. Then, performing intensive hyperparameter tuning for the neural network leads to higher accuracy. Finally, tackling both full and compact stencils and achieving high accuracy for both types is promising and should have an impact.

1.4. Contribution and organization

In this paper, we explore the application of a deep learning approach to predict or compute the interface orientation angle within computational cells that contain an interface separating two immiscible fluids. The dataset was compiled using various well-defined shapes where the interface orientation angle and volume fractions are readily available. Our findings demonstrate the superiority of the tuned neural network over traditional finite volume techniques in predicting the interface orientation angle (i.e. interface unit normal vector) for both full and compact computational stencils, which are described in Section 2.

While the present approach (Figure 1) is currently implemented on 2D Cartesian grids, it is by no means limited to such frameworks. The methodology can be extended to unstructured meshes – commonly employed in engineering for their adaptability to complex geometries – by using an overlaying Cartesian grid as a structured framework. This overlaying grid could potentially bridge the irregularity of unstructured meshes, simplifying feature extraction and implementation. Similar strategies have been successfully utilized in Overset mesh techniques, which employ overlapping grids to handle complex or moving geometries (Zafeiris & Papadakis, 2024), and in methods like coupled level-set VOF (Zhao & Chen, 2017) and height function approaches (Ivey & Moin, 2015; Owkes & Desjardins, 2015), where structured grids enable accurate interface tracking and reconstruction.

The paper is structured into six primary sections. Section 2 details how the data was generated and the types of shapes included in our dataset. Section 3 outlines the methodology used to assess the quality of the model. In Section 4, we report the comprehensive results from deep learning techniques, including NN (neural networks) and CNN (convolutional neural networks). Section 5 is devoted to discussing the DL results in comparison with traditional finite volume methods. Section 6 presents the benchmark problems' outcomes. Finally,

Section 7 summarizes the paper and suggests avenues for future research. Appendices are provided as follows: Appendix 2 provides a brief overview of machine learning methods suitable for our problem; it also includes illustrations of NN and CNN architectures, along with our hyper-parameter tuning approach.

2. Data generation and analysis

The interface and the volume fraction of each cell are depicted in Figure 2. Despite the fact that the interface is depicted in the figure as a line, the only data used in calculations are the volume fractions, or ϕ_P, of the cell under consideration and its neighbouring cells. For simplicity, we describe two-dimensional flow here; however, the method can be applied to three-dimensional flow as well. We aim to establish a functional relationship between the volume fraction values of the interface cell and the cells surrounding it to determine the orientation angle or unit vector of the interface. We consider two approaches:

- The first relates the interface cell, P, with its vertex neighbours (neighboring cells sharing a vertex with the interface cell), as shown in Figure 2(left),

$$\theta = f\left(\begin{bmatrix} \phi_{\mathrm{NW}} & \phi_N & \phi_{\mathrm{NE}} \\ \phi_W & \phi_P & \phi_E \\ \phi_{\mathrm{SW}} & \phi_S & \phi_{\mathrm{SE}} \end{bmatrix}\right) \tag{4}$$

  relating the interface orientation to the nine volume fractions in and around the P cell.
- The second relates the interface cell, P, with its immediate face neighbours (neighboring cells sharing a face/boundary with the interface cell), as shown in Figure 2(left),

$$\theta = f\left(\begin{bmatrix} \bullet & \phi_N & \bullet \\ \phi_W & \phi_P & \phi_E \\ \bullet & \phi_S & \bullet \end{bmatrix}\right) \tag{5}$$

  relating the interface orientation to the five volume fractions in and around the P cell.

Since the interface orientation is a periodic angle with $\theta \in [0, 2\pi)$, we opted for representing it using the components $(\cos\theta, \sin\theta)$ which have values between $[-1.0, 1.0]$. This naturally takes periodicity into account while maintaining outputs that have bounded values.

2.1. Generated shapes

Finding the functional relationship using machine learning involves three processes:

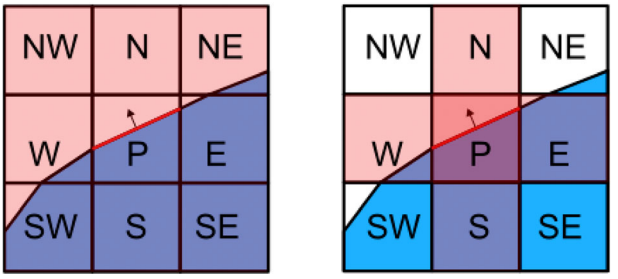


Figure 2. Illustration of the computational stencil used in the computation of the interface orientation (i.e. unit normal vector). Left: vertex neighbour stencil, Right: face neighbour stencil.

Table 1. Listing of the values used with equation (6) to generate the training dataset.

Parameter	Minimum value	M aximum value
r_0	0.25	0.45
a	0.2	0.4
b	0.0	15.0
c	2.0	4.0
θ_0	0.0	60.0

- Creating a training dataset comprised of a large number of mathematically defined complicated shapes across a wide range of interface orientations with respect to the grid.
- The data is fitted using Neural Networks (the training step) to determine the interface orientation (i.e. direction components) as a function of volume fractions.
- Testing the trained model with a second dataset of comparably complicated shapes that were not used for training.

We use the following formula, referred to as the parametric radial star, to build a dataset containing the interface orientation angles, their components, and volume fractions for well-defined geometric shapes:

$$r = \left(r_0 + a\sin\left(\frac{b(\theta - \theta_0)}{2}\right)^c\right) \tag{6}$$

where r_0, a, b, c and θ_0 are arbitrary constants that can be adjusted to generate a variety of desired shapes. The range of values used to generate the shapes in this work is provided in Table 1.

The volume fraction field is computed using the procedure detailed in Appendix 1. The generated dataset includes hundreds of shapes with varying levels of intricacy and complexity, defined by the parametric radial star formula. A sample of six representative shapes is shown in Figure 3 for demonstration purposes, with each shape characterized by the parameters $\{r_0, a, b, c, \theta_0\}$. Unlike prior approaches that relied on overly simplified shapes such as circles, this method introduces a higher degree of complexity, enabling the development of a more robust

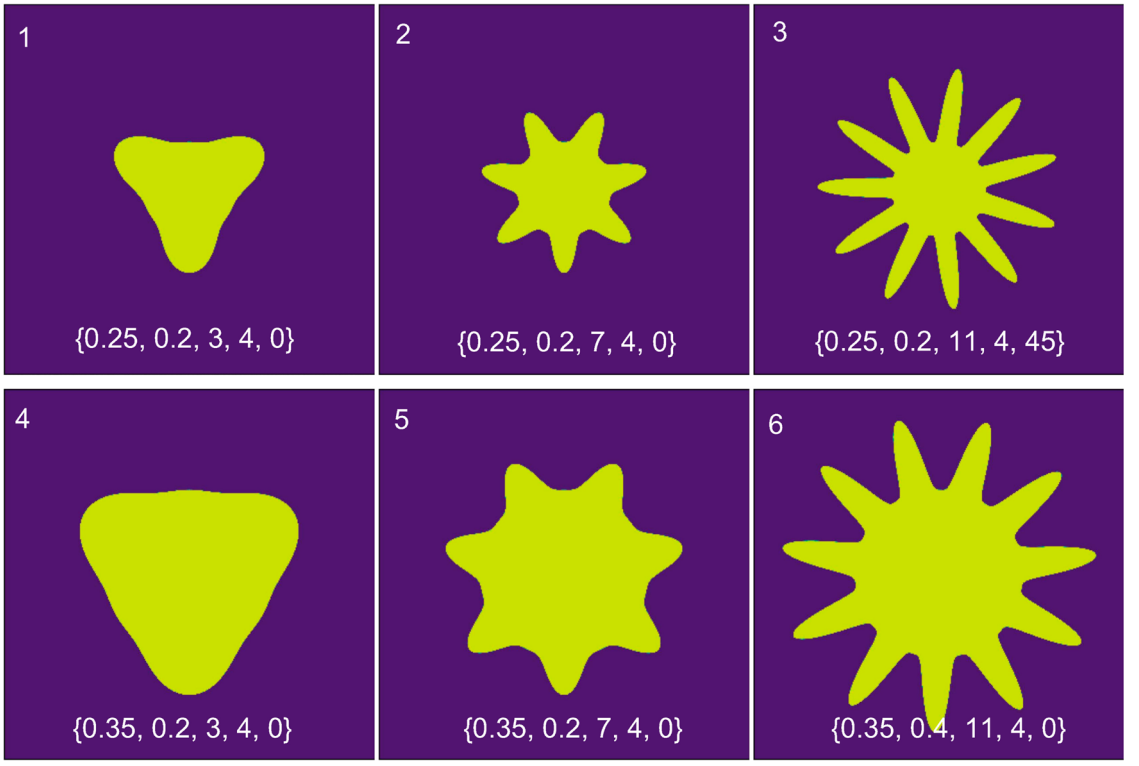


Figure 3. Illustration of sample interface shapes from the training dataset, each labelled with the parameters $\{r_0, a, b, c, \theta_0\}$.

and adaptable model for practical engineering applications. The parametric radial star formula further facilitates the analytical computation of interface orientation angles and volume fractions, providing accurate ground truth data for training the machine learning model.

2.2. Data description and statistics

This subsection details the features used in the deep learning model and presents statistical information about the data. The primary inputs to the model are the volume fraction (ϕ) within each cell, which includes two sets of features for testing and prediction: the vortex neighbour full stencil and the face neighbour compact stencil. The predicted output is the orientation angle (θ).

Table 2 provides a summary of the statistical properties of the data features. It includes the mean value, standard deviation (std), as well as percentiles such as 25%, 50% (median), and 75%. These statistics help in understanding the distribution and variability within each feature set. The statistics in Table 2 shows a well-balanced dataset with symmetric mean values around 0.5 and consistent median values. Standard deviations are generally uniform, with slight differences in diagonal directions, suggesting mild directional variance. The min and max values serve as boundary checks rather than insightful characteristics, while the $\cos(\theta)$ and $\sin(\theta)$ values reflect an even distribution of orientations.

3. Predictive models

In the beginning, we should clarify that according to ML tasks, which are illustrated in detail in Appendix 2, our target is suitable for supervised learning since the dataset is labelled. Also, we want to predict the orientation angle, then we aim to build a regression model. In this section, we will show how the performance of the regression models is being assessed.

3.1. Predictability assessment

For constructing and evaluating predictive models of the orientation angle, the data samples were split, as shown in Figure 4, into three sets:

- Training set: which assigned 70% of the samples for model training.
- Test set: had 20% of the samples for model testing based on specified quality metrics.
- Validation set: about 10% of the data (never been used in model training).

The validation samples will be used to evaluate the generalization performance of the trained model. The validation set contains whole new shapes that the model didn't learn about before. If the model is working well on the validation set, it indicates good generalization performance without overfitting. Additionally, we used cross-validation to ensure this issue wasavoided.

Table 2. Data statistics.

	Mean	Std	25%	50%	75%	Min	Max
Features							
ϕ_P	0.500012	0.343768	0.167270	5.000000e−01	0.836781	1.233260e−08	0.99999998
ϕ_N	0.501887	0.429708	0.000000	5.000000e−01	1.000000	0.000000	1.000000
ϕ_S	0.497873	0.429664	0.000000	4.905020e−01	1.000000	0.000000	1.000000
ϕ_E	0.499246	0.429741	0.000000	4.950889e−01	1.000000	0.000000	1.000000
ϕ_W	0.500410	0.429676	0.000000	5.000000e−01	1.000000	0.000000	1.000000
ϕ_{NE}	0.500769	0.460116	0.000000	5.022264e−01	1.000000	0.000000	1.000000
ϕ_{NW}	0.499917	0.461221	0.000000	4.995922e−01	1.000000	0.000000	1.000000
ϕ_{SE}	0.499885	0.461172	0.000000	5.000000e−01	1.000000	0.000000	1.000000
ϕ_{SW}	0.499377	0.460100	0.000000	4.955921e−01	1.000000	0.000000	1.000000
Outputs							
$\cos(\theta)$	−0.000817	0.706504	−0.703404	1.459151e−02	0.707094	−1.000000	1.000000
$\sin(\theta)$	0.002778	0.707704	−0.707477	1.167711e−03	0.709558	−1.000000	1.000000

Table 3. Summary of quality metrics.

Quality metric	Equation	Note
Mean square error	$\text{MSE}(y,\hat{y}) = \frac{1}{m}\sum_{i=0}^{m-1}(y_i-\hat{y}_i)^2$	Expected value of the squared (quadratic) error
Coefficient of determination	$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=1}^{m}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{m}(y_i-\bar{y})^2}$	Range from [−1, 1] and the best possible score is 1
Maximum residual error	$\text{Max Error}(y,\hat{y}) = \max(\lvert y_i-\hat{y}_i\rvert)$	Worst-case error between predicted and true values
Explained variance	$\text{Explained Variance}(y,\hat{y}) = 1 - \frac{\text{Var}\{y-\hat{y}\}}{\text{Var}\{y\}}$	Best possible score is 1, while lower values are worse
Mean absolute error	$\text{MAE}(y,\hat{y}) = \frac{1}{m}\sum_{i=0}^{m-1}\lvert y_i-\hat{y}_i\rvert$	expected value of the absolute error loss

We denote $\hat{y}_i$ as the predicted value of the ith sample, y_i as the corresponding true value, m as the number of samples, Var is variance.

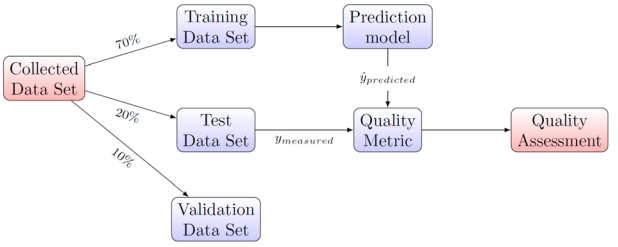


Figure 4. Schematic of the prediction model assessment (adapted from Rushdi et al. (2020)).

For orientation angle prediction, the model performance is evaluated for the testing data by comparing the model prediction to the ground-truth orientation angle. Quality metrics (Table 3) were employed as cost functions to be minimized during model training. The cost functions can be minimized by different optimization algorithms such as the gradient descent one.

If we choose the mean-square error as our performance measure or cost function from the quality metrics, our goal will be to identify the feature weight values that minimize the mean-square error. The following is a mathematical representation of the minimization problem:

$$\underset{\boldsymbol{\theta}}{\text{minimize}}\ \frac{1}{m}\sum_{i=1}^{m}(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)})^2 \tag{7}$$

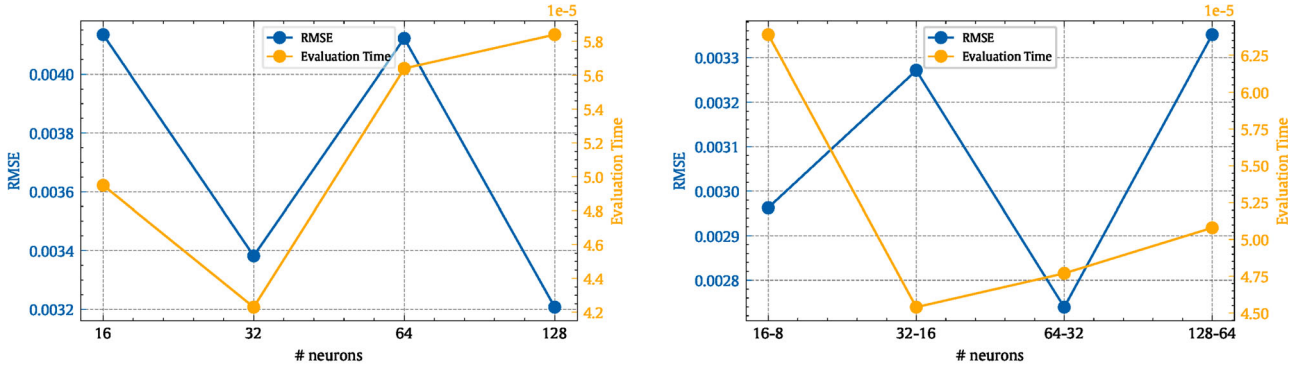
where h is the hypothesis function and $h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})$ is the predicted value. To numerically solve this minimization problem, we could use an algorithm such as the gradient descent (Ruder, 2016) with the following update rule:

$$\text{repeat until convergence}\left\{\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta}J(\boldsymbol{\theta_j})\right\} \tag{8}$$

where α is the learning rate which represents the steps. θ_j is the jth feature and j is from 0 to the number of features m. The symbol $:=$ denotes an iterative update process. Simply, if the derivative term has a negative sign, then θ_j will decrease approaching the local minimum point, and if the derivative term has a positive sign, then θ_j will increase approaching the local minimum point. We don't have to worry about getting stuck in local minima as our cost function is a convex function and has one global minimum point.

4. DL results

In this section, we present the results of deep learning models. We experimented with neural network architectures consisting of 1 and 2 hidden layers with different numbers of neurons. Our aim was to find the architecture that yields the best accuracy or higher prediction capabilities, while minimizing evaluation time, which is a critical constraint for the CFD application. Note that training



(a) 1 hidden layer (b) 2 hidden layers

Figure 5. Performance comparison according to root mean square error (RMSE) and evaluation time for (a) 1 hidden layer with different numbers of neurons (16, 32, 64, 128), and (b) 2 hidden layers with different sets of neurons (16-8, 32-16, 64-32, 128-64). We used the conventional style of descending powers on 2.

time, which is the time required to train the model, could be hours. However, the time entity of interest is the evaluation time, which is the time required to predict the output based on 1 sample (set of features).

As observed in Figure 5(a) for 1 hidden layer, 32 neurons showed the best combination between accuracy and evaluation time. Although the 128 neurons case results in better accuracy than the 32 neurons case, the evaluation time is very high, making it unsuitable for our application.

In contrast, Figure 5(b) for 2 layers shows enhanced accuracy, but with a dramatic increase in evaluation time. For this reason, we will choose the architecture of 1 hidden layer with 32 neurons as our best model and present its results in detail.

Following the tuning strategy presented in Appendix 3, we performed 9 trials, according to the dropout and regularization parameters, as shown in Figure 6. The performance of each trial was judged by the coefficient of determination (R^2) and RMSE, which is the root of MSE represented in Table 3. For all trials, we used a fixed set of hyperparameters: number of Epochs = 500, Batch Size = 256, and Learning Rate = 0.001. The best-tuned model was found to be the one with dropout = 0% and no L_2 regularization, which resulted in $R^2 = 0.999982$ and RMSE = 0.003.

Figure 7 shows the evolution of the training and test losses with training epochs. Once the model is trained to a satisfactory error threshold, the trained model can be used to predict orientation angle values for new input features. It could be noticed that the error dropped dramatically within the first 50 epochs and then became steady.

Figure 8 represents the true values vs predicted values of the orientation angle. A perfect model is represented by a green line with a 45° slope. The red line, which represents the NN model, is almost identical to the ideal line, indicating powerful prediction capabilities. Furthermore, Figure 8(a) demonstrates the robustness of our model, as it can accurately predict the validation set even though it has no prior knowledge of this data. While Figure 8(b,c) represent the powerful prediction accuracy in the test and training sets, respectively.

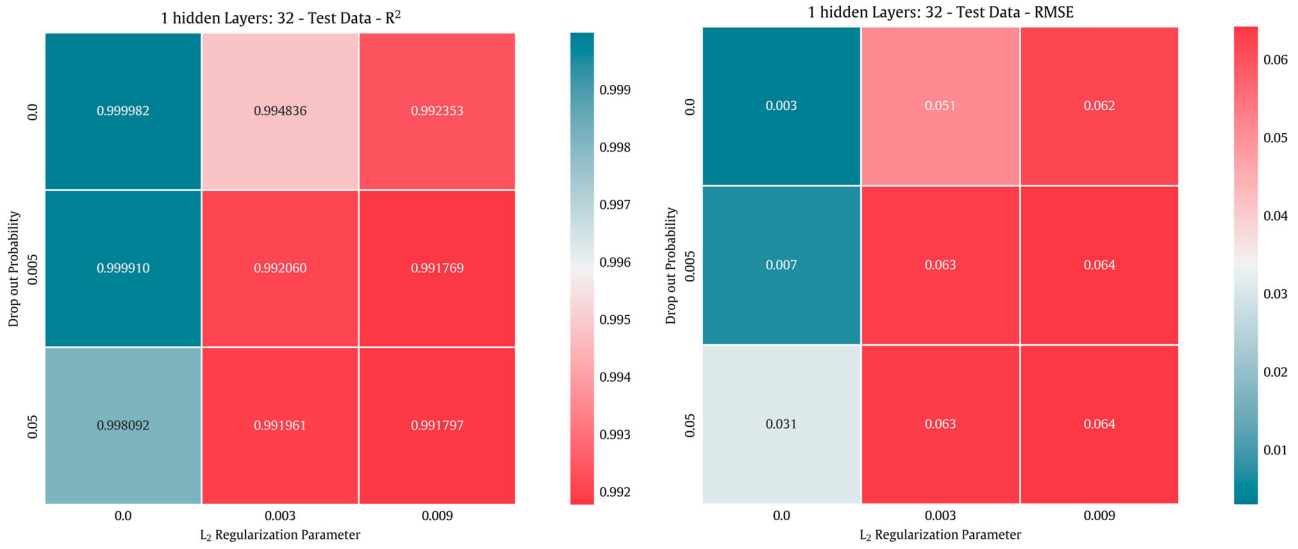
Figure 9 shows histograms of the prediction errors made by the NN regression model on the validation, test, and training sets, respectively. The error is a thin line around zero.

To investigate the model's behaviour, we plot the true values of the orientation angle against the absolute values of the prediction error in Figure 10. This figure reveals that the model exhibits weaker prediction capabilities at corner points (0°, 90°, and 180°) compared to its normal behaviour. Nevertheless, the error at these corner points is still within acceptable limits.

A convolutional neural network (CNN) was constructed, as described in Section A.4, and its performance was compared to that of the neural network (NN) model (Figure 11). While the CNN demonstrated strong prediction capabilities comparable to those of the NN, it required significantly longer training and evaluation times. Furthermore, due to dimensional differences, the CNN was incompatible with the compact stencil used in our study. Therefore, we will use the NN model featuring 1 hidden layer and 32 neurons when solving CFD cases, as shown in Section 6.

5. Comparative analysis

This section provides a comparative analysis of three computational methods for predicting orientation angles: neural networks (NN) with one hidden layer and 32 neurons, convolutional neural networks (CNN), and Youngs



(a) R^2 (b) RMSE

Figure 6. The tuning of dropout and regularization, based on (a) R^2 and (b) RMSE quality metrics, represented as heatmap.

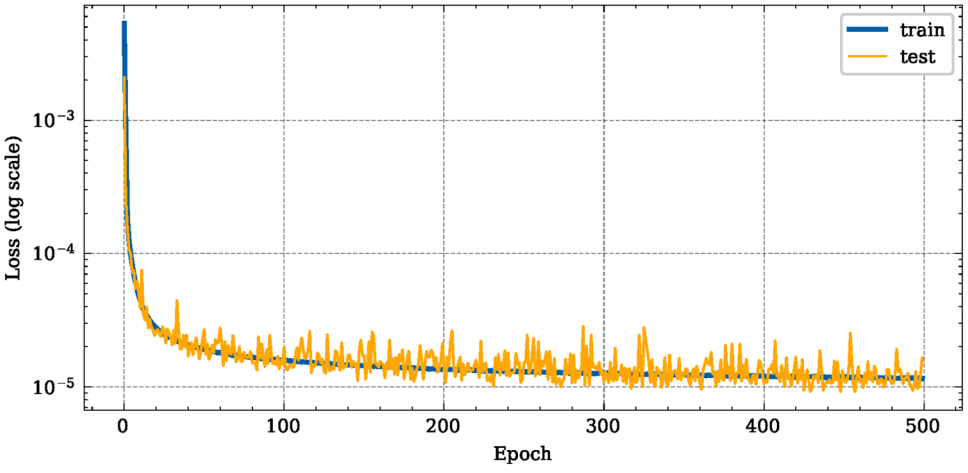
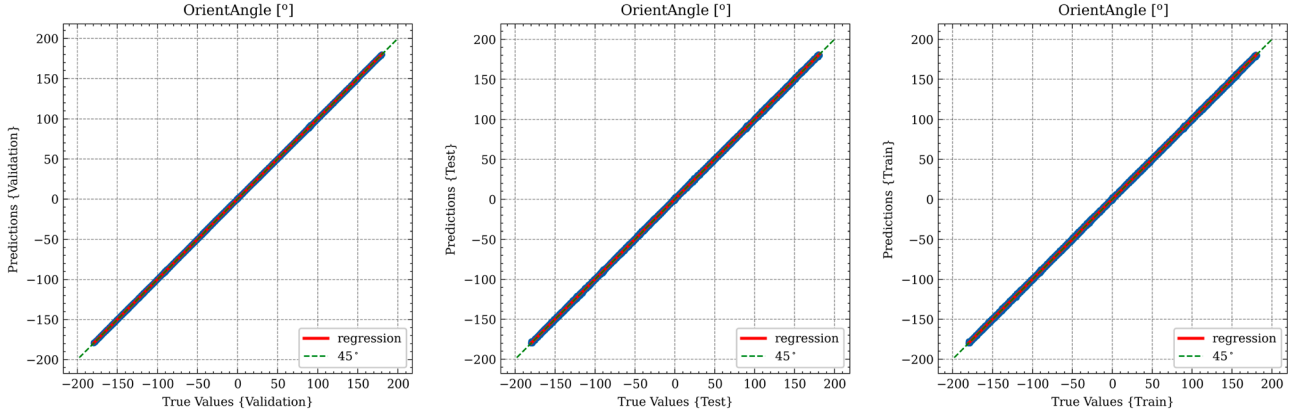
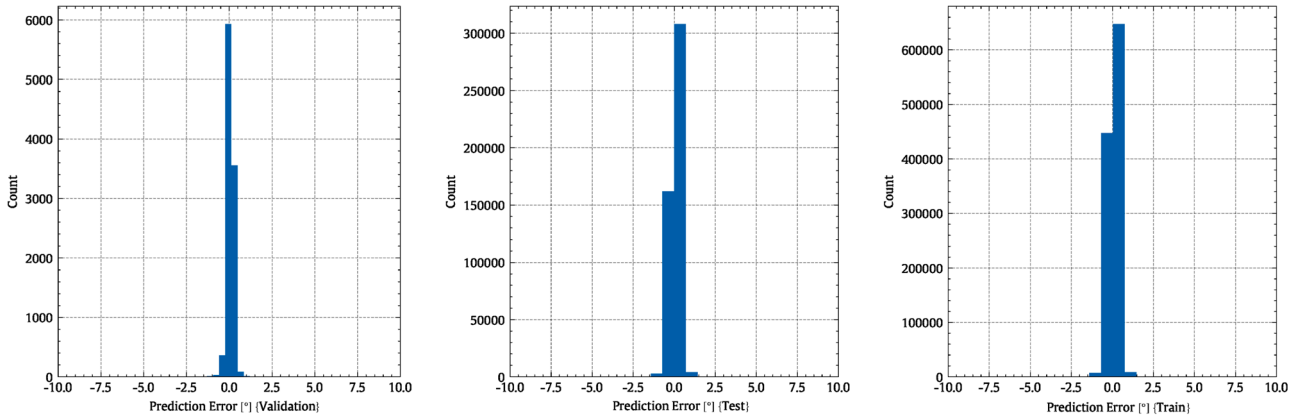


Figure 7. Learning curve (MSE-based) of the proposed deep learning model for orientation angle prediction.



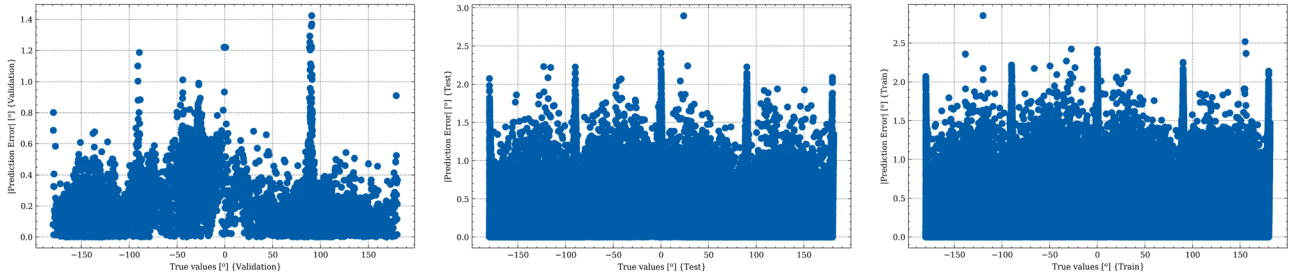
(a) Validation set (b) Test set (c) Train set

Figure 8. Predicted vs actual orientation angles: (a) validation, (b) test, and (c) training sets.



(a) Validation set (b) Test set (c) Train set

Figure 9. Histograms of prediction errors for the NN regression model: (a) validation, (b) test, and (c) training sets.



(a) Validation set (b) Test set (c) Train set

Figure 10. Prediction error [°] vs true values [°] for the NN regression model. (a) Validation set, (b) test set, (c) train set.

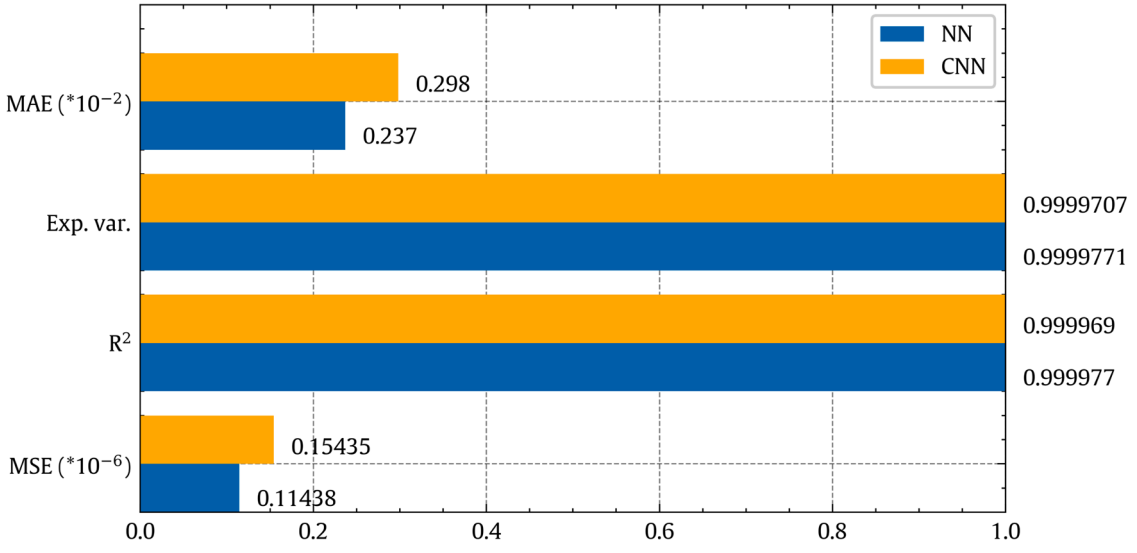


Figure 11. Performance comparison of neural network (NN) and convolutional neural network (CNN) models in orientation angle prediction.

method, a classical approach widely employed in the context of the finite volume method. It was demonstrated that the NN architecture with one hidden layer and 32 neurons outperformed the CNN model in terms of accuracy.

The discussion centers on the feasibility of replacing traditional classical methods with a NN model for computing orientation angles within computational fluid dynamics solvers. Our results, presented in Tables 4 and 5, indicate that the NN model achieves comparable

Table 4. Comparison of mean orientation angle errors [°] between machine learning models and classical method.

	NN (1 layer - 32)	CNN	Classical method
Full	0.180788507	0.224068289	1.065619
Compact	1.055531754	–	5.355739

Table 5. Maximum error in orientation angle prediction [°] for machine learning models and classical method.

	NN (1 layer - 32)	CNN	Classical method
Full	2.283015863	2.345339023	5.166765
Compact	6.546864392	–	16.319065

precision to existing classical methods, often surpassing them. Notably, the NN model exhibits lower average and maximum errors for both full and compact stencils, demonstrating its potential as a viable alternative to traditional classical methods.

This is particularly significant when using the compact stencil, which has shown poor performance and is rarely employed in practice. Furthermore, this study showcases the ability of machine learning models to capture complex correlations within computational stencils while maintaining low complexity. This suggests that NN models can be effective tools for enhancing the accuracy and efficiency of CFD solvers.

When integrating machine learning models into computational fluid dynamics software, model prediction time is a critical consideration, particularly for large-scale evaluations. Our results show that using a full stencil increases model prediction time by approximately 32% compared to a compact stencil, with prediction times of 0.38 and 0.29 s, respectively, for one million evaluations. This increase occurs despite the neural network's input layer for the full stencil containing nearly double the number of inputs.

The observed difference underscores the impact of stencil size on prediction efficiency, likely due to the additional data handling and matrix operations required for the full stencil. While stencil size is a significant factor in prediction time, neural network architecture and hardware optimization may also contribute. These findings highlight the importance of carefully selecting stencil configurations to achieve an optimal balance between model prediction efficiency and accuracy.

Although traditional methods may offer advantages in terms of computational evaluation cost, ML models have the potential to better leverage parallel computing architectures, such as high-performance CPUs and GPUs. As technology continues to evolve rapidly, this advantage is likely to become even more pronounced, potentially leading to a decrease in evaluation costs.

The present study aims to investigate the errors of the machine learning model on shapes not included in the training dataset. The results are compared with those obtained using classical methods, highlighting their main differences. The outcomes of both techniques are presented for full and compact stencils. Two distinct scenarios are considered, each with different levels of complexity in terms of used shapes.

The results presented in Figure 12 indicate that the ML model, referred to as NN, exhibits notably superior accuracy compared to existing classical methods when applied to the full stencil. The utilization of a compact stencil in the NN model also yields significantly higher accuracy compared to classical methods. Furthermore, the NN model's performance with a compact stencil is comparable to the performance of classical methods using a full stencil (e.g. Young's method), as shown previously in Tables 4 and 5. These outcomes highlight the ML model's ability to generalize effectively for smoother geometries with gradual curvature, benefiting from both training data and global spatial information.

Furthermore, the findings in Figure 13 reveal that the ML model achieves lower maximum error levels with the full stencil compared to classical methods, highlighting its superior accuracy in capturing critical features. While the error distribution is similar between the ML model and classical methods, the NN model demonstrates a more consistent reduction in error magnitude across the domain. This is particularly evident with the compact stencil, where the NN model shows localized maxima but maintains lower error across the majority of the domain, resulting in a reduced mean error overall.

The differences in performance between Figures 12 and 13 are influenced by the increased complexity of the shape in Figure 13, which introduces sharper features and abrupt curvature changes. These characteristics challenge classical methods, leading to higher localized errors due to their reliance on local gradient estimations. In contrast, the NN model's ability to learn and leverage global spatial relationships enables it to mitigate these challenges, achieving lower maximum and mean errors, even for sharp features not present in the training data.

We conducted SHAP (SHapley Additive exPlanations) analysis (Lundberg & Lee, 2017) for a set of random samples to identify the critical variables that influence orientation angle prediction used in the VOF simulations. The summary plots provided (Figure 14) highlight the importance of individual features and their impact on the model's output. The colour gradient indicates feature values (high in red and low in blue), while the SHAP value quantifies the magnitude and direction of a feature's contribution to a prediction. Additionally, the bar

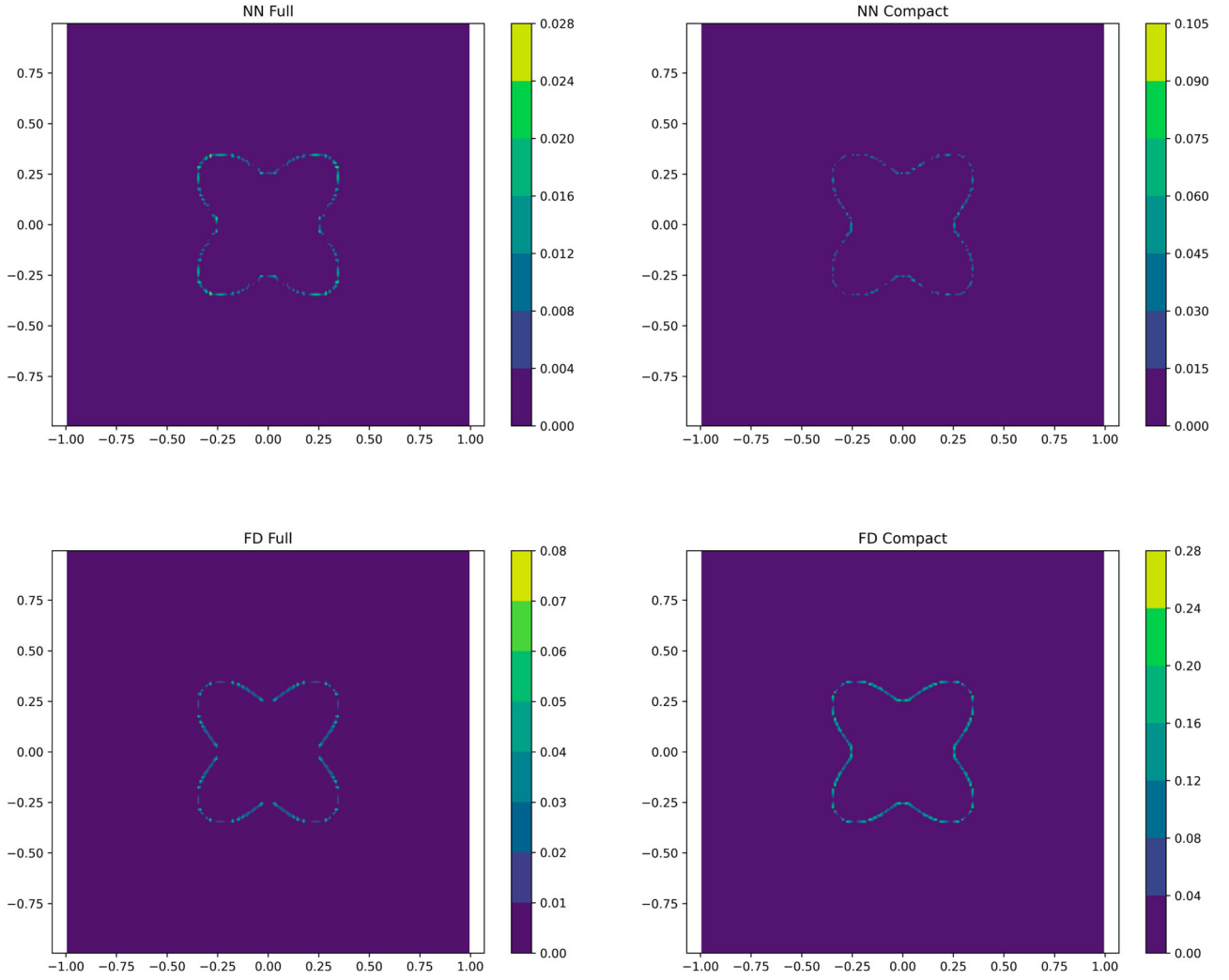


Figure 12. Assessment of difference error between the predicted interface normal and exact value estimated analytically for the first shape: a mild example not included in the training dataset.

plots (Figure 15) quantify the average importance of each feature.

The SHAP analysis reveals that the machine learning model relies heavily on neighbouring cell values (e.g. ϕ_W, ϕ_E, ϕ_S, and ϕ_N) while showing very poor dependency on the central cell (ϕ_P). This is consistent with the finite difference method, where interface normals are estimated based on gradients calculated as differences between neighbouring cells, and the central cell does not directly contribute to these computations. The model's behaviour demonstrates its ability to learn the same directional dependencies inherent in finite difference stencils, focussing on relative variations between neighbours to estimate the interface normal.

Furthermore, the model's reliance on specific directions (e.g. ϕ_N and ϕ_S for the cosine component, ϕ_W and ϕ_E for the sine component) highlights how it mirrors the physical principles underlying finite difference calculations while potentially capturing nonlinear relationships that finite differences cannot. The negligible role of ϕ_P further validates that the model does not depend on absolute scalar values but instead effectively utilizes the surrounding cell values for gradient-based predictions. This consistency supports the model's suitability as a replacement for finite difference methods in estimating interface normals.

6. Benchmark cases

The main objective of this section is to examine and assess the numerical accuracy of the developed ML model. We verify its performance against existing methods for evaluating the interface normal vector within the PLIC framework. All numerical experiments were conducted on uniform 2D quadrilateral grids.

The model was implemented in OpenFOAM v2312 within the isoAdvector solver framework (Roenby et al., 2016) as an alternative method to estimate the interface

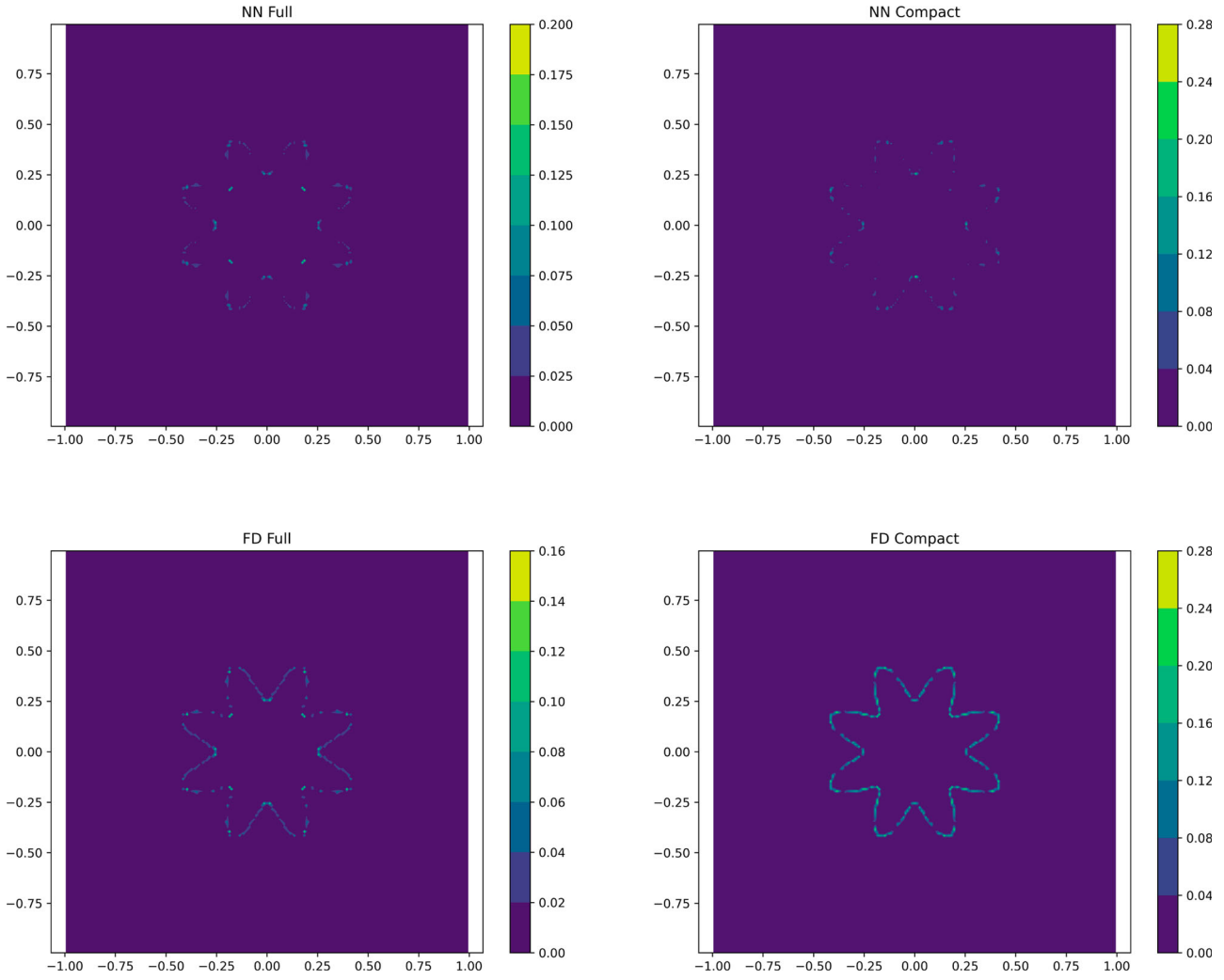
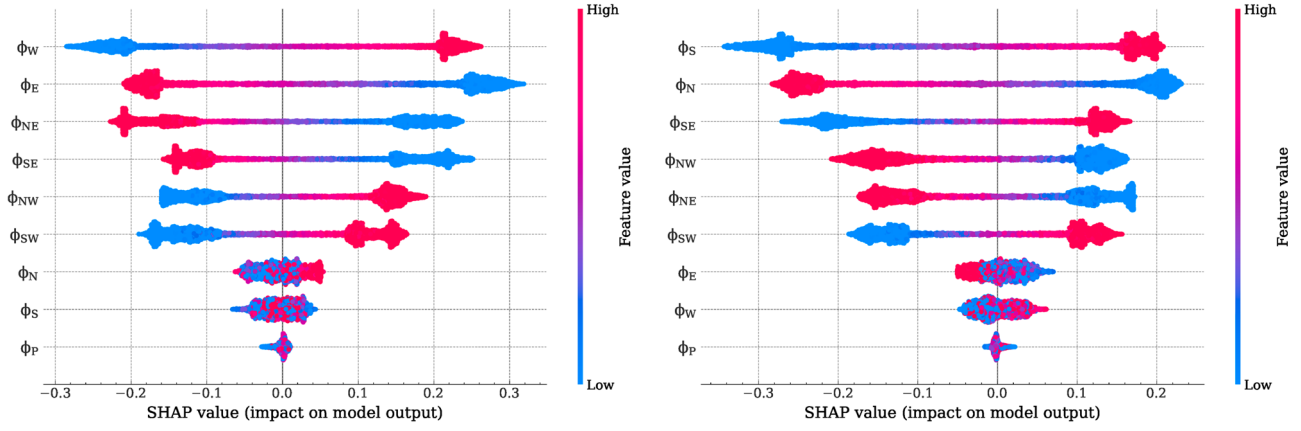
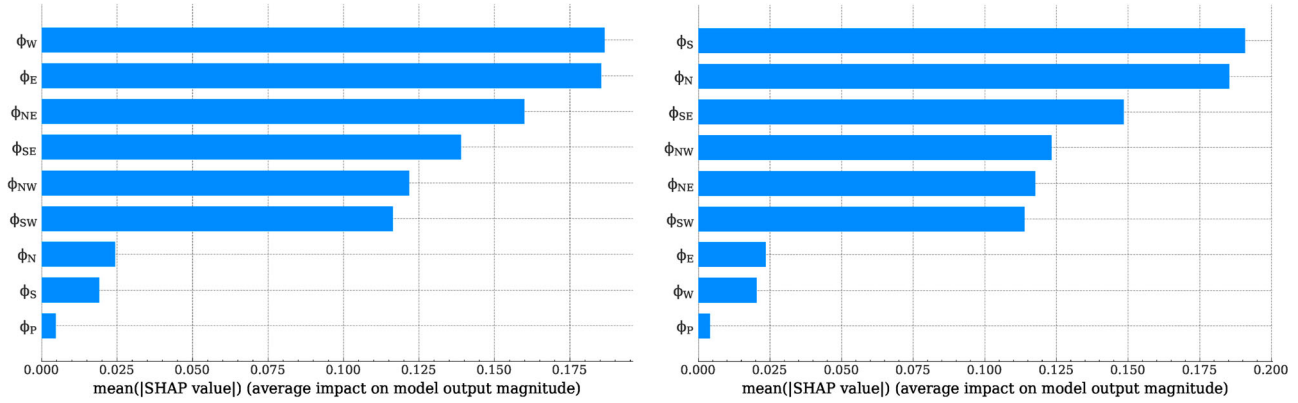


Figure 13. Assessment of difference error between the predicted interface normal and exact value estimated analytically for the second shape: a more severe example not included in the training dataset.



(a) First output (cos(orientation angle)) (b) Second output (sin(orientation angle))

Figure 14. Beeswarm summary plots of SHAP analysis. (a) First output (cos(orientation angle)) and (b) second output (sin(orientation angle)).



(a) First output (cos(orientation angle)) (b) Second output (sin(orientation angle))

Figure 15. Bar plots of SHAP analysis. (a) First output (cos(orientation angle)) and (b) second output (sin(orientation angle)).

normal vector used in the solver. To provide a point of comparison, Youngs' formula (Harvie & Fletcher, 2000) for calculating the interface normal vector was also implemented within the isoAdvector framework.

Youngs' method computes the interface normal vector by averaging and normalizing the gradient of the volume fraction field, ϕ, at the surrounding cell corners. The derivatives at these corners are determined using the volume fraction field at neighbouring cells, as illustrated in Figure 2. For instance:

$$m_{x,\mathrm{NE}} = \frac{\phi_E + \phi_{\mathrm{NE}} - \phi_P - \phi_N}{2h},$$
$$m_{y,\mathrm{NE}} = \frac{\phi_N + \phi_{\mathrm{NE}} - \phi_P - \phi_E}{2h}, \tag{9}$$

where h is the cell size.

The unit normal vector for a given cell P is then computed as:

$$(n_x, n_y)_P = \left(\frac{h\, m_{x,P}}{\sqrt{(m_{x,P})^2 + (m_{y,P})^2 + \epsilon}}, \right.$$
$$\left. \times \frac{h\, m_{y,P}}{\sqrt{(m_{x,P})^2 + (m_{y,P})^2 + \epsilon}} \right), \tag{10}$$

where ϵ is a small positive value, typically set to 10^{-16}, to avoid numerical issues such as division by zero.

The derivatives at the cell center P are determined using the values at the surrounding corners (NW, SW, NE, SE) by averaging as follows:

$$m_{x,P} = \tfrac{1}{4}\left(m_{x,\mathrm{NW}} + m_{x,\mathrm{SW}} + m_{x,\mathrm{NE}} + m_{x,\mathrm{SE}}\right), \tag{11}$$
$$m_{y,P} = \tfrac{1}{4}\left(m_{y,\mathrm{NW}} + m_{y,\mathrm{SW}} + m_{y,\mathrm{NE}} + m_{y,\mathrm{SE}}\right). \tag{12}$$

While more recent methods have been proposed, these are typically designed for curvilinear structured meshes, or unstructured grids. As our study is focussed specifically on uniform Cartesian grids, which align with the scope of Youngs' original work, we have chosen to use it as a baseline for comparison.

To validate the accuracy and robustness of the proposed method, calculations were performed for two well-known test cases from the literature: Zalesak's Notched Disc 2D solid body rotation (Zalesak, 1979) and Rider and Kothe's 2D vortex flow (Rider & Kothe, 1998). These benchmark cases are commonly used to evaluate numerical methods for interface tracking, as they test the ability of the algorithm to preserve interface sharpness and handle complex deformations. The results for these cases were compared with existing methods to assess the performance of the developed approach, with errors quantified using two error metrics, E_1 and E_2, which measure the deviation from the exact solution. The error metrics are defined as follows:

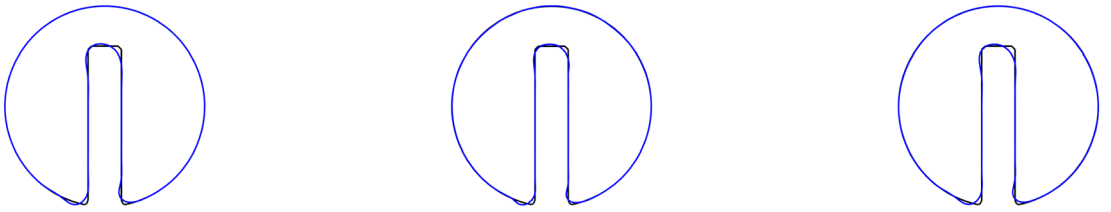
$$E_1 = \sum_{i=1}^{N} |\varphi_n - \varphi_e|_i |\Omega_i|, \tag{13}$$
$$E_2 = \frac{\sum_{i=1}^{N} |\varphi_n - \varphi_e|_i |\Omega_i|}{\sum_{i=1}^{N} |\varphi_e|_i |\Omega_i|}, \tag{14}$$

where φ_n is the numerical (computed) value of the variable φ in a given cell i, φ_e is the exact value used for comparing the accuracy of the numerical solution, Ω_i is the volume of cell i and N is the total number of cells in the computational domain.

Additionally, we performed a dam break simulation to evaluate the method's applicability to realistic and highly dynamic interfacial flows, which are commonly encountered in engineering applications.

6.1. 2D solid body rotation

We employ the well-known 2D solid body rotation test of the notched/slotted disc by Zalesak (1979). In the



(a) isoAdvector/NN 9-cell Stencil (b) isoAdvector/NN 5-cell Stencil (c) isoAdvector/Young's method

Figure 16. 0.5 VOF contour after one period on a grid with 200 × 200 resolution using different methods for estimating the interface normal for Zalesak's rotation test. (blue: numerical method, black: exact). (a) isoAdvector/NN 9-cell Stencil, (b) isoAdvector/NN 5-cell Stencil and (c) isoAdvector/Young's method.

Table 6. Error metrics for Zalesak's 2D solid body rotation test.

CFL	Method	E_1	E_2
0.25	isoAdvector/NN 9-cell Stencil	0.000822	0.0140
	isoAdvector/NN 5-cell Stencil	0.000807	0.0137
	isoAdvector/Young's method	0.000870	0.0148
0.10	isoAdvector/NN 9-cell Stencil	0.000712	0.0121
	isoAdvector/NN 5-cell Stencil	0.000708	0.0121
	isoAdvector/Young's method	0.000842	0.0143

computational domain $[0, 1] \times [0, 1]$, a circle with a radius of 0.15, centered at (0.5, 0.75), is initially set and notched with a slot defined by $|x - 0.5| \leq 0.025$ and $y \leq 0.85$. The notched circle is transported by a rotational velocity field with a constant angular velocity given by $(v_x, v_y) = (y - 0.5, 0.5 - x)$, which returns the circle to its initial position after one revolution.

In this test, time steps are selected so that the maximum CFL number is equal to 0.25, and the computation is performed for a period of 2π. The interface is shown to be almost identical to its original shape in Figure 16, where it is represented as a 0.5-contour of VOF. This indicates that the generated model can accurately predict the interface normal vector, indicating its overall accuracy in capturing the interface and producing nearly identical results as existing approaches. To examine the numerical results more quantitatively, Table 6 provide a summary of the numerical errors.

The model's performance is evaluated across various configurations, including isoAdvector/NN with 9-cell and 5-cell stencils, as well as isoAdvector/Young's method, under different conditions such as varying CFL numbers and grid resolutions. The results consistently demonstrate that the developed ML models achieve low error values, underscoring their accuracy and robustness in capturing interface dynamics for this test case. Notably, both the isoAdvector/NN 9-cell Stencil and 5-cell Stencil configurations exhibit particularly low error values, highlighting their potential for practical applications in computational fluid dynamics.

A particularly noteworthy observation is the performance of the isoAdvector/NN 5-cell Stencil, which achieves accurate interface reconstruction despite its reduced stencil size. Traditional finite volume methods with a similar 5-cell stencil often encounter significant challenges due to limited spatial information. However, the neural network's capacity to identify and exploit complex nonlinear patterns enables it to produce accurate predictions even with fewer data points or simpler stencils, demonstrating the adaptability and effectiveness of the NN-based approach.

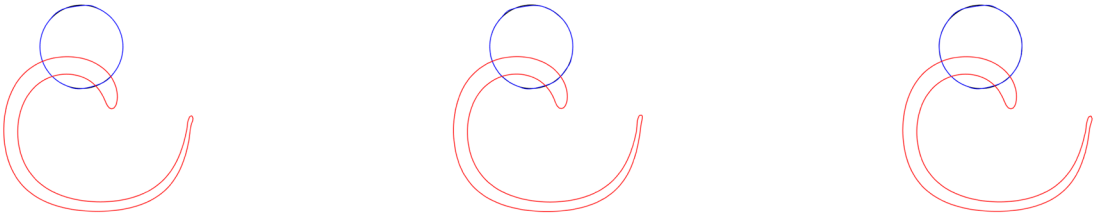
It is important to note that the simplicity of the velocity field in this test case influences the observed error behaviour. At smaller CFL numbers (i.e. smaller time steps), improvements in interface normal estimation (e.g. orientation accuracy) result in noticeable accuracy gains. Conversely, at higher CFL numbers (i.e. larger time steps), advection errors become more significant, diminishing the relative contribution of interface normal improvements. This behaviour reflects the dynamic relationship between static reconstruction and advection errors, which varies based on the numerical configuration and flow conditions.

6.2. 2D single vortex shear flow

The benchmark test for shear flow, as introduced by Harvie and Fletcher (2000), is commonly used to evaluate the precision of different interface-capturing methods in managing deformed and elongated interfaces. The initial VOF configuration is described as a circle with a radius of 0.15, centered at (0.5, 0.75) within a unit domain $[0, 1]^2$ on a 128 × 128 uniform grid. This initial circle is transported by a time-dependent velocity field given by

$$u = \frac{2}{\pi} \sin^2(\pi x) \sin(\pi y) \cos(\pi y) \cos\left(\frac{\pi t}{T}\right),$$
$$v = -\frac{2}{\pi} \sin^2(\pi y) \cos(\pi x) \sin(\pi x) \cos\left(\frac{\pi t}{T}\right), \quad (15)$$

where T is the revolution period.



(a) isoAdvector/NN 9-cell Stencil (b) isoAdvector/NN 5-cell Stencil (c) isoAdvector/Young's method

Figure 17. 0.5 VOF contour after one period on a grid with 128 × 128 resolution using different methods for estimating the interface normal for 2D shear flow test ($T = 4$ s). (red: numerical method at $t = T/2$, blue: numerical method at $t = T$, black: exact). (a) isoAdvector/NN 9-cell Stencil (b) isoAdvector/NN 5-cell Stencil and (c) isoAdvector/Young's method.

As the velocity field progresses, the initial circle is distorted and elongated into a spiral with a narrow tail. The peak deformation happens at $t = T/2$, after which the velocity field reverses, restoring the spiral to its original circular form when $t = T$. This benchmark test assesses the model's capability to precisely capture thin and highly distorted interface geometries.

As the velocity field evolves, the initial circle deforms into an elongated spiral with a narrow tail. The peak deformation occurs at $t = T/2$, after which the velocity field reverses, restoring the spiral to its original circular shape at $t = T$. This benchmark test evaluates the model's ability to capture thin and distorted interface geometries with precision, highlighting challenges associated with multiple error sources.

In this case, both advection errors, caused by the complex velocity field, and static reconstruction errors, resulting from inaccuracies in interface normal estimation and interface placement, contribute to the overall error. The thin and highly distorted regions of the interface amplify these errors, providing a practical assessment of the model's effectiveness in handling geometric deformations of the interface.

6.2.1. Case I: $T = 4$ s

In Figure 17, the 0.5 VOF contours for the 2D shear flow test at $T/2$ (represented by the red line) provide insights into the performance of different methods under maximum deformation. At this time, the initial circular interface is stretched into a spiral shape with a thin tail, representing a significant challenge for numerical methods to capture accurately. For the developed ML model, both the isoAdvector/NN 9-cell Stencil and isoAdvector/NN 5-cell Stencil configurations exhibit accurate tracking of the interface, closely matching the solution obtained using Young's method. After one period, the 0.5 VOF contours (represented by the blue line) illustrate the capability of different methods in capturing the interface evolution under significant deformation. These contours, plotted after one period of the flow, show the comparison between the numerical method and the exact solution. Qualitatively, the developed ML model with isoAdvector/NN 9-cell Stencil and isoAdvector/NN 5-cell Stencil configurations perform very well with minimal deviation from the exact solution.

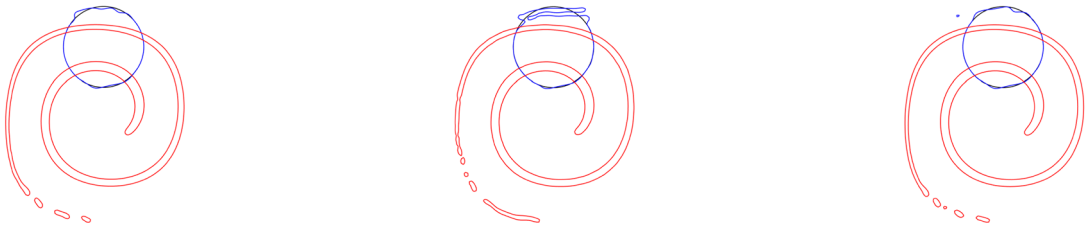
Table 7. Error metrics for 2D shear flow test at $T = 4$, $N = 128$.

Method	E_1	E_2
isoAdvector/NN 9-cell Stencil	0.000500	0.007074
isoAdvector/NN 5-cell Stencil	0.0007920	0.01121
isoAdvector/Young's method	0.0007542	0.01068

A more quantitative analysis is performed by examining the error metric results in Table 7. The isoAdvector/NN 9-cell Stencil configuration is particularly impressive, as it effectively handles intricate deformations and provides superior accuracy compared to the isoAdvector/Young's method. The isoAdvector/NN 5-cell Stencil, while having slightly higher errors, still performs comparably to Young's method. The slightly lower accuracy of the 5-cell stencil can be attributed to the reduced input data available in the stencil, which is less effective in handling the very thin structures, such as the thin tail of the spiral shape at $T/2$. Nevertheless, the 5-cell stencil's performance is impressive, especially considering that Young's method uses a 9-cell stencil and achieves similar error levels.

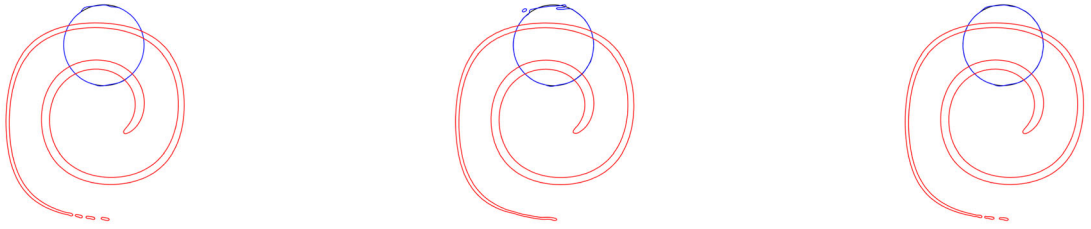
6.2.2. Case II: $T = 8$ s

A more challenging test is demonstrated by setting the period to $T = 8$ s. As shown in Figure 18, the spiral shape becomes even thinner, causing the error to accumulate more significantly over time. It should be noted that in this case, the errors due to advection plays more dominant role due to the nature of velocity field and time duration of the simulation. In this scenario, the isoAdvector/NN 9-cell Stencil performs very well compared to other schemes. On the other hand, the isoAdvector/NN 5-cell Stencil performs poorly in comparison. This is due to insufficient data in the stencil, although it still performs quite well compared to other schemes



(a) isoAdvector/NN 9-cell Stencil (b) isoAdvector/NN 5-cell Stencil (c) isoAdvector/Young's method

Figure 18. 0.5 VOF contour after one period on a grid with 128 × 128 resolution using different methods for estimating the interface normal for 2D shear flow test ($T = 8$ s). (red: numerical method at $t = T/2$, blue: numerical method at $t = T$, black: exact). (a) isoAdvector/NN 9-cell Stencil, (b) isoAdvector/NN 5-cell Stencil and (c) isoAdvector/Young's method.



(a) isoAdvector/NN 9-cell Stencil (b) isoAdvector/NN 5-cell Stencil (c) isoAdvector/Young's method

Figure 19. 0.5 VOF contour after one period on a grid with 256 × 256 resolution using different methods for estimating the interface normal for 2D shear flow test ($T = 8$ s). (red: numerical method at $t = T/2$, blue: numerical method at $t = T$, black: exact). (a) isoAdvector/NN 9-cell Stencil, (b) isoAdvector/NN 5-cell Stencil and (c) isoAdvector/Young's method.

with similar stencil sizes, which, based on our numerical experiments, crashed and could not complete the simulation. This is confirmed by repeating the simulation with a 256 × 256 resolution (Figure 19), where the performance of the isoAdvector/NN 5-cell Stencil improves considerably since more data are now available to resolve the thin structures. However, it still remains less accurate compared to the other methods (namely isoAdvector/NN 9-cell Stencil and isoAdvector/Young's method) considered in this study.

In Table 8 confirms our previous findings and, the isoAdvector/NN 9-cell Stencil configuration is particularly impressive, as it effectively handles intricate deformations and provides superior accuracy compared to the isoAdvector/Young's method. The isoAdvector/NN 5-cell Stencil, while having slightly higher errors, still performs comparably to Young's method. The slightly lower accuracy of the 5-cell stencil can be attributed to the reduced input data available in the stencil, which is less effective in handling the very thin structures, such as the thin tail of the spiral shape at $T/2$. Nevertheless, the 5-cell stencil's performance is impressive, especially considering that Young's method uses a 9-cell stencil and achieves similar error levels.

Table 8. Error metrics for 2D shear flow test at $T = 4$.

Resolution	Method	E_1	E_2
$N = 128$	isoAdvector/NN 9-cell Stencil	0.002214	0.03134
	isoAdvector/NN 5-cell Stencil	0.006577	0.09310
	isoAdvector/Young's method	0.002067	0.02926
$N = 256$	isoAdvector/NN 9-cell Stencil	0.0005643	0.007984
	isoAdvector/NN 5-cell Stencil	0.001457	0.02062
	isoAdvector/Young's method	0.0005944	0.008410

6.3. Validation against two-phase dam-break experiments

To validate the accuracy of the developed model, numerical predictions were compared with experimental results for a two-phase dam-break problem impacting a vertical wall (Kamra et al., 2019). The simulation utilized the $k - \epsilon$ turbulence model and was conducted with a CFL number of 0.5, highlighting the numerical stability and robustness of the model for practical engineering applications. Snapshots of the free-surface profile obtained from the simulation are presented in Figure 20. The predicted free-surface profiles show strong agreement with experimental observations, demonstrating the model's ability to accurately capture the intricate dynamics of the interface.

This level of accuracy may be attributed to the complex and rich dataset used during model training, which enhances its ability to predict the interface orientation (i.e.the normal vector). This capability is particularly critical for resolving the highly transient nature of the flow

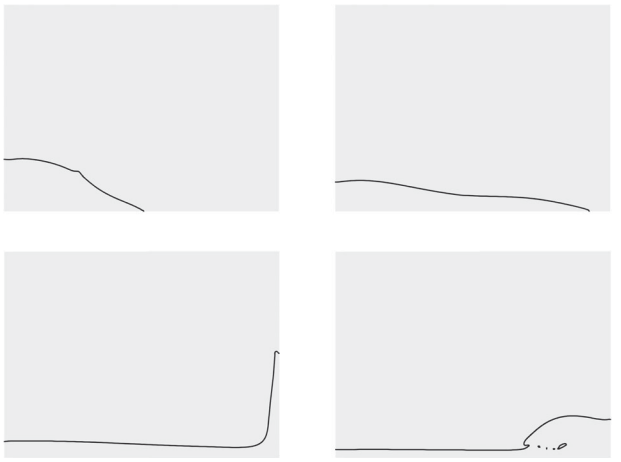


Figure 20. Snapshots of the free-surface profile obtained using isoAdvector/NN 9-cell Stencil.

and accurately capturing the free-surface evolution during the dam-break event.

Additionally, pressure measurements at a specific point on the vertical wall were compared with the model's numerical predictions, as depicted in Figure 21. A minor time lag in the initial impact was observed, which may be attributed to the experimental gate mechanism influencing the flow release dynamics (Kamra et al., 2019), as this effect was not accounted for in the numerical simulation. To address this discrepancy, a 70 ms time shift was introduced to synchronize the point of impact between the numerical and experimental datasets. This adjustment ensured that the comparison focussed on the dynamic variations in pressure rather than discrepancies caused by timing offsets inherent to the experimental setup.

The results demonstrate that the model delivers accurate predictions of both the free-surface evolution and the dynamic pressure distribution, with excellent agreement observed between the numerical and experimental data. These findings underline the model's robustness and its potential for simulating complex two-phase flows in real-world engineering applications.

7. Conclusion

In this work, we demonstrated the application of deep learning methodologies for predicting interface orientation angles (i.e. interface unit normal vectors) in interface computational cells within the context of the volume of fluid method for multiphase flow. The data was generated using the parametric radial star formula, providing the complexity needed for practical engineering problems. The interface orientation angles and volume fractions were analytically determined, serving as the ground truth for training the machine learning model.

We evaluated different neural network architectures and selected a neural network with one hidden layer of 32 neurons for its higher accuracy and lower evaluation time. This model was tested against a Convolutional Neural Network (CNN) and Young's method, and was found to be comparably accurate or even more accurate. Two models were developed using 3 × 3, 9-cell and 5-cell stencils, implemented in the OpenFOAM isoAdvector solver framework as isoAdvector/NN 9-cell and isoAdvector/NN 5-cell Stencil.

Numerical tests demonstrated the effectiveness of the developed ML model, particularly with the isoAdvector/NN 9-cell and 5-cell Stencil configurations. The ML-based methods consistently exhibited accurate interface tracking with minimal deviation from the exact solution, even under significant deformation and challenging flow conditions. The isoAdvector/NN 9-cell Stencil showed superior performance, while the isoAdvector/NN 5-cell

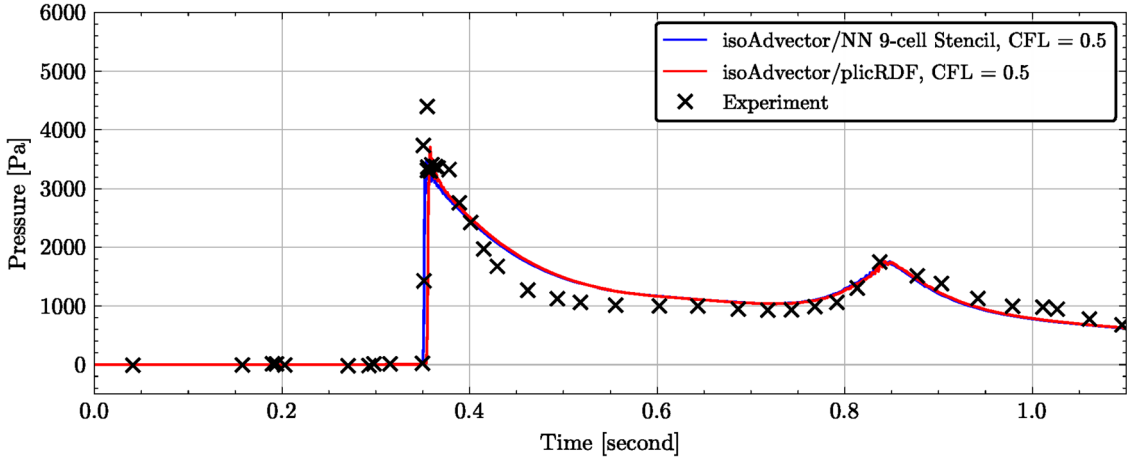


Figure 21. Comparison of pressure time history at a probe point on the vertical wall for the dam-break problem, between numerical predictions and experimental data.

Stencil, although slightly less accurate, still performed comparably well, highlighting the robustness of the ML model. These findings underscore the potential of integrating machine learning techniques into traditional numerical methods to enhance accuracy and reliability in computational fluid dynamics simulations for multiphase flow.

Future work should investigate applying this technique to non-uniform grids, particularly triangular and tetrahedral grids, to enhance the precision of orientation angle approximation while maintaining comparable accuracy with immediate face neighbours.

Nomenclature

Abbreviations

CFD	computational fluid dynamics
CFL	Courant–Friedrichs–Lewy number
CNN	convolutional neural network
ELVIRA	enhanced LVIRA method for VOF interface reconstruction
LVIRA	an interface reconstruction method used in VOF
ML	machine learning
NN	neural network
PLIC	piece-wise linear interface construction
VOF	volume-of-fluid, a method for interface tracking in fluid simulations

Physical Constants

a	amplitude factor in the parametric radial star formula
b	frequency factor in the parametric radial star formula
c	exponent factor in the parametric radial star formula
r_0	baseline radius for the parametric radial star formula

Variables

$\boldsymbol{v}$	velocity vector
∇	gradient operator
ϕ	volume fraction, ranging from 0 to 1, representing the proportion of the heavier phase in a computational cell
$\phi_N, \phi_S, \phi_E, \phi_W$	volume fractions of neighbouring cells (North, South, East, West)
ϕ_P	volume fraction at the interface cell P
$\phi_{\mathrm{NW}}, \phi_{\mathrm{NE}}, \phi_{\mathrm{SW}}, \phi_{\mathrm{SE}}$	volume fractions of diagonal neighbouring cells (Northwest, Northeast, Southwest, Southeast)
θ	interface orientation angle, in radians, with a range of $[0, 2\pi]$
r	radial distance in the parametric radial star formula for dataset shape generation
t	time

Disclosure statement

No potential conflict of interest was reported by the author(s).

Data availability statement

The data that support the findings of this study are available from the corresponding author, M. K., upon reasonable request.

Funding

This research was funded by the United Arab Emirates University [grant code G00004395].

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., & Isard, M. (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (pp. 265–283). USENIX Association. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38. https://doi.org/10.1109/MSP.2017.2743240

Ataei, M., Bussmann, M., Shaayegan, V., Costa, F., Han, S., & Park, C. B. (2021). NPLIC: A machine learning approach to piecewise linear interface construction. *Computers & Fluids*, *223*, 104950. https://doi.org/10.1016/j.compfluid.2021.104950

Ataei, M., Pirmorad, E., Costa, F., Han, S., Park, C. B., & Bussmann, M. (2021). A deep learning algorithm for piecewise linear interface construction (PLIC). arXiv preprint, arXiv:2107.13067.

Bnà, S., Manservisi, S., Scardovelli, R., Yecko, P., & Zaleski, S. (2016). VOFI – A library to initialize the volume fraction scalar field. *Computer Physics Communications*, *200*, 291–299. https://doi.org/10.1016/j.cpc.2015.10.026

Chen, Y., Price, W. G., & Temarel, P. (2012). An anti-diffusive volume of fluid method for interfacial fluid flows. *International Journal for Numerical Methods in Fluids*, *68*(3), 341–359. https://doi.org/10.1002/fld.v68.3

Chollet, F. *et al.* (2015). Keras. Retrieved September 7, 2021, from https://keras.io.

Chorin, A. J. (1980, March). Flame advection and propagation algorithms. *Journal of Computational Physics*, *35*(1), 1–11. https://doi.org/10.1016/0021-9991(80)90030-3

DeBar, R. (1974). *Method in two-D Eulerian hydrodynamics* (Tech. Rep.).

Després, B., & Jourdren, H. (2020). Machine learning design of volume of fluid schemes for compressible flows. *Journal of Computational Physics*, *408*, 109275. https://doi.org/10.1016/j.jcp.2020.109275

Fateen, M., & Mine, T. (2021). Predicting student performance using teacher observation reports. *International Educational Data Mining Society*. ERIC.

Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorFlow: Concepts, tools, and and techniques to build intelligent systems* (2 ed.). O'Reilly Media.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Gupta, A., Anpalagan, A., Guan, L., & A. S. Khwaja (2021). Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array*, *10*, 100057. https://doi.org/10.1016/j.array.2021.100057

Harvie, D. J., & Fletcher, D. F. (2000). A new volume of fluid advection algorithm: The stream scheme. *Journal of Computational Physics*, *162*(1), 1–32. https://doi.org/10.1006/jcph.2000.6510

Hastie, T., Tibshirani, R., & Friedman, J. (2009a). Overview of supervised learning. In *The elements of statistical learning* (pp. 9–41). Springer.

Hastie, T., Tibshirani, R., & Friedman, J. (2009b). Unsupervised learning. In *The elements of statistical learning* (pp. 485–585). Springer.

Ito, K., Kunugi, T., Ohno, S., Kamide, H., & Ohshima, H. (2014, September). A high-precision calculation method for interface normal and curvature on an unstructured grid. *Journal of Computational Physics*, *273*, 38–53. https://doi.org/10.1016/j.jcp.2014.04.058

Ivey, C. B., & Moin, P. (2015). Accurate interface normal and curvature estimates on three-dimensional unstructured non-convex polyhedral meshes. *Journal of Computational Physics*, *300*, 365–386. https://doi.org/10.1016/j.jcp.2015.07.055

Jofre, L., Lehmkuhl, O., Castro, J., & Oliva, A. (2014). A 3-D volume-of-fluid advection method based on cell-vertex velocities for unstructured meshes. *Computers & Fluids*, *94*, 14–29. https://doi.org/10.1016/j.compfluid.2014.02.001

Kamra, M. M., Al Salami, J., Sueyoshi, M., & Hu, C. (2019). Experimental study of the interaction of dambreak with a vertical cylinder. *Journal of Fluids and Structures*, *86*, 185–199. https://doi.org/10.1016/j.jfluidstructs.2019.01.015

Kothe, D. B., & Mjolsness, R. C. (1992, November). RIPPLE – A new model for incompressible flows with free surfaces. *AIAA Journal*, *30*(11), 2694–2700. https://doi.org/10.2514/3.11286

Li, M., Liu, Y., Liu, X., Sun, Q., You, X., Yang, H., Luan, Z., Gan, L., Yang, G., & Qian, D. (2020). The deep learning compiler: A comprehensive survey. *IEEE Transactions on Parallel and Distributed Systems*, *32*(3), 708–727. https://doi.org/10.1109/TPDS.71

López, J., Hernández, J., Gómez, P., & Faura, F. (2004). A volume of fluid method based on multidimensional advection and spline interface reconstruction. *Journal of Computational Physics*, *195*(2), 718–742. https://doi.org/10.1016/j.jcp.2003.10.030

López, J., Zanzi, C., Gómez, P., Faura, F., & Hernández, J. (2008). A new volume of fluid method in three dimensions – Part II: Piecewise-planar interface reconstruction with cubic-Bézier fit. *International Journal for Numerical Methods in Fluids*, *58*(8), 923–944. https://doi.org/10.1002/fld.v58:8

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 4765–4774). Curran Associates, Inc.

Önder, A., & Liu, P. L.-F. (2022). Deep learning of interfacial curvature: A symmetry-preserving approach for the volume of fluid method. arXiv preprint, arXiv:2206.06041.

Owkes, M., & Desjardins, O. (2015). A mesh-decoupled height function method for computing interface curvature. *Journal of Computational Physics*, *281*, 285–300. https://doi.org/10.1016/j.jcp.2014.10.036

Park, I., Kim, K., Kim, J., & Van, S. (2009). A volume-of-fluid method for incompressible free surface flows. *International Journal for Numerical Methods in Fluids*, *61*(12), 1331–1362. https://doi.org/10.1002/fld.v61:12

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A.. 2017). Automatic differentiation in pytorch. In *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA*. https://openreview.net/forum?id=BJJsrmfCZ

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, É., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*(85), 2825–2830.

Pilliod, J. E., & Puckett, E. G. (2004). Second-order accurate volume-of-fluid algorithms for tracking material interfaces. *Journal of Computational Physics*, *199*(2), 465–502. https://doi.org/10.1016/j.jcp.2003.12.023

Puckett, E. G. (1991). A volume-of-fluid interface tracking algorithm with applications to computing shock wave refraction. In *proceedings of the Fourth International Symposium on Computational Fluid Dynamics* (pp. 933–938). Mathematical Sciences Publishers.

Qi, Y., Lu, J., Scardovelli, R., Zaleski, S., & Tryggvason, G. (2019). Computing curvature for volume of fluid methods using machine learning. *Journal of Computational Physics*, *377*, 155–161. https://doi.org/10.1016/j.jcp.2018.10.037

Raissi, M., Wang, Z., Triantafyllou, M. S., & Karniadakis, G. E. (2019). Deep learning of vortex-induced vibrations. *Journal of Fluid Mechanics*, *861*, 119–137. https://doi.org/10.1017/jfm.2018.872

Rider, W. J., & Kothe, D. B. (1998). Reconstructing volume tracking. *Journal of Computational Physics*, *141*(2), 112–152. https://doi.org/10.1006/jcph.1998.5906

Roenby, J., Bredmose, H., & Jasak, H. (2016). A computational method for sharp interface advection. *Royal Society Open Science*, *3*(11), 160405. https://doi.org/10.1098/rsos.160405

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint, arXiv:1609.04747.

Rushdi, M. A., Rushdi, A. A., Dief, T. N., Halawa, A. M., Yoshida, S., & Schmehl, R. (2020). Power prediction of airborne wind energy systems using multivariate machine learning. *Energies*, *13*(9), 2367. https://doi.org/10.3390/en13092367

Rushdi, M. A., Yoshida, S., Watanabe, K., & Ohya, Y. (2021). Machine learning approaches for thermal updraft prediction in wind solar tower systems. *Renewable Energy*, *177*, 1001–1013. https://doi.org/10.1016/j.renene.2021.06.033

Rushdi, M. A., Yoshida, S., Watanabe, K., Ohya, Y., & Ismaiel, A. (2024). Deep learning approaches for power prediction in wind–solar tower systems. *Energies*, *17*(15), 3630. https://doi.org/10.3390/en17153630

Scheufler, H., & Roenby, J. (2019). Accurate and efficient surface reconstruction from volume fraction data on general meshes. *Journal of Computational Physics*, *383*, 1–23. https://doi.org/10.1016/j.jcp.2019.01.009

Seetharam, K., Shrestha, S., & Sengupta, P. (2020). Artificial intelligence in cardiac imaging. *US Cardiology Review*, *13*(2), 110–116. https://doi.org/10.15420/usc

Seide, F., & Agarwal, A. (2016). CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 2135–2135). Association for Computing Machinery. https://dl.acm.org/doi/abs/10.1145/2939672.2945397

Thorp, H. H. (2023). ChatGPT is fun, but not an author. *Science*, *379*(6630), 313.

Van Engelen, J. E., & Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine Learning*, *109*(2), 373–440. https://doi.org/10.1007/s10994-019-05855-6

Yadav, M., Perumal, M., & Srinivas, M. (2020). Analysis on novel coronavirus (COVID-19) using machine learning methods. *Chaos, Solitons & Fractals*, *139*, 110050. https://doi.org/10.1016/j.chaos.2020.110050

Youngs, D. L. (1982). Time-dependent multi-material flow with large fluid distortion. In *Numerical methods for fluid dynamics* (pp. 273–486). Academic Press.

Zafeiris, S., & Papadakis, G. (2024). An overset interpolation algorithm for multi-phase flows using 3D multiblock polyhedral meshes. *Computers and Mathematics with Applications*, *161*, 155–173. https://doi.org/10.1016/j.camwa.2024.02.048

Zalesak, S. T. (1979). Fully multidimensional flux-corrected transport algorithms for fluids. *Journal of Computational Physics*, *31*(3), 335–362. https://doi.org/10.1016/0021-9991(79)90051-2

Zhao, Y., & Chen, H.-C. (2017, April). A new coupled level set and volume-of-fluid method to capture free surface on an overset grid system. *International Journal of Multiphase Flow*, *90*, 144–155. https://doi.org/10.1016/j.ijmultiphaseflow.2017.01.002

Appendices

Appendix 1. Computational techniques for evaluating indicator function integrals

The computational domain $\Omega = \Omega^1 \cup \Omega^2$ consists of two regions separated by an interface ($\mathbf{x} \in \Gamma$) of finite thickness. A smooth indicator function $H(\mathbf{x})$ is introduced to describe the region occupied by fluid 1 ($\mathbf{x} \in \Omega^1$) and its transition to the complementary region ($\mathbf{x} \in \Omega^2$). The indicator function is defined as:

$$H(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in \Omega^1, \\ 0, & \mathbf{x} \in \Omega^2, \\ 0 < \phi < 1, & \mathbf{x} \in \Gamma. \end{cases} \tag{A1}$$

The average value of the indicator function over a subdomain Ω_i is expressed as:

$$\bar{\phi}_i = \frac{1}{|\Omega_i|} \int_{\Omega_i} H(\mathbf{x})\, \mathrm{d}\mathbf{x}. \tag{A2}$$

The indicator function $H(\mathbf{x})$ is constructed using the distance function $\Psi(\mathbf{x}, y)$, which is derived from the geometric profile of the interface. Examples include configurations such as a radial star equation (6) or the initial interface geometry used in benchmark problems. The indicator function is then defined as:

$$H(\mathbf{x}) = \begin{cases} 1, & \Psi(\mathbf{x}, y) \le 0, \\ 0, & \text{otherwise}. \end{cases} \tag{A3}$$

For example, when the interface is represented as a circle of radius R centered in the square computational domain $[0, 1] \times [0, 1]$, the indicator function $H(\mathbf{x})$ becomes:

$$H(\mathbf{x}) = \begin{cases} 1, & \sqrt{(x - 0.5)^2 + (y - 0.5)^2} \le R, \\ 0, & \text{otherwise}. \end{cases} \tag{A4}$$

Several methods are available for initializing the volume fraction field based on a prescribed interface profile, including the Vofi library (Bnà et al., 2016), which leverages implicit functions and Gauss–Legendre integration to accurately compute volume fractions in Cartesian grids. In this study, two approaches are considered for evaluating the integral of the indicator function over a subdomain Ω_i:

A.1. Sampling-based method

This method involves sampling multiple points within each computational cell, calculating the distance function $\Psi(\mathbf{x}, y)$ at these points, and determining the corresponding volume fraction. Figure A.1 illustrates a computational grid near the interface, where sampling points are distributed within cells intersecting the interface. The volume fraction is estimated as the ratio of red samples (representing fluid 1) to the total number of samples in each cell. With a sufficiently large number of sampling points, the volume fraction can be computed with high accuracy. Numerical experiments were performed to identify the optimal number of sampling points needed to ensure accurate and reliable volume fraction estimation.

A.2. Tanh-based method

In this approach, the hyperbolic tangent function is applied to the distance function to smooth the transition across the interface. The modified indicator function is expressed as:

$$H_{\tanh}(\mathbf{x}) = \frac{1}{2}\left(1 + \tanh\left(\frac{\Psi(\mathbf{x}, y)}{\epsilon}\right)\right), \tag{A5}$$

where ϵ is a parameter controlling the sharpness of the transition. For sharp interfaces, a small value such as $\epsilon \approx 10^{-6}$ is typically used. The integration of this smooth indicator function is then performed using Gauss quadrature. Numerical experiments were conducted to determine the optimal number of quadrature points required for accurate computation of the volume fraction.

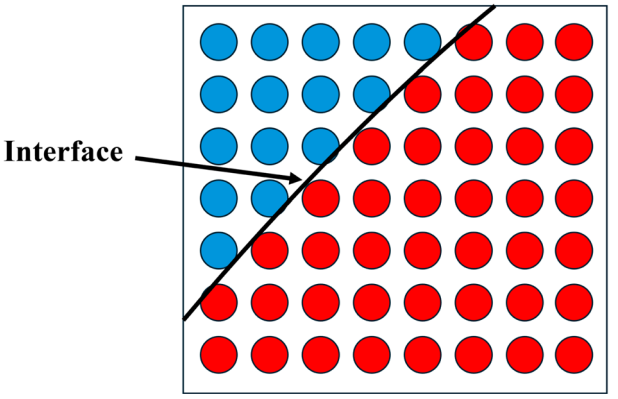


Figure A.1. Computational grid near the interface (black line), separating the red (fluid 1) and blue (fluid 2) regions. Sampling points are used to estimate the volume fraction.

Both methods have been validated against benchmark problems and have been shown to provide effective and accurate volume fraction computations based on the indicator function.

Appendix 2. Machine learning methods

Machine learning (ML) and deep learning (DL) methods are sub-fields of artificial intelligence (AI). They have gained a lot of research attention in the past few years because of their outstanding capabilities to model nonlinear input–output relationships for many tasks like classification or regression problems. These methods can be used to solve multivariate problems with a very large number of input variables or features.

ML and DL methods have been applied in many fields in recent years. They were, mainly, fruitfully in:

- *Computer vision*: tasks like object/pattern recognition, image classification, and tracking are almost well solved with high accuracy. This has a huge impact on self-driving cars application (Gupta et al., 2021).
- *Natural Language Processing (NLP)*: tasks like sentiment analysis, machine translation, speech recognition, and chatpots. This part is evolving fast as we see how advanced voice assistants like Siri or Alexa are, also the human-like interlocutor ChatGPT (Thorp, 2023). NLP is used also in the educational field for students' performance prediction tasks (Fateen & Mine, 2021).

While the knowledge is being accumulated and the ML and DL models get more sophisticated, more fields benefit from this. Healthcare fields benefit a lot by using ML/DL for medical diagnosis, drug discovery, and prediction of life-threatening conditions, as well as dealing with the novel coronavirus (COVID-19) (Yadav et al., 2020). Engineering fields, also, applied ML/DL in the energy field for power prediction (Rushdi et al., 2020, 2021) and many other applications.

The ML/DL methods are currently included in hardware-optimized software libraries (Li et al., 2020) such as Scikit Learn (Pedregosa et al., 2011), TensorFlow (Abadi et al., 2016), Keras (Chollet *et al.* , 2015), Pytorch (Paszke et al., 2017), and CNTK (Seide & Agarwal, 2016). This led to the growing number of fields in which ML/Dl was applied.

As shown in Figure A.2, the ML/DL methods or tasks can be categorized into the following four categories (Géron, 2019) based on the nature of learning supervision:

- *Supervised learning* (Hastie et al., 2009a): The training data fed to the algorithm is labelled. The typical supervised learning tasks are regression and classification.
- *Unsupervised learning* (Hastie et al., 2009b): The training data is not labelled. The typical tasks are clustering, anomaly detection, and dimensionality reduction.
- *Semi-supervised learning* (Van Engelen & Hoos, 2020): These algorithms deal with partially labelled data. Typically, a few labelled data and a lot of unlabelled data.
- *Reinforcement learning* (Arulkumaran et al., 2017): The closest category to AI. The learning system is called agent, typically a robot, which can observe the environment, make a decision, and perform actions to get rewards.

Based on these categories, supervised learning is the way to formulate the tackled problem and to do the current task as the data is labelled. Also, we need to predict the orientation angle based on the values of input features, so this is a regression task.

In supervised learning methods, pairs of input variables $\mathbf{x}$ and output variables y are used to learn an approximation of the input–output mapping $y = f(\mathbf{x})$. The learning process seeks to optimize some objective function, such that for new unlabelled data samples, corresponding outputs are accurately predicted. For example, one-dimensional linear regression is the problem of fitting a line $y = ax + b$ to a number of labelled points $\{\mathbf{x}, y\}_{i=1}^{n}$ such that a least-square error function is minimized.

Appendix 3. Deep learning overview

This appendix provides a concise overview of the neural network (NN) and convolutional neural network (CNN) representations, along with the tuning strategy employed for the NN architecture.

A.3. NN overview

Artificial neural network (ANN) models have garnered growing research attention due to their exceptional ability to model nonlinear input-output relationships. Generally, ANNs emulate the functioning of biological neural networks found in the human brain. A 'neuron' in an ANN functions similarly to a biological neuron, serving as a mathematical construct that gathers and processes information based on a predefined architecture.

A neural network primarily consists of layers of interconnected nodes. Figure A.3 illustrates the neural network model employed for predicting orientation angle, featuring two hidden layers. Each node in this network model functions as a perceptron, which is essentially utilized to execute multiple linear regression. The perceptron channels the input signals into an activation function, which determines the learning model's output, accuracy, and computational efficiency. Activation functions can be designed as either linear or nonlinear to encapsulate the complexity of the predicted function. Figure A.4 presents a schematic representation of a single perceptron, encompassing linear summation and nonlinear activation operations.

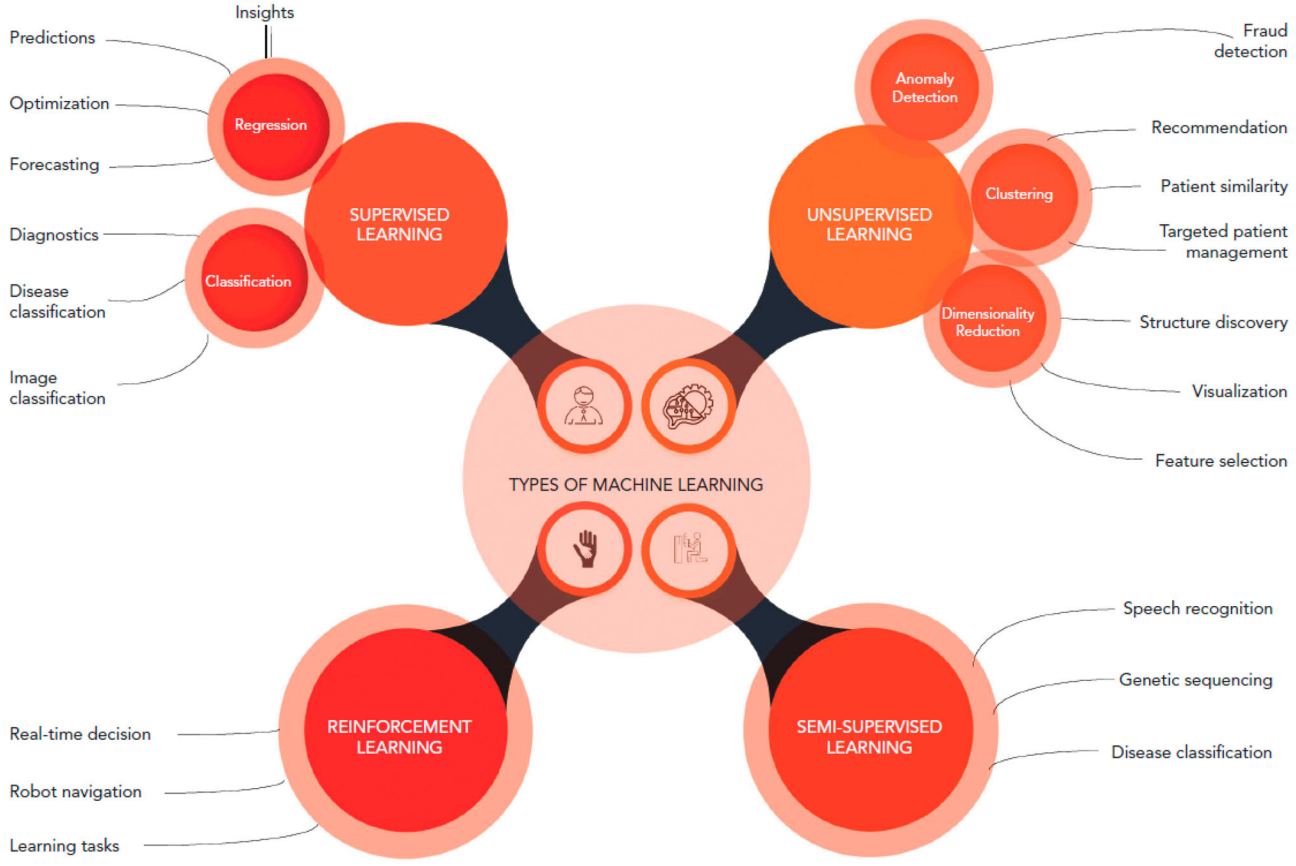


Figure A.2. Main categories of machine learning problems with key applications (Seetharam et al., 2020).

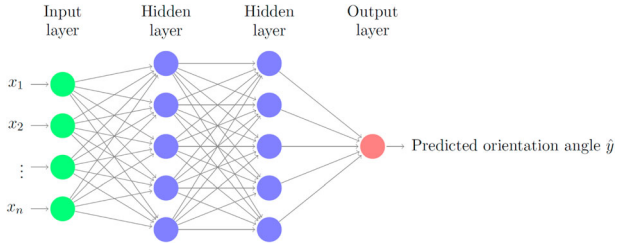


Figure A.3. A NN schematic for output prediction with two hidden layers (adapted from Rushdi et al. (2024)).

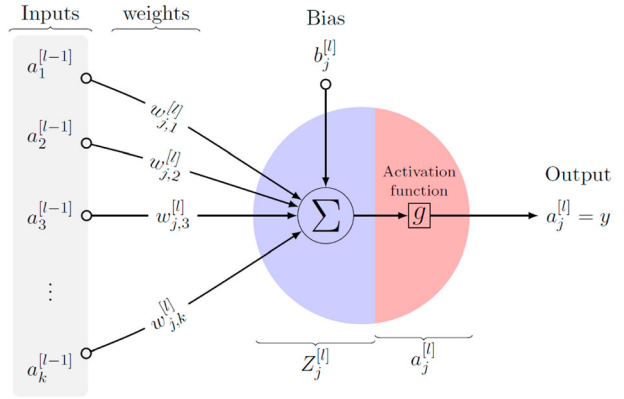


Figure A.4. A single neuron representation of the *j*th node in the *l*th layer of a NN architecture (Rushdi et al., 2024).

Table A.1. Neural network weight update rules.

Step	Equation
Forward propagation	$Z_j^{[l]} = W_j^{[l]T} \cdot A^{[l-1]} + b_j^{[l]}$ $A_j^{[l]} = g(Z_j^{[l]}) = \hat{y}$
Loss function	$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} f(\hat{y}^{(i)}, y^{(i)})$ $= \frac{1}{2m}(\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2}(A^{[l]} - Y)^2$
Backward propagation	$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial A}\frac{\partial A}{\partial Z}\frac{\partial Z}{\partial w}$ $= (A^{[l]} - Y)\left(\frac{\partial A}{\partial Z}\right)A^{[l-1]}$ $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial A}\frac{\partial A}{\partial Z}\frac{\partial Z}{\partial b}$ $= (A^{[l]} - Y)\left(\frac{\partial A}{\partial Z}\right)(1)$
Gradient descent	$w := w - \alpha\frac{\partial J}{\partial w}$ $b := b - \alpha\frac{\partial J}{\partial b}$

As previously noted, in a multi-layered perceptron (MLP) or neural network, perceptrons or neurons are organized into interconnected layers. The input layer gathers input patterns, while the output layer produces the predicted classification or regression results. In our study, the utilized network is designed

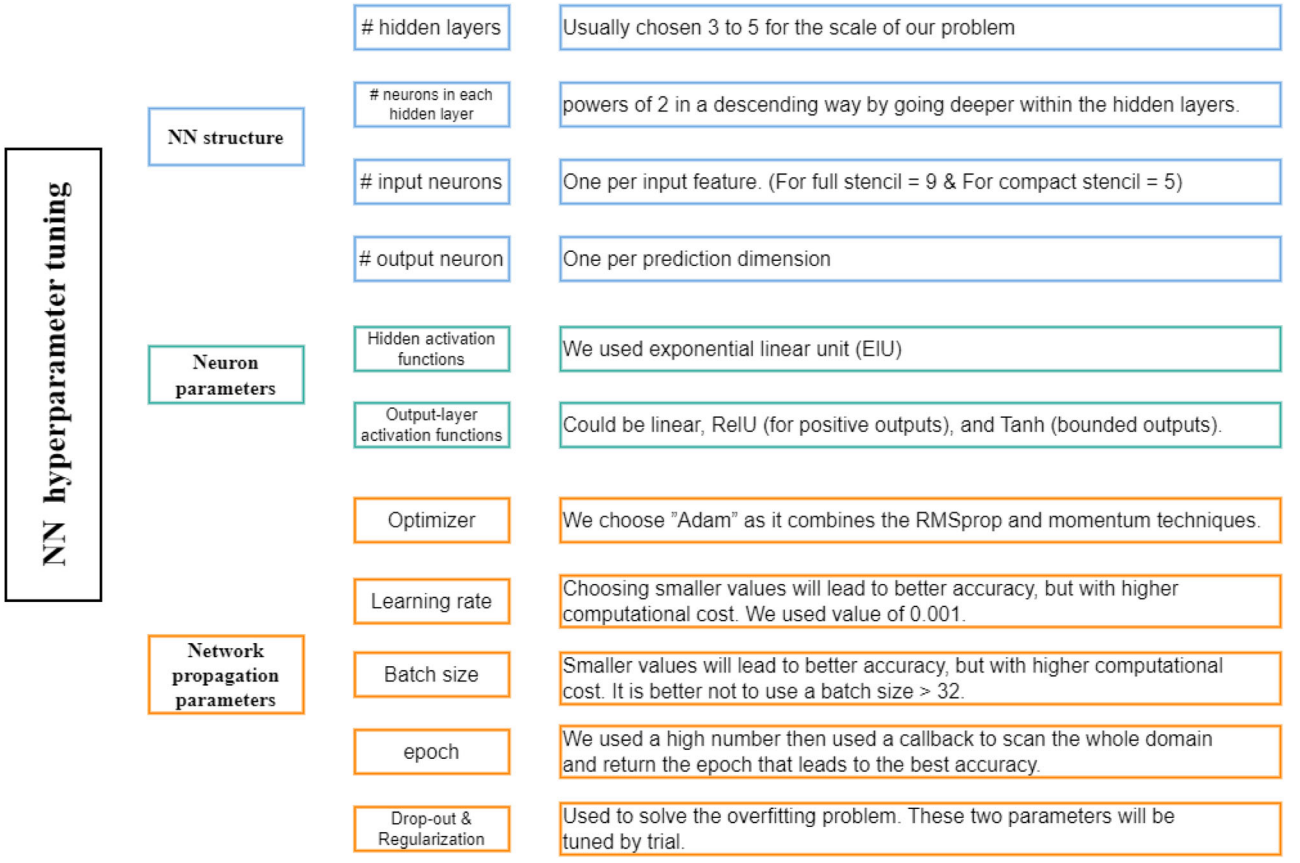


Figure A.5. Summary of NN hyper-parameter tuning.

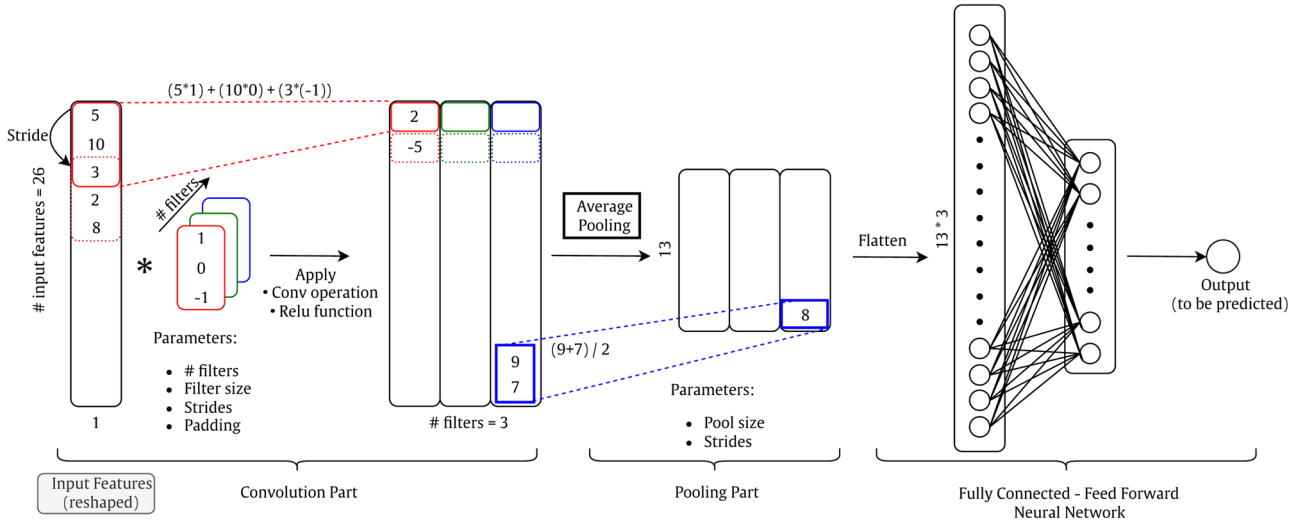


Figure A.6. Illustration of the primary components of convolutional neural networks (ConvNets). The numerical values are provided solely as examples to demonstrate convolution and pooling operations (Rushdi et al., 2024).

to predict the orientation angle. The weights and biases within the hidden layers are meticulously fine-tuned to minimize the network's margin of error. The hidden layers primarily extract the salient features of the input data that are most indicative of the orientation angle. The weight update rules for a single jth neuron in the lth layer (over m samples) can be derived as shown in Table A.1.

Then the gradient-descent rule is applied iteratively to update the network parameters until convergence. Note: $\frac{\partial A}{\partial Z}$ varies based on the activation function used.

Neural Network has many parameters to be tuned. A summary of these hyper-parameters and our strategy/choices are shown in Figure A.5.

A.4. CNN overview

Convolutional neural networks (CNN or ConvNet) are a class of artificial neural networks frequently employed for classification tasks in image processing/recognition applications. Nonetheless, 1-D ConvNet could be employed for regression problems. CNN performs exceptionally well with pictures, speech, and audio signal inputs. ConvNets are composed of a Conv component, a pooling component, and a fully connected feed-forward neural network. The Conv component is the fundamental building block of ConvNets, and it contains three hyperparameters (a) Number of filters, which affects the depth of the output, (b) Stride, which is the distance that the filter/kernel moves over the input matrix, (c) Zero-Padding, which used when the filters do not fit the input matrix.

To introduce non-linearity to the model, a ReLU transformation is applied after each convolution operation. Then, to reduce the volume size, a pooling component is applied. Finally, the volume is flattened and input to a feed-forward neural network, as shown in Figure A.6.