A Deep Learning Algorithm for Piecewise Linear Interface Construction (PLIC)

Mohammadmehdi Ataei,[1,2] **Erfan Pirmorad,**[2] **Franco Costa,**[3] **Sejin Han,**[4] **Chul B Park,**[2] **Markus Bussmann**[2]

[1]Vector Institute for Artificial Intelligence, 661 University Ave Suite 710, Toronto, ON M5G 1M1, Canada
[2]University of Toronto, 5 King's College Rd, Toronto, ON M5S 3G8, Canada
[3]Autodesk, Inc., 259-261 Colchester Rd., Kilsyth, VIC. 3137, Australia
[4]Autodesk, Inc. 2353 North Triphammer Rd., Ithaca, NY 14850, USA
{ataei, pirmorad}@mie.utoronto.ca, {franco.costa, sejin.han}@autodesk.com, {park, bussmann}@mie.utoronto.ca

Abstract

Piecewise Linear Interface Construction (PLIC) is frequently used to geometrically reconstruct fluid interfaces in Computational Fluid Dynamics (CFD) modeling of two-phase flows. PLIC reconstructs interfaces from a scalar field that represents the volume fraction of each phase in each computational cell. Given the volume fraction and interface normal, the location of a linear interface is uniquely defined. For a cubic computational cell (3D), the position of the planar interface is determined by intersecting the cube with a plane, such that the volume of the resulting truncated polyhedron cell is equal to the volume fraction. Yet it is geometrically complex to find the exact position of the plane, and it involves calculations that can be a computational bottleneck of many CFD models. However, while the forward problem of 3D PLIC is challenging, the inverse problem, of finding the volume of the truncated polyhedron cell given a defined plane, is simple. In this work, we propose a deep learning model for the solution to the forward problem of PLIC by only making use of its inverse problem. The proposed model is up to several orders of magnitude faster than traditional schemes, which significantly reduces the computational bottleneck of PLIC in CFD simulations.

Introduction

Multiphase flows (flows consisting of more than one component) are ubiquitous in nature and are of interest in a variety of practical applications, and as such a significant amount of work has been conducted on the computational modeling of such flows (e.g. Balachandar and Eaton 2010; Rothman and Zaleski 1994; Wang et al. 2016). One of the challenges of multiphase flow modeling is to track interfaces between different phases. Among numerous Eulerian and Lagrangian interface tracking schemes, the Volume-of-Fluid (VOF) Eulerian method is widely used (Hirt and Nichols 1981).

In the VOF method, as shown in figure 1a, the volume fraction of one the two phases within each computational cell is represented by a scalar field α. For example, for a liquid-gas system, $\alpha = 1$ or $\alpha = 0$ in each computational cell that is occupied by the liquid or gas phase, respectively,
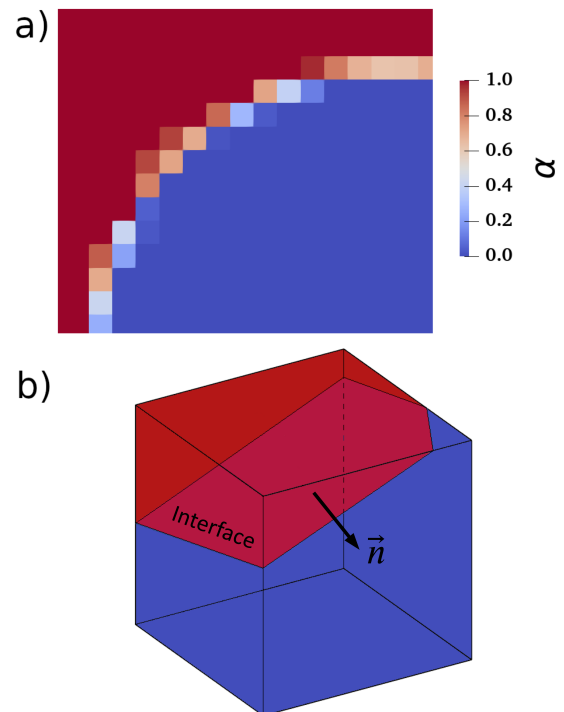
Figure 1: a) An example of a VOF scalar field for a liquid-gas system. $\alpha = 1$ in liquid cells, $\alpha = 0$ in gas cells, and $0 < \alpha < 1$ in interface cells. b) Interface reconstruction using PLIC.

and $0 < \alpha < 1$ in interface cells. An advection equation is solved for α to track the location of interface throughout the simulation. There are two major categories of VOF schemes: algebraic and geometric (Lafaurie et al. 1994). Geometric VOF schemes provide more accurate fluid-fluid interface reconstruction compared to algebraic schemes, but require more complicated geometrical calculations. Of these methods, Piecewise Linear Interface Construction (PLIC) is frequently used to geometrically reconstruct an interface as a line (2D) or plane (3D) (Youngs 1982). As shown in figure 1b, for a cubic cell, PLIC linearly approximates an interface, such that the volume of the resulting truncated polyhedron
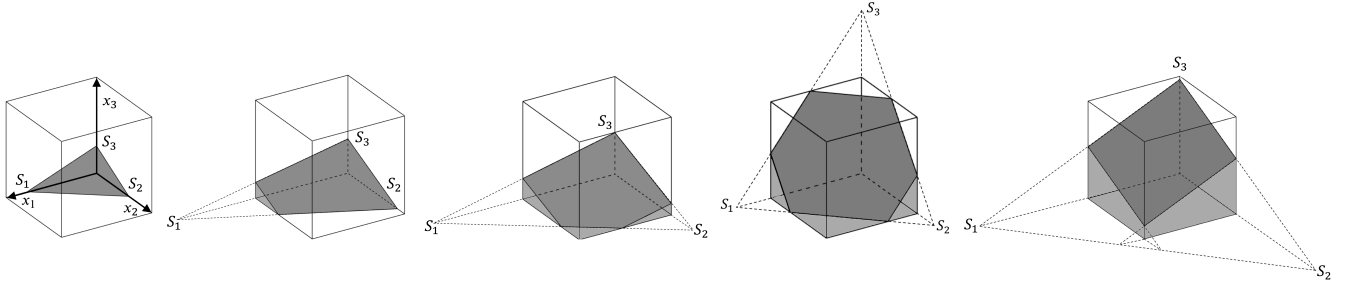
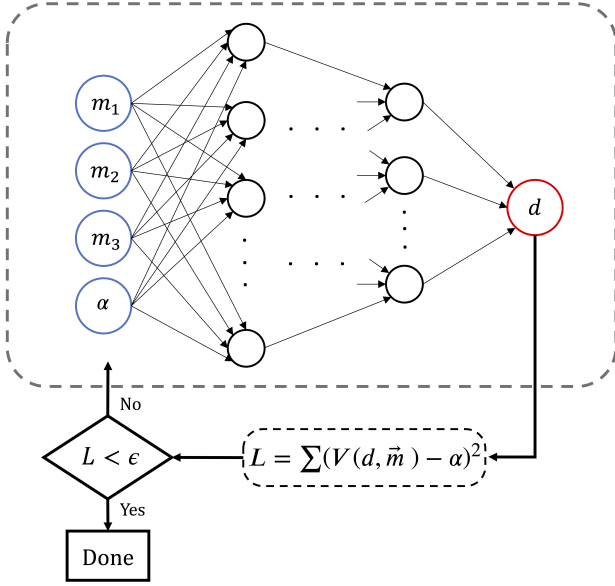Figure 2: All possible cases for intersection of a plane and a unit cube.

Figure 3: DNN architecture.

cell is equal to the volume fraction α, where the plane normal unit vector $\vec{n}$ is calculated as the gradient of the volume fraction scalar field, as $\vec{n} = -\frac{\nabla \alpha}{||\nabla \alpha||}$ (Kothe and Mjolsness 1992). Eliminating symmetrical cases, the number of possible cube-plane intersections can be reduced to the five scenarios shown in figure 2, where $\alpha \leq 0.5$ (for $\alpha > 0.5$, α is changed to $1 - \alpha$ to match one of the five cases). The constant of the plane d is the smallest distance from the plane to the origin of the Cartesian coordinate that conforms with the cubic cell edges, which must be calculated. The problem of computing d is known as the forward problem of PLIC (Scardovelli and Zaleski 2000).

To date, many studies have used PLIC for interface reconstruction on structured and unstructured grids, and also for related calculations such as calculating the distance between two interfaces, and have proposed improvements to reduce the implementation complexity and increase its efficiency (López et al. 2005; Ataei et al. 2021b; Dai and Tong 2019; Ataei et al. 2021a; Nahed and Dgheim 2020; Scheufler and Roenby 2019). A simple analytical solution can be used

to solve PLIC in 2D for a square cell. However, analytical solutions in 3D for a cubic cell include multiple *if-else* conditions, and require more sophisticated operations (e.g. dealing with the inverse of third degree polynomials and complex numbers) (Scardovelli and Zaleski 2000; Yang and James 2006). For example, Scardovelli and Zaleski (Scardovelli and Zaleski 2000) found an analytical solution to the forward problem by inverting the inverse problem of PLIC (i.e. finding the volume fraction of a polyhedron formed given a plane with $\vec{n}$ and d), which is a simpler geometric calculation. This analytical solution includes a number of conditional statements to select a solution from a set of equations, corresponding to different cases of the intersection of a plane with a cube. This selection can slow down a CFD solver, and is not easily parallelizable. To reduce the computational cost, alternative iterative and approximation techniques have been developed to calculate d (Skarysz, Garmory, and Dianat 2018; Rider and Kothe 1998; Kawano 2016).

Machine Learning (ML) has attracted considerable attention in the computational sciences, including CFD. Over the last decade, ML techniques have been used extensively in fluid mechanics to solve a variety of problems, including model discovery, modeling of reduced order, processing of experimental data, shape optimization, and flow control (Brunton, Noack, and Koumoutsakos 2020; Raissi, Yazdani, and Karniadakis 2020; Brenner, Eldredge, and Freund 2019; Singh, Medida, and Duraisamy 2017; Pfaff et al. 2020; Nikolaev, Richter, and Sadowski 2020; D'Elia et al. 2020; Yoon, Melander, and Verzi 2020). Previously, we have published an alternative neural network based technique to perform PLIC calculations, by training a neural network with a large synthetic dataset of PLIC solutions (Ataei et al. 2021a). In this work, we propose a direct solution to the forward problem of PLIC using a fully connected Deep Neural Network (DNN). Unlike the previous work, we demonstrate that a DNN can solve the forward problem given only the inverse problem of PLIC, without requiring a synthetic dataset. We show that, in addition to its simplicity, this approach significantly outperforms other traditional PLIC implementations.

Methodology

We solve the forward problem of PLIC by converting it into an optimization problem that can be solved using a DNN. In

the forward problem, we must find d such that the volume V of a polyhedron cell is equal to the volume fraction α in a unit cubic cell:

$$V = \alpha \tag{1}$$

This equation can be converted into an optimization problem for a DNN by defining a loss function L as:

$$L = \sum_{batch} (V(d, \vec{m}) - \alpha)^2 \tag{2}$$

where $\vec{m} = (m_1, m_2, m_3)$ are the components of $\vec{n}$ along the axes shown in figure 2 (Jafari, Shirani, and Ashgriz 2007), calculated as:

$$\begin{aligned} m_1 &:= \min(|n_x|, |n_y|, |n_z|) \geq 0 \\ m_3 &:= \max(|n_x|, |n_y|, |n_z|) > 0 \\ m_2 &:= |n_x| + |n_y| + |n_z| - m_1 - m_3 \geq 0 \end{aligned} \tag{3}$$

As shown in figure 3, we design a multilayer perceptron (MLP) neural network with fully connected layers, comprised of an input layer, H hidden layers each with N neurons, and an output layer, to solve this optimization problem. The network outputs d based on $\vec{m}$ and α as inputs. If the difference between $V(d, \vec{m})$, calculated based on the predicted d (the output of the DNN), and the known α is sufficiently minimized, then the forward PLIC problem is solved by the DNN. One of the advantages of this technique is that (unlike conventional methods for training a DNN), there is no need for synthetic data to train the DNN, as the DNN automatically finds the solution to the forward PLIC problem by minimizing this loss function.

Fortunately, formulating V (i.e. the inverse problem of PLIC) in terms of the plane parameters $(d, \vec{m})$ is a straightforward geometrical problem. The volume under the plane can be found by calculating the volume of the pyramid formed between the intersection of the plane with the three coordinates (S_1, S_2, S_3), and then subtracting the volume of the parts that extend beyond the cube (see figure 2). Such calculations result in a set of equations for V for each of the five cases, as follows (Scardovelli and Zaleski 2000):

$$V = \frac{1}{\lambda} \cdot \begin{cases} f_1(d, \vec{m}) & 0 \leq d \leq m_1 \\ f_2(d, \vec{m}) & m_1 \leq d \leq m_2 \\ f_3(d, \vec{m}) & m_2 \leq d \leq \min(m_1 + m_2, m_3) \\ f_4(d, \vec{m}) & m_3 \leq d \\ f_5(d, \vec{m}) & \min(m_1 + m_2, m_3) \leq d \leq m_3 \end{cases} \tag{4}$$

where:

$$\begin{aligned} f_1 &= d^3 \\ f_2 &= d^3 - \delta_1^3 \\ f_3 &= d^3 - \delta_1^3 - \delta_2^3 \\ f_4 &= d^3 - \delta_1^3 - \delta_2^3 - \delta_3^3 \\ f_5 &= 6 m_1 m_2 \left(d - \tfrac{1}{2}(m_1 + m_2)\right) \end{aligned} \tag{5}$$

where:

$$\begin{aligned} \delta_1 &= d - m_1 \\ \delta_2 &= d - m_2 \\ \delta_3 &= d - m_3 \\ \lambda &= 6\, m_1 m_2 m_3 \end{aligned} \tag{6}$$

This DNN is implemented in the Pytorch deep learning library (Paszke et al. 2019). We use ReLU activation functions for all layers except for the output layer, which is a linear activation function. The gradient of the custom loss function in Eq. 2 is calculated using the Pytorch automatic differentiation *autograd* package.

For the inputs of the DNN $\vec{m}$ is uniformly sampled where all three coordinates are positive. Similarly, α is sampled uniformly between 0 and 0.5. We used 5000 randomly shuffled sample points in total, split into three datasets: 70% training, 20% test, and 10% validation. The model was trained on the training dataset, the validation dataset was used to ensure that the model was not overfitting during the training, and the test dataset was used to evaluate the final accuracy of the model.

The network weights and biases were randomly initialized at the beginning using Xavier initialization (Glorot and Bengio 2010). The results presented here are the average of six runs with six different random seeds for network initialization. The weights and biases were tuned using the Adam Optimizer (Kingma and Ba 2015) with a batch size of 64 for 512 epochs.

Results

Evaluation metrics

The training was performed on a PC running Ubuntu 20.04 (Intel i7-8700K, NVIDIA 1080 Ti graphics card (CUDA version 11.0), 32GBs of RAM). It takes about 20 minutes to train each network.

The training progress of a DNN with $H = 1$, $N = 24$ is shown in figure 4. The Mean Squared Error (MSE) of both training and validation datasets decreased during training. The model performance was evaluated against the test dataset. Figure 5 shows the MSE and the maximum absolute error for DNNs with different H and N. Notice the network with $H = 2$, $N = 48$ is the most accurate; however, other networks with fewer neurons and hidden layers also perform on-par or better when compared to PLIC models that approximate PLIC (e.g. Kawano 2016).

To visualize the accuracy of the DNN model, figure 6 plots the value of α based on the value of d predicted by the DNN ($H = 2$, $N = 48$) against the real value of α, for 200 random data points from the test dataset. It can be seen that there is excellent agreement between the predicted α and real α, close to the ideal where all points would lie on the diagonal line.

Speedup

The advantages of using a DNN for PLIC calculations is that it performs much faster than an analytical solution. Moreover, the DNN model can be easily parallelized on both CPUs and GPUs.

Figure 7 shows a comparison of the DNN ($H = 2$, $N = 48$) versus the Scardovelli and Zaleski (Scardovelli and Zaleski 2000) analytical solution. To compare the models, the analytical solution was implemented in C++, and compiled using optimization flag $-O3$ using the GCC compiler version 9.3. The Pytorch DNN model was converted into Torch-
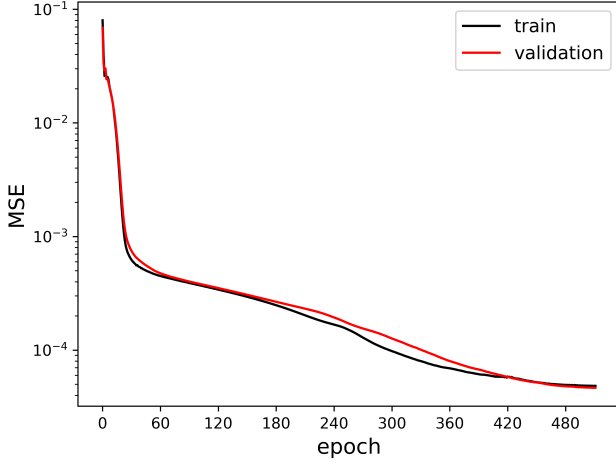
Figure 4: Training progress over 512 epochs with batch size of 64 for a DNN with one hidden layer with 24 neurons with seed = 100.
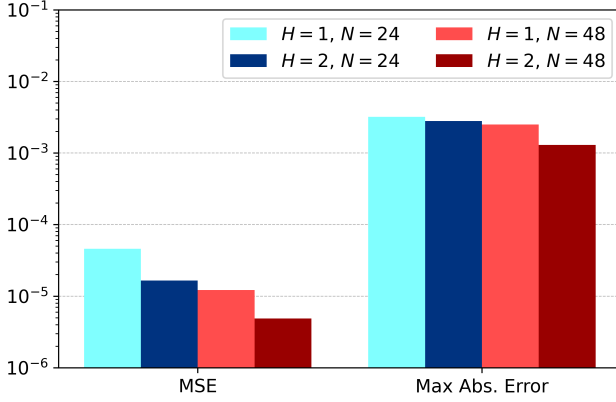

Figure 5: Mean Squared Error (MSE) and maximum absolute error on the test dataset, for different DNN models with different H and N.

script and imported into C++ as well. The `std::chrono` library and Pytorch's Profiler were used to measure the execution time of the Scardovelli and Zaleski PLIC solution and the neural model, respectively.

First, we compare the execution time of both models on one CPU core. Since the DNN can perform PLIC calculations in batches, we ran the models for different numbers of cells with randomly and uniformly generated α and $\vec{m}$. On one CPU, the DNN model is 3.4 times faster than the analytical model for 1000 cells. However, as the number of cells increases to 10 million, the speedup gain is only 1.3. This is expected, as the overhead associated with moving the tensors to RAM becomes substantial for large batch sizes. The DNN model accelerates significantly better on a GPU (even taking into account the overhead of moving data to the GPU), and with 10 million cells we achieve more than 41 times speedup. In our previous work, we demonstrated
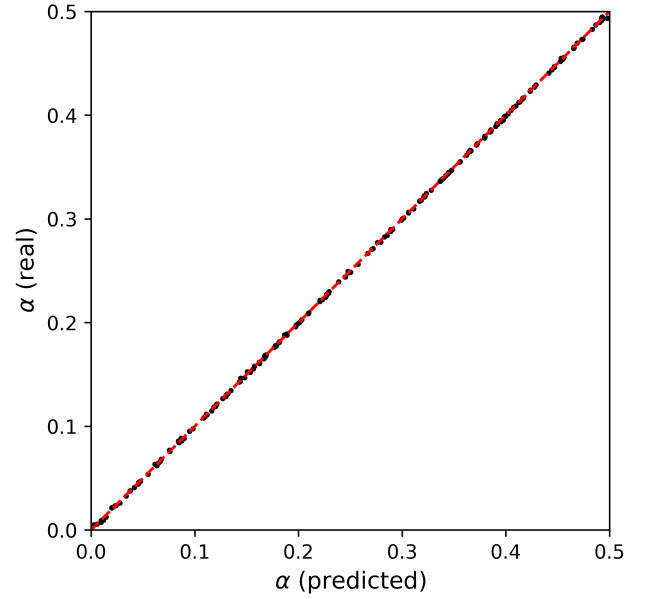

Figure 6: Plots of the DNN predictions vs real α by a DNN with two hidden layers each with 48 neurons (seed = 400). To avoid clutter, only 200 random points are selected from the test dataset.

that, while traditional algorithms such as Scardovelli and Zaleski (Scardovelli and Zaleski 2000) require fewer floating operations than a neural network approach, the neural network model better utilizes the hardware and operates at higher floating operations per second (Ataei et al. 2021a).
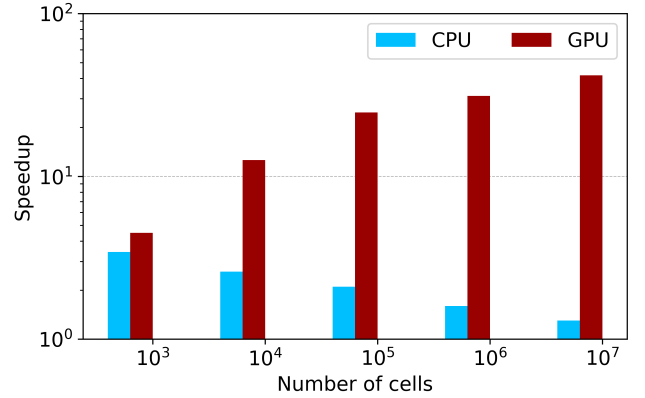

Figure 7: Speedups of the DNN model compared to the analytical solution of Scardovelli and Zaleski (running on one CPU core as the baseline).

Implementation in a CFD Solver

LBfoam (Ataei et al. 2021b) is a free surface lattice Boltzmann solver for foaming simulations. In LBfoam, a disjoining pressure Π between two adjacent bubble interfaces is responsible for stabilizing liquid lamellae (the thin layer of

liquid separating two bubbles). The disjoining pressure is active up to a distance δ_{max}, and is a function of the distance between bubble interfaces δ as:

$$\Pi = \begin{cases} 0 & \delta > \delta_{max} \\ k_\Pi\left(1 - \frac{\delta}{\delta_{max}}\right) & \delta < \delta_{max} \end{cases} \quad (7)$$

where k_Π is a constant. In order to calculate δ, LBfoam reconstructs adjacent bubble interfaces with PLIC.

To verify that the DNN model works as expected in a CFD solver, we replaced the PLIC calculation model in LBfoam with the trained DNN model with $H = 2$ and $N = 48$.

Figure 8 shows simulation results based on the same parameters and at the same timestep. The DNN was used for the simulation on the left, and the analytical PLIC was used on the right. As can be seen, the simulation results are identical. Depending on the number of cells that require interface reconstruction (i.e. the interface cells closer than δ_{max}), the speedup gain for interface construction was between 1.5-3 times.

Conclusions

In this paper, we present a DNN model to solve the forward problem of PLIC. We show that a DNN can learn to solve the PLIC problem without a synthetic dataset, using its inverse problem. The DNN implementation of PLIC is up to several orders of magnitude faster than analytical approaches, for the same accuracy. The DNN implementation is easily portable to various CFD solvers, and can be efficiently accelerated by CPUs and GPUs in parallel. We implemented the DNN model in the LBfoam CFD solver to verify that the model can perform well in a practical setting.

Acknowledgements

References

Ataei, M.; Bussmann, M.; Shaayegan, V.; Costa, F.; Han, S.; and Park, C. B. 2021a. NPLIC: A Machine Learning Approach to Piecewise Linear Interface Construction.

Ataei, M.; Shaayegan, V.; Costa, F.; Han, S.; Park, C. B.; and Bussmann, M. 2021b. LBfoam: An open-source software package for the simulation of foaming using the Lattice Boltzmann Method. *Computer Physics Communications* 259: 107698.

Balachandar, S.; and Eaton, J. K. 2010. Turbulent dispersed multiphase flow. *Annual Review of Fluid Mechanics* 42(1): 111–133.

Brenner, M. P.; Eldredge, J. D.; and Freund, J. B. 2019. Perspective on machine learning for advancing fluid mechanics. *Phys. Rev. Fluids* 4: 100501.

Brunton, S. L.; Noack, B. R.; and Koumoutsakos, P. 2020. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics* 52(1): 477–508.

Dai, D.; and Tong, A. Y. 2019. Analytical interface reconstruction algorithms in the PLIC-VOF method for 3D polyhedral unstructured meshes. *International Journal for Numerical Methods in Fluids* 91(5): 213–227.

D'Elia, M.; Karniadakis, G.; Pang, G.; and Parks, M. 2020. Nonlocal physics-informed neural networks - A unified theoretical and computational framework for nonlocal models. In *AAAI Spring Symposium: MLPS*.

Glorot, X.; and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, 249–256. JMLR Workshop and Conference Proceedings.

Hirt, C.; and Nichols, B. 1981. Volume of fluid (VOF) method for the dynamics of free boundaries. *Journal of Computational Physics* 39(1): 201–225.

Jafari, A.; Shirani, E.; and Ashgriz, N. 2007. An improved three-dimensional model for interface pressure calculations in free-surface flows. *International Journal of Computational Fluid Dynamics* 21(2): 87–97.

Kawano, A. 2016. A simple volume-of-fluid reconstruction method for three-dimensional two-phase flows. *Computers & Fluids* 134-135: 130–145.

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kothe, D. B.; and Mjolsness, R. C. 1992. RIPPLE - A new model for incompressible flows with free surfaces. *AIAA Journal* 30(11): 2694–2700.

Lafaurie, B.; Nardone, C.; Scardovelli, R.; Zaleski, S.; and Zanetti, G. 1994. Modelling merging and fragmentation in multiphase flows with SURFER. *Journal of Computational Physics* 113(1): 134–147.

López, J.; Hernández, J.; Gómez, P.; and Faura, F. 2005. An improved PLIC-VOF method for tracking thin fluid structures in incompressible two-phase flows. *Journal of Computational Physics* 208(1): 51–74.

Nahed, J.; and Dgheim, J. 2020. Estimation curvature in PLIC-VOF method for interface advection. *Heat and Mass Transfer* 56(3): 773–787.

Nikolaev, A.; Richter, I.; and Sadowski, P. 2020. Deep learning for climate models of the Atlantic ocean. In *AAAI Spring Symposium: MLPS*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An imperative style, high-performance deep learning library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32, 8026–8037. Curran Associates, Inc.
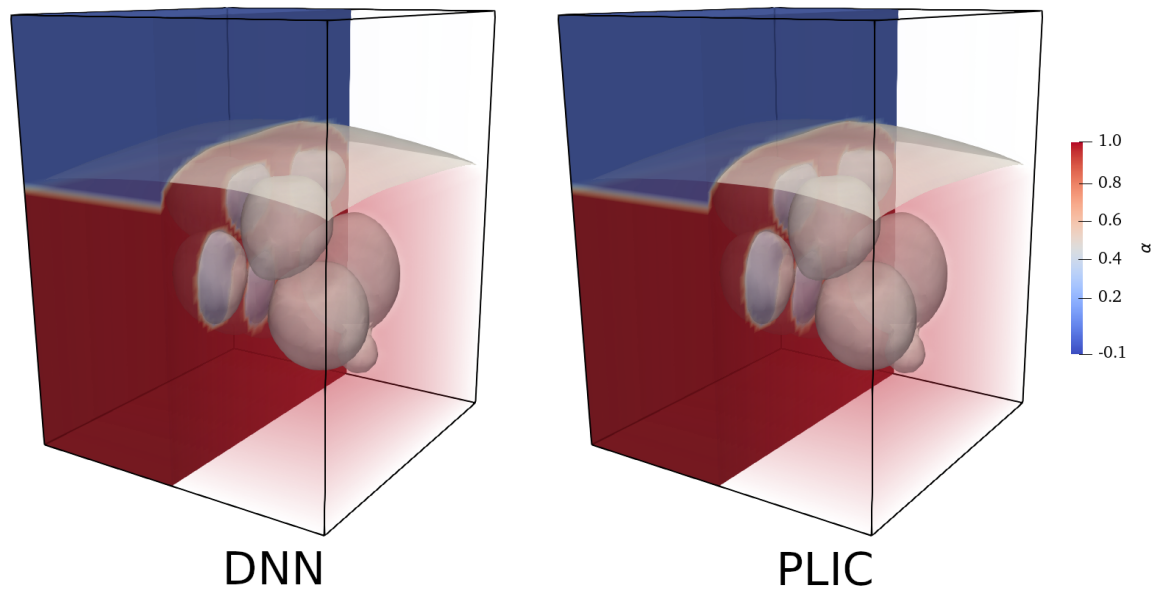
Figure 8: 3D foaming simulation using the LBfoam software. The disjoining pressure between bubbles is calculated using the DNN (left) and the analytical PLIC (right).

Pfaff, T.; Fortunato, M.; Sanchez-Gonzalez, A.; and Battaglia, P. W. 2020. Learning mesh-based simulation with graph networks.

Raissi, M.; Yazdani, A.; and Karniadakis, G. E. 2020. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science* 367(6481): 1026–1030.

Rider, W. J.; and Kothe, D. B. 1998. Reconstructing volume tracking. *Journal of Computational Physics* 141(2): 112–152.

Rothman, D. H.; and Zaleski, S. 1994. Lattice-gas models of phase separation: interfaces, phase transitions, and multiphase flow. *Rev. Mod. Phys.* 66: 1417–1479.

Scardovelli, R.; and Zaleski, S. 2000. Analytical relations connecting linear interfaces and volume fractions in rectangular grids. *Journal of Computational Physics* 164(1): 228–237.

Scheufler, H.; and Roenby, J. 2019. Accurate and efficient surface reconstruction from volume fraction data on general meshes. *Journal of Computational Physics* 383: 1–23.

Singh, A. P.; Medida, S.; and Duraisamy, K. 2017. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA Journal* 55(7): 2215–2227.

Skarysz, M.; Garmory, A.; and Dianat, M. 2018. An iterative interface reconstruction method for PLIC in general convex grids as part of a Coupled Level Set Volume of Fluid solver. *Journal of Computational Physics* 368: 254–276.

Wang, Z.-B.; Chen, R.; Wang, H.; Liao, Q.; Zhu, X.; and Li, S.-Z. 2016. An overview of smoothed particle hydrodynamics for simulating multiphase flow. *Applied Mathematical Modelling* 40(23): 9625–9655.

Yang, X.; and James, A. J. 2006. Analytic relations for reconstructing piecewise linear interfaces in triangular and tetrahedral grids. *Journal of Computational Physics* 214(1): 41–54.

Yoon, H.; Melander, D.; and Verzi, S. J. 2020. Permeability prediction of porous media using convolutional neural networks with physical properties. In *AAAI Spring Symposium: MLPS*.

Youngs, D. L. 1982. Time-dependent multi-material flow with large fluid distortion. *Numerical Methods for Fluid Dynamics* .