AIAA AVIATION Forum
June 15-19, 2020, VIRTUAL EVENT
AIAA AVIATION 2020 FORUM

10.2514/6.2020-3058

Towards a hybrid computational strategy based on Deep Learning for incompressible flows

E. Ajuria*
ISAE-SUPAERO / CERFACS, Toulouse, 31400/31100, France

A. Alguacil† and M. Bauerheim‡
ISAE-SUPAERO, Toulouse, 31400, France

A. Misdariis§ and B. Cuenot¶
CERFACS , Toulouse, 31100, France

E. Benazera‖
Jolibrain , Toulouse, 31000, France

The Poisson equation is present in very different domains of physics and engineering. In most cases, this equation can not be solved directly and iterative solvers are used. For many solvers, this step is computationally intensive. In this study, an alternative resolution method based on neural networks is evaluated for incompressible flows. A fluid solver coupled with a Convolutional Neural Network is developed and trained on random cases with constant density to predict the pressure field. Its performance is tested in a plume configuration, with different buoyancy forces, parametrized by the Richardson number. The neural network is compared to a traditional Jacobi solver. The performance improvement is considerable, although the accuracy of the network is found to depend on the flow operating point: low errors are obtained at low Richardson numbers, whereas the fluid solver becomes unstable with large errors for large Richardson number. Finally, a hybrid strategy is proposed in order to benefit from the calculation acceleration while ensuring a user-defined accuracy level. In particular, this hybrid CFD-NN strategy, by maintaining the desired accuracy whatever the flow condition, makes the code stable and reliable even at large Richardson numbers for which the network was not trained for. This study demonstrates the capability of the hybrid approach to tackle new flow physics, unseen during the network training.

I. Nomenclature

CNN	=	Convolutional Neural Network
CPU	=	Central Processing Unit
GPU	=	Graphics Processing Unit
LES	=	Large Eddy Simulation
MAC	=	Marker-and-Cell
MLP	=	MultiLayer Perceptron
MSE	=	Mean Squared Error
NN	=	Neural Network
PCG	=	Preconditioned Conjugate Gradient
PDE	=	Partial Differential Equation
RBFNN	=	Radial Basis Function Neural Networks
Ri	=	Richardson Number

*PhD Student, DAEP / CFD, ekhi.ajuria@cerfacs.fr
†PhD Student, DAEP, Antonio.ALGUACIL-CABRERIZO@isae-supaero.fr
‡Associate Professor, DAEP, Michael.BAUERHEIM@isae-supaero.fr and AIAA Member Grade
§Researcher, CERFACS, misdariis@cerfacs.fr
¶Project manager, CERFACS, cuenot@cerfacs.fr
‖CEO, Jolibrain, emmanuel.benazera@jolibrain.com

II. Introduction

In recent years, the use of Machine Learning has widely spread through different scientific communities, especially on computer vision. Neural Networks have a long history, starting in the late nineteen fifties when Rosenblatt [1] developed the first Artificial Neural Network called *Perceptron*. Convolutional Neural Networks (CNN) first saw the light in 1998, when LeCun et al. [2] successfully implemented a network to extract spatial 2D features in images. This last decade, Machine Learning was marked with a rapid development towards new applications, such as engineering, medicine and science, mainly due to the development of open-source Deep Learning frameworks, as well as the improvement of Graphical Processing Units (GPU). For instance, in 2012, Krizhevsky successfully implemented a Deep Convolutional Network that ran on two GPUs [3], obtaining a significant amelioration on the ImageNet classification problem [4]. This opened the path to the application of CNNs for large fluid mechanics problems which usually require simulations on multiple computational units (CPUs or GPUs).

In fluid mechanics, the governing equations are non-linear coupled PDEs, for which solutions contain a large range of time and spatial scales, and exhibit complex behaviors, such as turbulence, bifurcation and transition. For years, the CFD community has improved the numerical schemes [5] and flow modeling to achieve high accuracy while increasing exponentially the computational cost of fluid simulations. In that context, Machine Learning was introduced recently to the fluid mechanics domain, in order to benefit from the potential reduction of computational cost. The role of Machine Learning in fluid dynamics is reviewed by Brunton et al. [6], who distinguish the efforts of Machine Learning into two main categories: (i) extracting features from flows and (ii) modeling flow dynamics.

On the one hand, the feature extraction by Machine Learning strategy focuses on model reduction techniques. One of the principal applications corresponds to LES, using Neural Networks (NN) to infer high-resolution structures from poorly resolved domains, as in Fukami et al. [7] who used a CNN for the reconstruction of turbulent flows, successfully respecting the energy spectrum. Some attempts have also been made using GAN's (Generative Adversarial Adversarial Networks) [8, 9], and a similar strategy is found for the reconstruction of velocity fields, from experimental PIV images [10].

On the other hand, the flow modeling strategy intends to accelerate fluid solvers based on Machine Learning techniques. First attempts to solve Partial and Ordinary Differential Equations using Neural Networks, date back to the nineteen nineties where Dissanayake et al. [11], Gonzalez et al. [12] or Lagaris et al. [13] focused on treating the NN as a continuous function which approximates the solution by taking the variable that it depends on as inputs. Thus, they transformed the classical finite element solving procedure into an unconstrained optimization problem. Dissayanake, for example, tested a simple four-layer MultiLayer Perceptron (MLP), to solve two benchmark problems, one of them being the Poisson equation with Dirichlet boundary conditions. A similar strategy was followed by Lagaris et al., who tested Neural Networks on different PDEs with both Neumann and Dirichlet boundary conditions. Gonzalez et al. slightly modified this strategy by approximating the right-hand-side of PDEs using a MLP. They inputted spatial values of the state variables in order to calculate temporal evolutions and tested their network in a Kuramoto-Shivashinsky equation. These methods could achieve high accuracy with a rather low number of parameters. However, they were limited by the available computational power, and their solution was only applicable to specific cases. With the rise of the available computational resources, these methods were refined and further developed. An overview of those methods can be found in Kumar and Yadav's work [14], who studied the evolution of MLP and Radial Basis Function Neural Networks (RBFNN). RBFNNs were already developed in the late '80s (Moody et al. [15]), and correspond to fully connected Neural Networks that compute the Eulerian distance between the input and a certain prototype in each neuron. Therefore, each neuron instead of operating a linear operation $a = w_i X + b_i$, contains a bell function where the distance from a reference is measured. The work of Choi et al. [16] compared both methods, revealing a benefit from RBFNNs, although it states the general limitations of the NN approximations for more complex problems.

Nowadays, due to the increase of computational power, notably by the development of more efficient hardware (GPU and memory storage), the development of an open-source Machine Learning community and the creation of user-friendly Machine Learning frameworks, the resolution of PDEs has considerably evolved. One of the main changes in the solving procedure is found in the work of Raissy et al. [17]. In previous works, PDE resolution was mainly treated as a black box, where NN could not benefit from any prior physical knowledge. Consequently, Raissy et al. [17] implemented physical constraints into the neural network: compared with classical approaches where networks are optimized by minimizing the error between prediction and a reference target solution, here the network is trained by minimizing the residual of the physical equation. To compute these gradients, authors use the automatic differentiation [18] technique. This technique called *physics informed neural networks* (PINN), is tested on continuous-time models, solving the Burgers equation, and on discrete-time models, with the Shrödinger equation. Advantages of PINN are twofold: (i) it enforces

some conservative laws of physics inside the network, and (ii) the network could theoretically be trained without a CFD or experiment database, yet the latter is not yet fully explored in the literature. Authors extended their study to the incompressible Navier-Stokes equation [19, 20], analyzing among others the flow around a cylinder. Additional works, such as Chen et al. [21] centered their efforts on solving Ordinary Differential Equations (ODE) with various network architectures, while Long et al. [22] first introduced convolutional neural networks to approximate differential operators with convolutions, which allows the extraction of a physical knowledge from the data.

Despite all these works, further developments are still required to propose deep learning strategies in real fluid solvers: this is the main objective of the present paper, focusing on incompressible Navier-Stokes solvers. For incompressible flows, the most computationally expensive part corresponds to the resolution of the Poisson equation, which has to be resolved to update the velocity field. This step is traditionally solved through iterative methods such as Jacobi or Preconditioned Conjugate Gradient (PCG) [23, 24], that although being accurate, are computationally expensive. For most applications, these methods have to be truncated, thus a trade-off has to be found between accuracy and the available computational budget. Recently, advanced methods have been proposed to further accelerate the Poisson equation resolution, for example using a multigrid strategy [25]. Yet standard methods are now mature, and a breakthrough is needed to further accelerate incompressible flow solvers. The main idea of the present study is to solve the Poisson equation with NNs in order to accelerate fluid simulations. The idea was first developed by Yang et al. [26] using MLP. Xiao et al. [27] used a CNN to solve the Poisson equation on large computational domains combining a physical and a supervised MSE loss function. Similarly, Özbay [28], resolve the Poisson equation with two coupled convolutional neural networks. The first one is dedicated to the resolution of the Poisson equation with an arbitrary source term, whereas the second tackles the homogeneous problem with various boundary conditions. Among these works, the present study follows the work of Tompson et al. [29], who first employed a CNN to solve the Poisson equation in a real CFD solver in 2016, achieving an acceptable accuracy and stability. They used an innovative learning strategy, similar to PINN's, which enforces the conservation of mass as a loss function. Whereas classical applications of deep learning, such as classification problems, cannot incorporate prior knowledge of the problem into the network, the use of physical equations to explicitly add physics into deep neural networks is highly valuable, yet under-exploited. In particular, it opens the path to new computational strategies where both CNN and classical numerical methods are hybridized: the main objective of this paper is to propose such a hybrid incompressible fluid solver where a CNN is coupled with a Jacobi method.

Tompson's work was developed for computer vision, with an objective of visually acceptable results, but no guarantee of quantified accuracy. In the present work, the control of accuracy is a main driver of the method. A fluid solver, based on Tompson's code and Thuerey's open-source code, Mantaflow [30], briefly described in Section III is implemented and coupled with a CNN. The training phase and architecture of the deep neural network are presented in Section IV. Compared with Tomson's work, the architecture has been improved using a multi-scale network, but the training phase is similar, with simulations on random geometries and constant density. In Section V, the resulting coupled fluid solver will be tested on buoyant plume simulations, performed at various flow conditions, defined by the Richardson number (Ri). In these simulations, the density is variable, allowing the network to be tested in configurations ($Ri > 0$) unseen during the training phase performed at $Ri = 0$. Finally, a hybrid strategy is proposed in section VI, which benefits from the calculation acceleration due to the CNN, while guaranteeing a user-defined level of accuracy. Since the maximum error is observed near walls, the hybrid approach is challenged on additional test cases of the same plume impinging a cylinder.

III. Equations and Resolution Procedure

A. Fluid Equations

The fluid simulator used in this study is based on Mantaflow [30], which solves the 2D and 3D incompressible *Navier-Stokes* equations:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{1}{\rho_0} \nabla p = \frac{1}{\rho_0} \boldsymbol{f} + \nu \nabla^2 \boldsymbol{u} \tag{1}$$

$$\nabla \cdot \boldsymbol{u} = 0 \tag{2}$$

where $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}, t)$ is the velocity field, $p = p(\boldsymbol{x}, t)$ the pressure field, ν the kinematic viscosity, $\boldsymbol{f}$ the external force field and ρ_0 the density, which under the incompressibility assumption, is considered constant, leading to the simplified

continuity equation (2), stating a zero-divergence velocity field. For a complete introduction to these equations and to fluid dynamics for fluid animations, the reader can refer to Bridsons [31].

The external forces vector includes all the body forces acting on the fluid, which correspond to gravity. The present study focuses on buoyancy-driven flows such as plumes. To introduce buoyancy effects in an incompressible formulation, the Boussinesq approximation is applied, which is valid for flows where the density differences are small compared to the overall mean density. In these cases, the density can be considered constant for all terms in the momentum Eq. (1), except for the gravity, which is then expressed as:

$$\boldsymbol{f} = \rho \boldsymbol{g} = (\rho_0 + \Delta\rho)\boldsymbol{g} \tag{3}$$

where the density has been separated into two components: ρ_0 is the background density, and $\Delta\rho$ is the density variation. Considering that gravity only applies in the vertical j-th direction ($\boldsymbol{g}$=$g\boldsymbol{j}$) it is included in a new pressure term p' = p + $\rho_0 g y$, which injected into Eq. (1) and using the decomposition of Eq. (3) gives the following momentum equation:

$$\frac{D\boldsymbol{u}}{Dt} = -\frac{1}{\rho_0}\nabla p' + \frac{\Delta\rho}{\rho_0}\boldsymbol{g} + \nu\nabla^2\boldsymbol{u} \tag{4}$$

where $\frac{D\boldsymbol{u}}{Dt} = \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}$ is the material derivative of the velocity field. From the Boussinesq hypothesis, taking the variable density continuity equation, an additional equation for the density is obtained, which can be written as:

$$\frac{1}{\rho_0}\frac{D\Delta\rho}{Dt} = 0 \tag{5}$$

This equation states that density stays constant along a streamline, i.e. it is advected as a passive scalar. Therefore, the implemented solver, based on the Mantaflow solver developed by Thuerey [30], solves Eqs. (2), (4) and (5), and will be coupled to a CNN for a fast inference of the Poisson solution.

B. Numerical discretization

Following Tompson's work, a classical finite difference scheme is applied on a Marker-and-Cell (MAC) staggered grid [32] where the velocities are taken at the cell faces, whereas the scalar variables are taken at the cell centers. This type of grid allows the direct calculation of velocity derivatives at the cell centers, and facilitates the implementation of boundary conditions, particularly at walls, which impose the object velocity to the surrounding fluid in the normal direction of the boundary. In the particular case of a static wall, the boundary condition imposes a zero velocity along the wall-normal.

The continuity and momentum equations can be solved with a standard operator splitting method as explained by Bridson [31]. A summary of the entire procedure is found in Fig. 1, where two main steps can be identified:

- Advection and External force addition
- Pressure projection

In the first step, the density and the velocity fields are advected using a semi-Lagrangian Maccormack method [33], which is unconditionally stable and second-order accurate. Then, the external forces are applied (noted as f_{ext} in Fig. 1) with a forward Euler integration scheme. Note that the viscous term is treated as an external force. After this step, slip boundary conditions are imposed at walls to the resulting velocity field $\boldsymbol{u}^*$, which at this point does not satisfy the continuity equation (the divergence is not null).

The second step can be viewed as a correction of the velocity fields to enforce a non-divergent field $\boldsymbol{u}^*$, thus ensuring mass conservation. This is achieved by solving:

$$\frac{\partial \boldsymbol{u}^*}{\partial t} = \frac{1}{\rho_0}\nabla p' \tag{6}$$

which is solved with a forward Euler scheme:

$$\boldsymbol{u} \simeq \boldsymbol{u}^* - \Delta t \frac{1}{\rho_0}\nabla p \tag{7}$$

By applying the divergence to both sides of Eq. (6) the Poisson equation (8) is obtained. It is solved to calculate the pressure gradient appearing in Eq. (7). The wall boundary conditions are again applied after the velocity update.

$$\frac{\Delta t}{\rho_0}\nabla^2 p = -\nabla \cdot \boldsymbol{u}^* \tag{8}$$

Equation 8 states that the Laplacian of the pressure is equal to the divergence field of the non-corrected velocity $\boldsymbol{u}^*$. The resolution of this equation is the focus of this article. Indeed, this step is by far the most computationally expensive, as it can take up to 80% of the calculation time in certain cases, as it is usually difficult to efficiently parallelize. Using a Neural Network has the potential to reduce this calculation time.

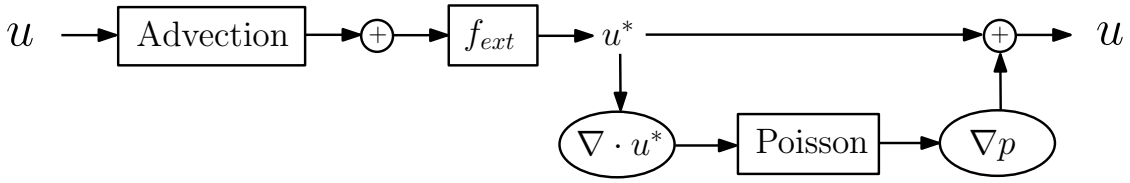


Fig. 1 Illustration of the operator splitting process in the incompressible fluid solver.

IV. Solving the Poisson equation

A. Jacobi method

The Poisson equation is usually solved with iterative algorithms such as Jacobi or PCG. As mentioned by Tompson [29], these solvers are usually truncated, particularly for real-time applications for computer vision or large fluid problems. Therefore, a compromise must be found between computational cost and residual convergence level.

The Jacobi method is an extensively studied [24, 34] iterative optimization method used to solve linear systems, which dates back to 1845 [35]. To solve a linear system of type $AX = B$, the Jacobi method decomposes the A matrix as $A = D + R$, being D a diagonal matrix and R a matrix where all the diagonal elements are zero. Substituting A and rearranging the equation terms, Eq. (9) is obtained:

$$X = D^{-1}(B - RX) \tag{9}$$

The Jacobi method solves Eq. (9) iteratively by introducing the sequence X_t, convergin towards the solution X:

$$X_t = D^{-1}(B - RX_{t-1}) \tag{10}$$

where the solution at iteration X_t depends on the solution at the previous Jacobi iteration X_{t-1}. In this study, the Poisson equation is solved by a neural network and the results will be compared with a Jacobi method implemented on GPUs. Comparison with other solvers such as PCG or Multigrid methods implemented on GPUs will be studied in future work, as although accurate, they are difficult to parallelize, thus limiting their efficiency on modern hardware containing GPUs.

B. Deep convolutional neural network architecture

Compared with the previous Jacobi method, considered as the reference, this study investigates the use of CNN to infer rapidly the Poisson solution. The CNN implemented in this study performs linear operations on the input data X ($a = w_i X + b_i$), combined with non-linear activation functions, in order to minimize a target function, which depends on the calculated output. Particularly, Convolutional Neural Networks do not apply the linear operations to the whole grid at a time, but rather use a kernel of reduced size k, sliding through the domain, thus performing the operations "by zones". For Neural Networks, each linear operation is followed by an activation function, which introduces non-linearity into the network. These activation functions are needed to ensure the universal approximation theorem [36]. The trainable parameters (w_i and b_i) of the network are optimized by minimizing a loss function using a gradient descent algorithm. For a further explanation of Machine Learning and Neural Networks, the reader is referred to Goodfellow's work [37].

In the present work, the specific MultiScale architecture proposed by Matthieu [38] is used. It is based on convolutional kernels on three branches, as depicted in Fig. 2. The network features two input channels: the divergence of the non-corrected velocity field and an occupancy field which identifies borders and obstacles; and one output: the

pressure field. This architecture extracts information from different spatial scales and sequentially concatenates it. The input of size n^2 is first down-sampled to a quarter of its original size ($n^2_{1/4}$), "creating" an image which contains information about the larger spatial structures. This image is then processed through a series of convolutional layers (each followed by an activation function) generating a final image of size ($n^2_{1/4}$). This image is then interpolated to twice its size, which creates an image with a half-scale ($n^2_{1/2}$) then concatenated to the half version of the original images (of size $n^2_{1/2}$). The process is then repeated at the half-scale, finally concatenating the generated image to the original inputs and undergoing a final series of convolutions, which at the end outputs the pressure field (p' of size n^2).

The model counts with 418640 trainable parameters found in the convolutional layers, which define the weights and biases of the filters of the convolutional layers. The quarter-scale branch is the smallest among the three scales, as it has 4 layers (32, 64, 64 and 32 filters respectively). The half-scale has 6 layers (containing respectively 32, 64, 128, 64, 32 and 1 filter), whereas the full scale has 7 layers (with 32, 64, 128, 64, 32, 8, 1 filter). No activation function is used after the last two layers of each scale.

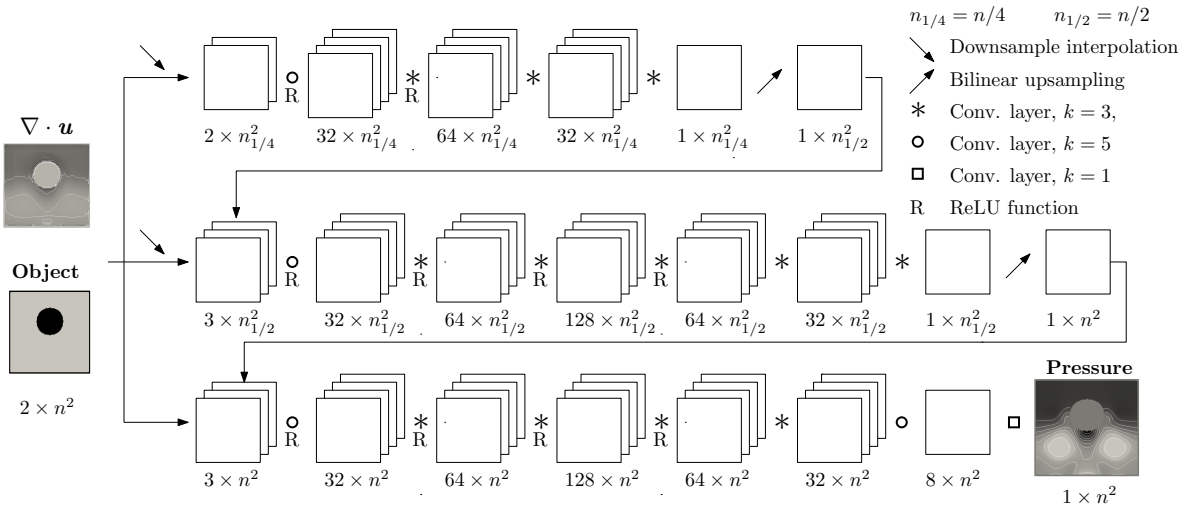


Fig. 2 Multi-Scale Architecture.

C. Loss Function

The neural network is parametrized by multiple weights and biases to optimize, as described in Section IV.B. In practice, the optimization of the neural network is achieved by backpropagating the error gradients. The error is defined through a function comparing the network output with a ground-truth, called "loss function". Its choice is critical in Machine Learning problems. For typical classification problems, the labels or categories are usually well defined (e.g. "cat" or "dog" in the ASIRRA test [39]) simplifying the decision to consider the obtained results as correct or incorrect. However, the majority of physical problems are regression problems, where the ground truth is a continuous field, adding an extra dimension to the problem. The Mean Squared Error (MSE) (Eq. (11)) is one of the most used functions for this purpose, as it computes the L2-norm distance of the output to the ground truth. For the particular case of a 2D field of size (m,n), where O corresponds to the output of the neural network and T the targeted field, the loss function is:

$$MSE = \frac{1}{m \cdot n} \sum_{i}^{m} \sum_{j}^{n} \Big\{ O(i,j) - T(i,j) \Big\}^2 \tag{11}$$

When a ground-truth is needed to compute the loss function, the learning process is known as *supervised*, and it corresponds to the majority of cases in image classification and field reconstruction problems. In the particular case of the resolution of the Poisson equation, Yang et al. [26] trained their model using this strategy, comparing their results with pressure fields issued from a traditional linear solver. However, this technique needs the implementation of a conventional CFD solver which might slow down the training procedure as for each new divergence field, a well-resolved pressure field is needed. One major limitation is that MSE-like loss functions have no physical meaning: the error is

averaged over the computational domain (e.g. no particular focus on near-wall errors), and it is difficult to relate to the final objective, which is to ensure mass conservation.

In this study, the same strategy as Tompson is employed to avoid *supervised* learning. The calculated pressure field is used to correct the advected velocity field u^*, which results in a velocity field with a small, ideally zero, divergence. This property is local at each grid point in the corrected velocity field, so it can be used as a measure of the network error. Therefore, an *unsupervised* learning strategy is considered, where the function f_{loss} to minimize is:

$$f_{loss} = \frac{1}{m\,n}\sum_{i}^{m}\sum_{j}^{n}\left\{\nabla\cdot\boldsymbol{u}(i,j)\right\}^2 = \frac{1}{m\,n}\sum_{i}^{m}\sum_{j}^{n}\left\{\nabla\cdot\left[\boldsymbol{u}^* - \frac{1}{\rho_0}\nabla p_{out}(i,j)\right]\right\}^2 \tag{12}$$

where $\boldsymbol{u}$ and $\boldsymbol{u}^*$ correspond to the corrected velocity and the advected velocity (respectively the velocities after and before the Poisson equation resolution and velocity update), and p_{out} to the pressure field outputed by the network. The main advantage of the *unsupervised* learning is that no ground truth pressure field is needed (only p_{out} appears in Eq. (12), as well as $\boldsymbol{u}^*$, which is the network input), which avoids possible errors related to the uncertainty of the reference data.

One possible advantage of this *unsupervised* learning is to make the network more robust to error-propagation by adding a long-term loss. Once the network calculates the pressure field, instead of starting the backpropagation process, the weight and bias values are "frozen" and inputed to the new velocity field into the advection module, continuing the process while the network does not learn anything. After N iterations, the divergence of the corrected velocity field is calculated, which is the measure of the network's error. The number of iterations N can be either 4 or 16, as at the beginning of each timestep, a random sample is drawn from a uniform distribution on the interval [0,1), so that if the drawn sample is smaller than 0.9, N will be equal to 4, otherwise, N will be equal to 16. This procedure makes the NN more robust, as its objective is not only to minimize the divergence in the actual time step but also to keep it low for "future" time steps. This is crucial when using a Poisson solver for fluid mechanics since the error of the Poisson solver could be non-linearly amplified by the advection step. A sketch of this procedure is displayed in Fig. 3.

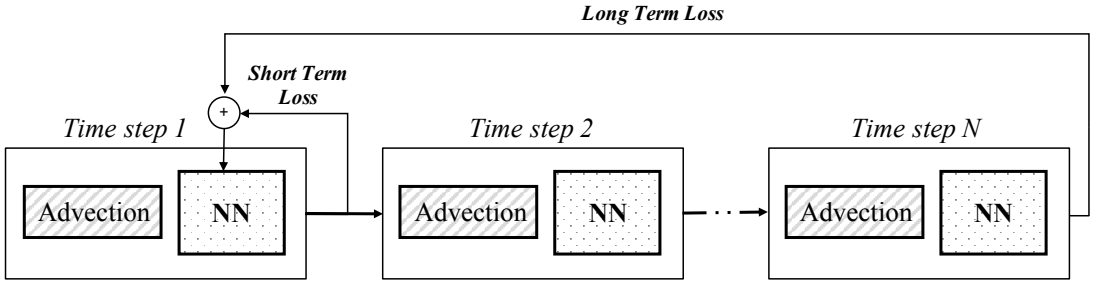


Fig. 3 Long Term Loss Scheme.

D. Dataset

The training dataset is created with the open-source framework Mantaflow [30], mainly used for fluid simulation in computer graphics. It is parallelized in C++ and it is easy to couple with deep learning frameworks, as it runs on GPU and it uses structured uniform grids. Although the training does not actually need a *ground truth*, it enables to cover a wide variety of fluid configurations which will improve the robustness of the neural network. The solver uses a staggered grid discretization and the Poisson equation is solved with PCG.

The dataset corresponds to a series of random velocity fields, with constant density (no density fluctuations are introduced) initialized through a wavelet turbulent noise technique [40]. In close-bounded domains, random geometries (cylinders, squares, ...) are arbitrarily added, translated and scaled. To these fields, divergence sources are added and propagated through time to generate more diverse fields, as seen in Fig. 4, where the propagation of a divergence source is highlighted. Moreover, during training, gravity-like forces of different magnitudes and directions were randomly added in order to ensure the generalization of the learning process. The dataset contains 320 different geometric scenes, each one propagated for 256-time steps (each of 0.1s). Only one over four time steps are recorded, resulting in a dataset

with 64 frame scenes (where each frame is 0.4 s apart). In Machine Learning, the dataset has usually to be divided between the training and the testing dataset, so that the network can be evaluated in some cases that have not been used for training. For the studied case, an analog testing dataset was generated, which is composed of 320 scenes, each with 64 frames.

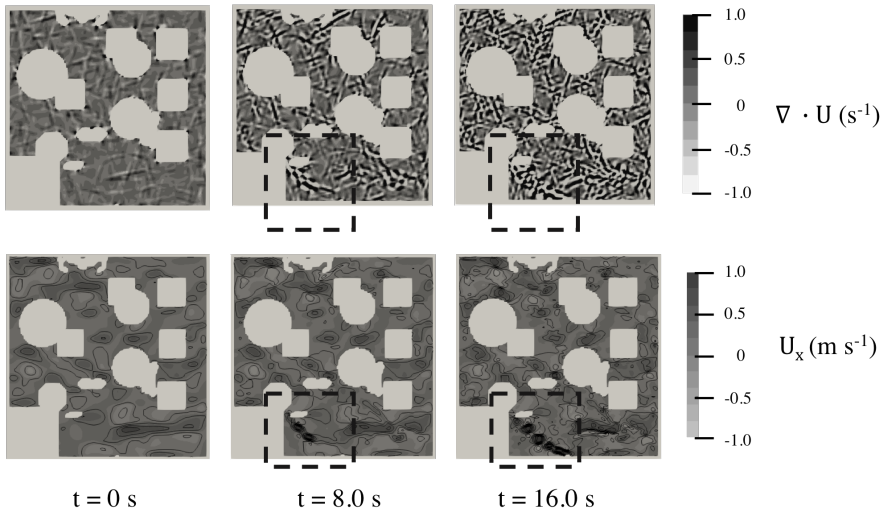


Fig. 4 Divergence and U_x field frames at three instants, of a scene taken from the training dataset. Dashed-line boxes highlight the propagation of a divergence source.

V. Results

The performance of the developed CNN-based fluid solver has to be evaluated on configurations different from the training dataset. This way the network's ability to generalize is tested, ensuring that during the training the network has not *overfitted* *. Here, the solver performance is tested in a plume configuration, where buoyant forces play an important role in the development of the flow.

A. Plume simulations at various Richardson numbers

Tompson's plume test case, described in Fig. 5 is replayed in order to qualitatively and quantitatively evaluate the results obtained with the NN. Plumes are flow structures where a fluid is injected into a quiescent environment. If the injected fluid is less dense, the flow is driven by buoyancy forces. Here, Euler equations are considered, corresponding to an infinite Reynolds number. The plume's behavior is therefore described by only one dimensionless number called the Richardson number, which corresponds to the ratio of buoyant forces to the momentum-driven forces:

$$Ri = \frac{\frac{\Delta\rho}{\rho_0} g L}{U^2} \tag{13}$$

where g corresponds to gravitational acceleration, $\frac{\Delta\rho}{\rho_0}$ to the ratio of the density fluctuations to the background density, L is a characteristic length, here corresponding to the radius of the inlet, and U the velocity of the injected fluid. Note that the Boussinesq approximation is made, thus the density fluctuations should remain small compared to the background density. High Richardson flows are driven by buoyant forces, whereas the flow is driven by momentum for low Richardson numbers (this type of flows are also known as jets). The domain is composed of a squared grid, of size 128 m, and the radius of the inflow is 18.5 m (which will be the characteristic length used to define the Richardson number). The ratio of the density fluctuations to the background density $\frac{\Delta\rho}{\rho_0}$ is 0.01, and the gravity (g) is set at

*In machine learning, *overfitting*. corresponds to the tendency of a network to "learn by heart" the training data-set, thus losing its capability to generalize

0.002 m s^{-2}. As already mentioned, the fluid is here supposed inviscid so that the plume evolution is parametrized by only the Richardson number, which will be modified by modifying the inlet velocity (Tab. 1).

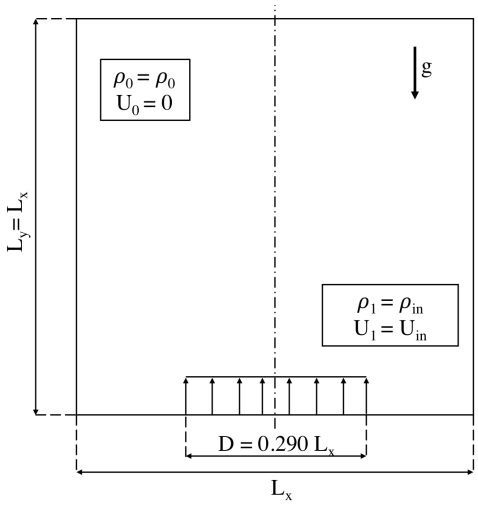


Fig. 5 Plume test case configuration (not to scale).

Table 1 Richardson numbers.

Ri	**0.15**	**0.74**	**100**
U_{inlet} (m/s)	5.00×10^{-2}	2.24×10^{-2}	1.93×10^{-3}

As mentioned in Section IV, during the training, the flows contained in the dataset do not have any density variation, which means that the training was performed at Ri = 0. Therefore, configurations with non-null Richardson number constitute an interesting testing case, as the network has never seen a flow dominated by buoyant forces.

The density evolution according to a non-dimensionalized time $\widetilde{t}$ defined as $t\left[L_y U_{inlet}^{-1}\right]^{-1}$ is shown in Fig. 6 for the three cases of Table 1. For clarity reasons, the evolution of $|\Delta\rho\,| \,/\, \rho_0$ is plotted, thus even if the density differential might seem positive, the injected fluid is indeed lighter.

Figure 6 shows that for the low Richardson number case (Ri = 0.15), the plume evolution is well predicted, as its head propagates at a similar velocity to the Jacobi 200 case. However, some disturbances are found at the plume's head for the Ri = 0.74 case, whereas for the Ri = 100 case the plume's head becomes unphysical due to numerical instabilities. In order to better understand the origin of these discontinuities, the plume head position over time is plotted in Fig. 7 for the Ri = 0.15 case, and the NN is compared to the Jacobi solver with different number of iterations (noted JaX, being X the number of iterations performed at each timestep). As an iterative process, the Jacobi method converges towards a more accurate answer. Therefore, the bigger the number of iterations made by Jacobi, the more accurate the result will be, but the calculation time will consequently increase as well, as the convergence rate of Jacobi is not linear. For the present configuration, the Jacobi solver with 10000 iterations is taken as the "ground-truth" for the plume case. These calculations were performed on a NVIDIA® Tesla® V100 32 Gb of memory, GPU. Obtained computational performances with the CNN and the Jacobi method are reported in Table 2:

Table 2 Computational performance of various algorithms in the plume configuration.

Time	**NN**	**Ja28**	**Ja100**	**Ja200**
T (ms)	1.5	54	194	383
T/ T_{NN}	-	36	130	256
T_p/T_{it}(%)	2	33	64	78

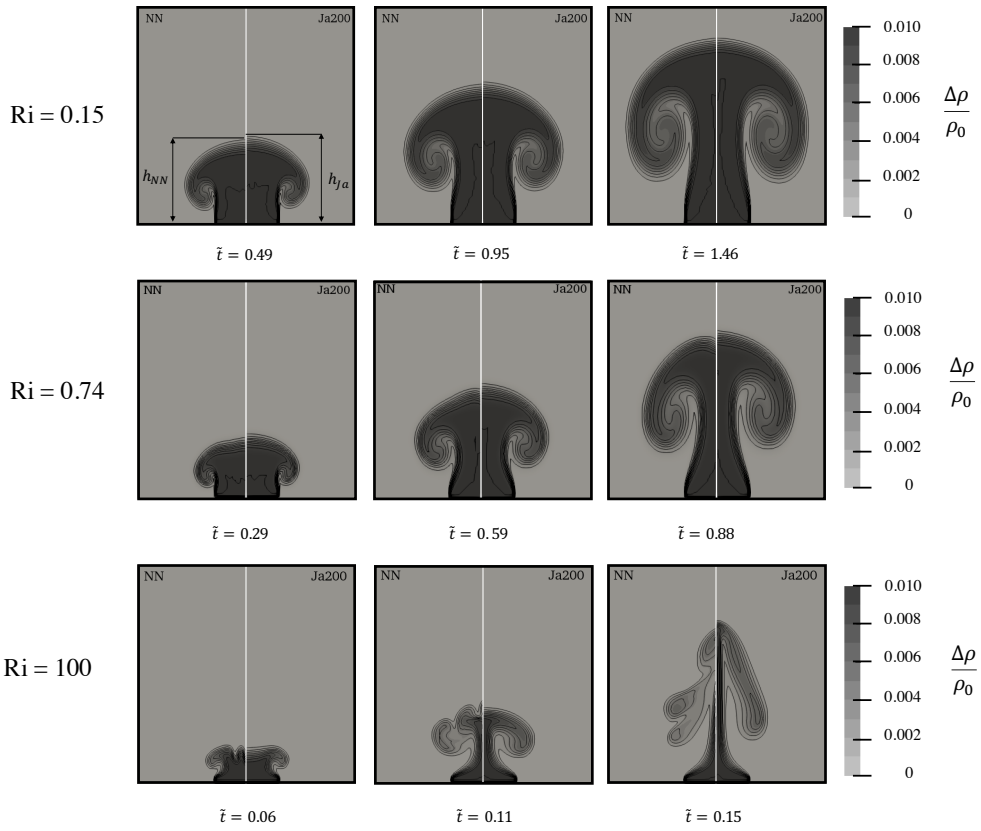


Fig. 6 Plume evolution comparison between the NN and Jacobi 200 for the 3 studied Richardson numbers.

In Table 2, T is the time needed to solve the Poisson equation. The second line indicates the gain of the CNN and the third line gives the relative cost of the pressure correction term. The overall calculation time is strongly reduced when the neural network is used to infer the Poisson solution: for the CNN, the time required for the pressure resolution (T_p) is 2% of the total calculation time of one iteration (T_{it}), whereas it takes up to 78% of T_{it} for the Jacobi 200. For this particular configuration, the Jacobi 200 is around 250 times slower than the CNN. Note however that modern algorithms such as PCG and multi-grid approaches probably outperform the Jacobi method as well, yet the comparison of the present neural network with an optimized PCG on GPU will be studied in future work.

For a quantitative comparison of the solutions, the dimensionless head position $\widetilde{h}$ (h /L_y) is displayed in Fig. 7, which corresponds to the most advanced point respect to the total length of the domain. The relative error corresponds to the relative difference between each solver's head position and the position of the Jacobi 10000. Results show that the Neural Network accuracy is similar to the Jacobi 200 solver, leading to an error below 10 % in the final positions. A significant error is observed with the CNN when $\widetilde{t} < 0.15$, which then decreases down to the Jacobi 200 level at around $\widetilde{t} = 0.5$. The error then slowly increases again, reaching a value of around 5 % for the final steps. A more detailed analysis is required to understand the cause of this large error for long times. Indeed, two scenarios are possible: (i) the error is generated at large times because a specific fluid mechanism not correctly captured, or (ii) the error at early times is amplified non-linearly at longer times by the advection step. To discriminate between these scenarios, the divergence normalized by the inlet velocity $\frac{L}{U_0}(\frac{\partial U_x}{\partial x} + \frac{\partial U_y}{\partial y})$ is computed. Both the maximum and mean normalized divergence are displayed in Fig. 8 for the neural network and the Jacobi methods.

The evolution of the maximum of the velocity divergence shows that the highest divergence is found in the first iterations. This is due to the fact that the NN has never seen flow injection during its training. However, once the flow is established, the Neural Network is able to correctly calculate the pressure field, yet difficulties arise when the plume starts interacting with the top wall. However, if only the maximum of the divergence is analyzed, the network accuracy seems worse than the Jacobi 100, which does not correspond with the conclusion driven from the evolution of the

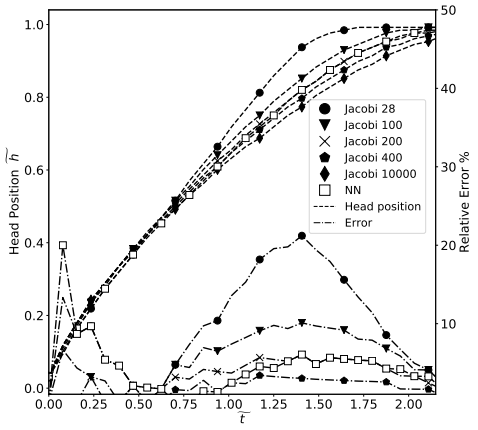
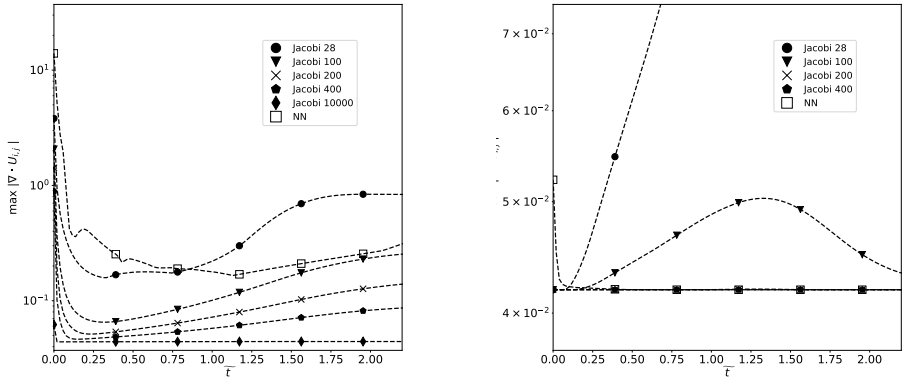


Fig. 7 Plume head position for Ri=0.15 using the NN and different Jacobi solvers.



(a) $max|\nabla \cdot U|$ **(b)** $mean|\nabla \cdot U|$

Fig. 8 Temporal evolution of maximum (8a) and mean (8b) normalized divergence for the plume test case at Ri=0.15.

plume's head. When looking to the mean velocity divergence (Fig. 8a), it can be appreciated that the network does perform in a similar way as a Jacobi 200, and much better than Jacobi 28 and 100. This suggests that Jacobi smoothes the error through the field, whereas the NN outputs an accurate mean field with errors concentrated locally, in particular close to boundaries. Penalizing these large local errors during the NN training might be an interesting improvement for future work. Similarly, including inflow and outflow boundary conditions in the training dataset might reduce the initial error on the divergence field.

VI. Hybrid computational strategy

A. Methodology

Previous results have shown the difficulty of solving the Poisson equation with a neural network when the Richardson number increases. In such cases, buoyancy effects increase and the velocity-pressure correlation deviates from the Ri = 0 dataset on which the NN was trained (Fig. 6). To tackle this issue, a hybrid approach is proposed which combines the Neural Network with the Jacobi algorithm, which is called only when divergence predicted with the NN is higher

than a given threshold. This keeps the benefit of fast computation with NN while guaranteeing accuracy. A sketch of this methodology is shown in Fig. 9.

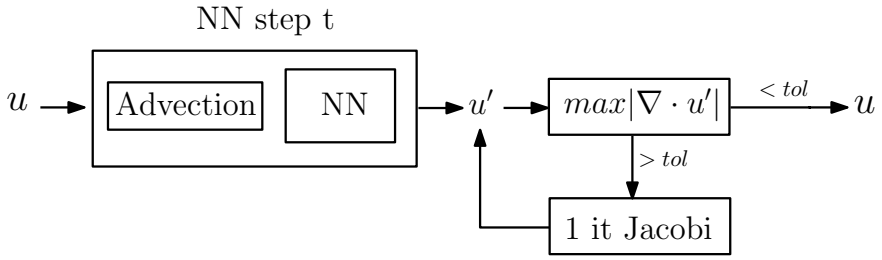


Fig. 9 Hybrid Fluid Solver scheme.

B. Application to plume simulations

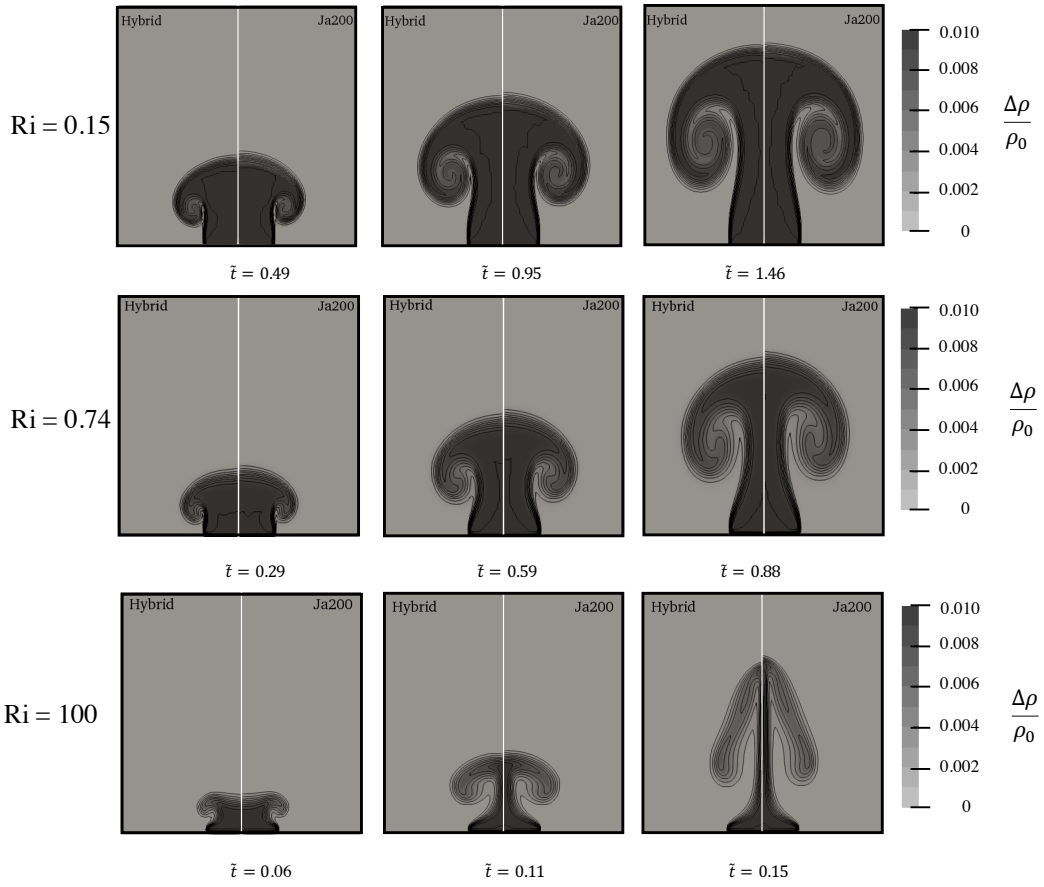


Fig. 10 Plume evolution comparison between the Hybrid strategy and Jacobi 200 for the 3 studied Richardson numbers.

Applying this hybrid strategy to the previous configurations, it can be appreciated that unphysical disturbances are no longer present for the plume with high Richardson number (Fig. 10). This is probably because the first iterations are now better resolved, avoiding the propagation of the error made in these first iterations. Even for the high Richardson numbers (Ri = 100) the network's predictions of the plume's head propagation speed and its shape are in good agreement

with Jacobi 200, and the unphysical perturbations on the surface are no longer present. As the Richardson number increases, the flow behavior gets further away from the regime on which it was trained (Ri = 0). This can already be noted by closely analyzing Fig. 6, as already at the first snapshot ($\widetilde{t} = 0.06$) a notable unphysical perturbation is observed. At this point, the flow is already non-symmetric, which will cause the jet to tilt towards the right at later times, as well as causing the apparition of a *double tip* on the plume's extremities. This difficulty can also be inferred by analyzing the number of Jacobi iterations needed by the Hybrid method, found in Fig. 11. For each Richardson number, the divergence threshold was set to ensure the absence of perturbations in the plume's head, as well as ensuring the symmetry of the plume. The number of Jacobi iterations is plotted against a dimension-less time, $t^* = \frac{t}{T_{0.9}}$, where $T_{0.9}$ is the time taken by the plume in each simulation to reach the 90% of the domain's height. Low Richardson number flows need around a hundred Jacobi iterations on the first two or three steps (where the influence of the inflow is significant). After this difficult start compensated by the Jacobi method, the CNN continues with only one iteration per timestep. However, for the Ri = 100 case, between 3 and 5 iterations are still needed while the plume grows to ensure the plume's symmetry. Note that these additional iterations are very few compared with the full Jacobi 200 method. The results obtained for the high Richardson test case highlight the benefit of the Hybrid method, as it extends the validity range of the neural network. Indeed, one of the main limitations of the neural networks is their limited capability to generalize as they strongly depend on the training dataset. On the particular case of this study, the Ri = 100 test case steps outside the scope of the network's training dataset, which in a traditional approach would force to either dismiss high Richardson flows, or to retrain the network with a more representative dataset. The hybrid approach offers a robust and reliable strategy to benefit from the acceleration of the neural network, without being forced to retrain every time a test case differs too much from the dataset scope.

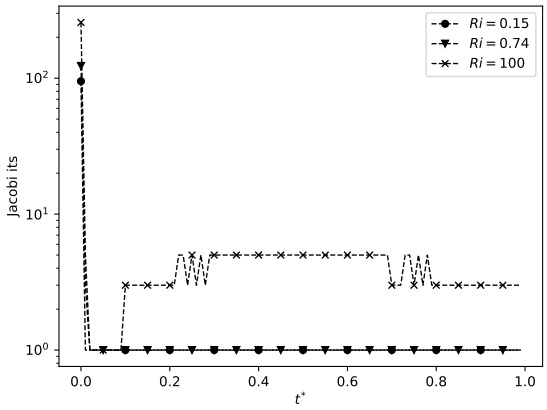


Fig. 11 Number of Jacobi iterations needed by the Hybrid method for the three studied Richardson numbers with a divergence threshold that ensures correct shape and speed of the plume's head.

C. Plume simulations at various Richardson numbers with an obstacle

In order to further quantify the gain of this innovative hybrid strategy, a more complex test case derived from the plume configuration is considered. A cylinder is added in the middle of the domain, to increase the fluid/wall interactions. Indeed, the analysis of the neural network prediction on the plume case revealed the difficulty of the network to predict accurate values close to boundaries. Consequently, this additional test case corresponds to a challenging case for the neural network, thus making a fair evaluation of the benefits of the proposed hybrid strategy. The studied configuration is found in Figure 12.

The flow values correspond to the Richardson 0.15 case, with an inlet velocity of 0.05 m s^{-1} and a $\Delta\rho$ of 0.01. Note that in this case the fluid is still assumed inviscid. For the demonstration, the threshold for the user-defined maximum divergence error is set to 0.3, which corresponds to the value of the long-time error observed in the plume case (Fig. 8a). It is expected from the analysis of section V that the presence of the cylinder in this new case will challenge the neural network, providing higher errors, due to local errors close to the cylinder walls. Therefore, the threshold of 0.3 should enforce the hybrid strategy to compensate for the high local error of the network. The main objective is then to assess (i)

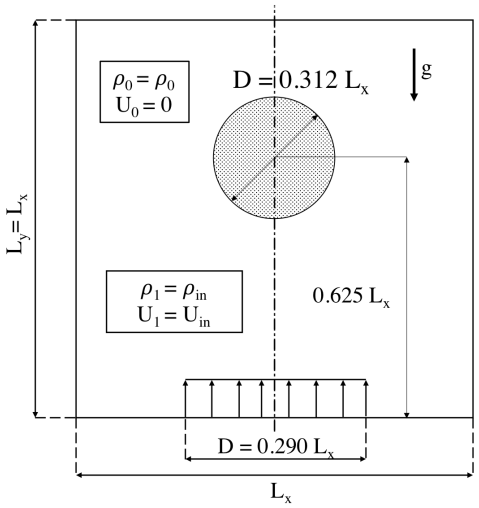


Fig. 12 Plume with cylinder configuration.

if the hybrid strategy is able to actually maintain an error below the aforementioned threshold, and (ii) to assess the inference time of the hybrid approach compared to the pure CNN predictions.

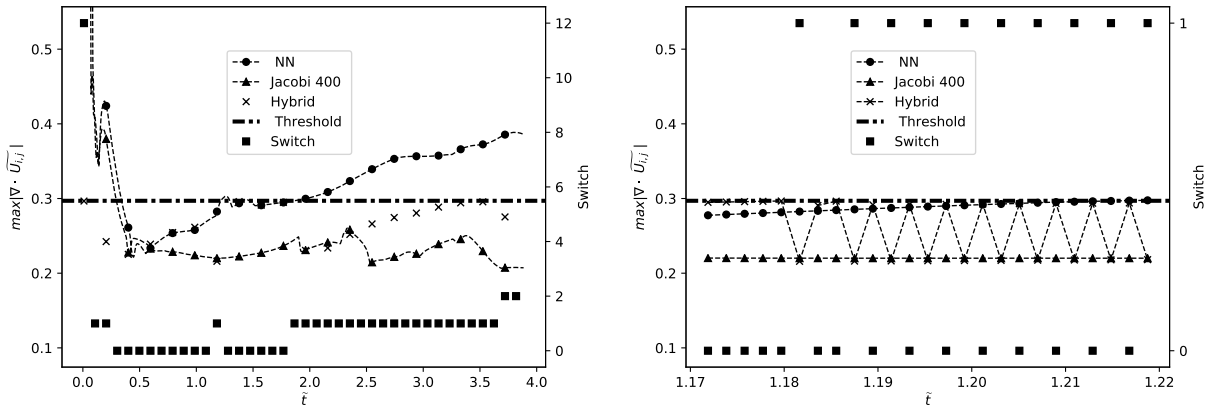


Fig. 13 Evolution of the maximum of the divergence for the Ri=0.15 plume with cylinder configuration for the NN, Jacobi 400 and the Hybrid solver. The right figure zooms around $\widetilde{t}$ = 1.2 when the threshold is reached.

Figure 13a displays the maximum normalized velocity divergence with time for the Jacobi 400, the pure CNN prediction, and the hybrid strategy, compared with the user-defined threshold 0.3. To monitor the hybridization, the switch indicator is also displayed. 0 indicates that the CNN prediction was accurate enough so that no Jacobi iterations were necessary, while a positive integer corresponds to the number of iterations needed to reach the threshold of 0.3. As already observed in the previous Section, Figure 13 reveals that the neural network has difficulties during the first iterations of the simulations, mainly because of the inlet, which generates large divergence errors. Although it is not visible on the graph, the first two timesteps require up to 4000 Jacobi iterations to reach the target divergence error. Once this transition phase has passed, the CNN does not need any Jacobi iteration until $\widetilde{t}$ = 1.17, (see Fig. 14) where the second vortex starts to develop on top of the plume's head. At this point, the network needs only one Jacobi iteration every two timesteps, as seen in Fig. 13 b.

Figure 14 illustrates that the hybrid approach is able to accurately predict the flow behavior, even after the fluid-cylinder interactions start. The comparison between Jacobi and the hybrid approach shows a good agreement

on the plume shape, with no unphysical perturbations propagated through time. However, the propagation speeds are slightly different, since the neural network propagates the plume faster, as in the plume case discussed in Section V (Fig. 7). Moreover, this example highlights the gain of the hybrid approach. First, compared with the pure CNN prediction which is diverging in time with a normalized error exceeding 0.4 at $\widetilde{t} = 4$, the hybrid approach is able to control the error below the user-defined threshold of 0.3. Second, the switch parameter indicates that, except for the two first initial time steps, the hybrid strategy needs only a few Jacobi iterations to reach this threshold. It suggests that error observed in the pure CNN prediction is due to the accumulation of error in time, and not due to the impossibility of the network to predict a particular flow mechanism occurring at a longer time.

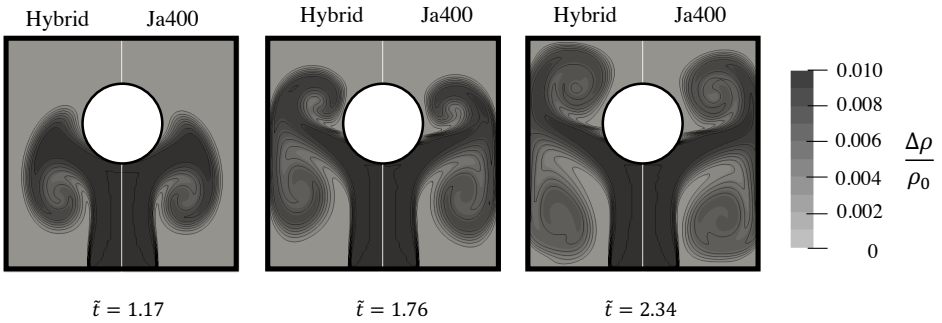


Fig. 14 Comparison between the Hybrid strategy and the Jacobi 400 solver for the plume test case with cylinder (Ri=0.15)

In terms of computational performance, the hybrid method reaches the user-defined threshold of 0.3, with only a 5% increase in the calculation time compared to the pure CNN calculation. For the same accuracy to be achieved with only Jacobi iterations, the calculation is 3.2 times slower than the hybrid method, while a calculation with Jacobi 400 is up to 12 times slower. This example demonstrates the capability of the proposed hybrid strategy to maintain a desired accuracy of the CNN-based solver while providing a significant speed-up, here evaluated at 320% at the same accuracy level.

VII. Conclusion

This work shows that the Poisson equation can be solved (up to a certain accuracy level) using Convolutional Neural Networks. A plume is simulated with an incompressible fluid solver coupled with a deep neural network, obtaining promising results. However, a satisfying accuracy level is not always achieved by the network: in particular, the accuracy is found case-dependent. This is exemplified on a plume case controlled by only one parameter: the Richardson number. For low Richardson numbers, a good accuracy of the neural network is observed, whereas for higher values larger errors are reported, even making the code numerically unstable. To circumvent this problem, a hybrid strategy between a Neural Network and a traditional Jacobi solver is proposed, which enables to accelerate fluid simulations while ensuring a user-defined accuracy level. The hybrid method enables as well to evaluate test cases that fall outside the scope of the training dataset, without been forced to retrain the network. This novel strategy is then evaluated on a more challenging test case where the plume impinges a cylindrical wall. Even in this case, the hybrid method provides a significant speed-up of 320% while maintaining the desired accuracy level.

References

[1] Rosenblatt, F., "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, Vol. 65, No. 6, 1958, p. 386.

[2] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, Vol. 86, No. 11, 1998, pp. 2278–2324.

[3] Krizhevsky, A., Sutskever, I., and Hinton, G. E., "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Curran Associates, Inc., 2012, pp. 1097–1105. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[4] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L., "Imagenet: A large-scale hierarchical image database," *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[5] Deng, X., Mao, M., Tu, G., Zhang, H., and Zhang, Y., "High-order and high accurate CFD methods and their applications for complex grid problems," *Communications in Computational Physics*, Vol. 11, No. 4, 2012, pp. 1081–1102.

[6] Brunton, S. L., Noack, B. R., and Koumoutsakos, P., "Machine learning for fluid mechanics," *Annual Review of Fluid Mechanics*, Vol. 52, 2019.

[7] Fukami, K., Fukagata, K., and Taira, K., "Super-resolution reconstruction of turbulent flows with machine learning," *Journal of Fluid Mechanics*, Vol. 870, 2019, pp. 106–120.

[8] Xie, Y., Franz, E., Chu, M., and Thuerey, N., "tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow," *ACM Transactions on Graphics (TOG)*, Vol. 37, No. 4, 2018, p. 95.

[9] Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., and Solenthaler, B., "Deep fluids: A generative network for parameterized fluid simulations," *Computer Graphics Forum*, Vol. 38, Wiley Online Library, 2019, pp. 59–70.

[10] Lee, Y., Yang, H., and Yin, Z., "PIV-DCNN: cascaded deep convolutional neural networks for particle image velocimetry," *Experiments in Fluids*, Vol. 58, No. 12, 2017, p. 171.

[11] Dissanayake, M., and Phan-Thien, N., "Neural-network-based approximations for solving partial differential equations," *communications in Numerical Methods in Engineering*, Vol. 10, No. 3, 1994, pp. 195–201.

[12] Gonzalez-Garcia, R., Rico-Martinez, R., and Kevrekidis, I., "Identification of distributed parameter systems: A neural net based approach," *Computers & chemical engineering*, Vol. 22, 1998, pp. S965–S968.

[13] Lagaris, I. E., Likas, A., and Fotiadis, D. I., "Artificial neural networks for solving ordinary and partial differential equations," *IEEE transactions on neural networks*, Vol. 9, No. 5, 1998, pp. 987–1000.

[14] Kumar, M., and Yadav, N., "Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey," *Computers & Mathematics with Applications*, Vol. 62, No. 10, 2011, pp. 3796–3811.

[15] Moody, J., and Darken, C. J., "Fast learning in networks of locally-tuned processing units," *Neural computation*, Vol. 1, No. 2, 1989, pp. 281–294.

[16] Choi, B., and Lee, J.-H., "Comparison of generalization ability on solving differential equations using backpropagation and reformulated radial basis function networks," *Neurocomputing*, Vol. 73, No. 1-3, 2009, pp. 115–118.

[17] Raissi, M., Perdikaris, P., and Karniadakis, G. E., "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10561*, 2017.

[18] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M., "Automatic differentiation in machine learning: a survey," *Journal of machine learning research*, Vol. 18, No. 153, 2018.

[19] Raissi, M., Perdikaris, P., and Karniadakis, G. E., "Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations," *arXiv preprint arXiv:1711.10566*, 2017.

[20] Raissi, M., and Karniadakis, G. E., "Hidden physics models: Machine learning of nonlinear partial differential equations," *Journal of Computational Physics*, Vol. 357, 2018, pp. 125–141.

[21] Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K., "Neural ordinary differential equations," *Advances in neural information processing systems*, 2018, pp. 6571–6583.

[22] Long, Z., Lu, Y., Ma, X., and Dong, B., "PDE-net: Learning PDEs from data," *arXiv preprint arXiv:1710.09668*, 2017.

[23] Eisenstat, S. C., "Efficient implementation of a class of preconditioned conjugate gradient methods," *SIAM Journal on Scientific and Statistical Computing*, Vol. 2, No. 1, 1981, pp. 1–4.

[24] Saad, Y., *Iterative methods for sparse linear systems*, Vol. 82, siam, 2003.

[25] Chentanez, N., and Müller, M., "Real-time Eulerian water simulation using a restricted tall cell grid," *ACM Transactions on Graphics (TOG)*, Vol. 30, ACM, 2011, p. 82.

[26] Yang, C., Yang, X., and Xiao, X., "Data-driven projection method in fluid simulation," *Computer Animation and Virtual Worlds*, Vol. 27, No. 3-4, 2016, pp. 415–424. https://doi.org/10.1002/cav.1695, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1695.

[27] Xiao, X., Zhou, Y., Wang, H., and Yang, X., "A Novel CNN-based Poisson Solver for Fluid Simulation," *IEEE Transactions on Visualization and Computer Graphics*, 2018, pp. 1–1. https://doi.org/10.1109/TVCG.2018.2873375.

[28] Özbay, A. G., Laizet, S., Tzirakis, P., Rizos, G., and Schuller, B., "Poisson CNN: Convolutional Neural Networks for the Solution of the Poisson Equation with Varying Meshes and Dirichlet Boundary Conditions," *arXiv:1910.08613 [physics, stat]*, 2019. URL http://arxiv.org/abs/1910.08613, arXiv: 1910.08613 version: 1.

[29] Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K., "Accelerating Eulerian Fluid Simulation with Convolutional Networks," *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, JMLR.org, 2017, pp. 3424–3433. URL http://dl.acm.org/citation.cfm?id=3305890.3306035.

[30] Thuerey, N., and Pfaff, T., "MantaFlow," , 2016.

[31] Bridson, R., *Fluid Simulation*, A. K. Peters, Ltd., Natick, MA, USA, 2008.

[32] Harlow, F. H., and Welch, J. E., "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface," *The physics of fluids*, Vol. 8, No. 12, 1965, pp. 2182–2189.

[33] Selle, A., Fedkiw, R., Kim, B., Liu, Y., and Rossignac, J., "An unconditionally stable MacCormack method," *Journal of Scientific Computing*, Vol. 35, No. 2-3, 2008, pp. 350–371.

[34] Quarteroni, A., Sacco, R., and Saleri, F., *Numerical mathematics*, Vol. 37, Springer Science & Business Media, 2010.

[35] Jacobi, C. G., "Ueber eine neue Auflösungsart der bei der Methode der kleinsten Quadrate vorkommenden lineären Gleichungen," *Astronomische Nachrichten*, Vol. 22, No. 20, 1845, pp. 297–306.

[36] Hornik, K., Stinchcombe, M., White, H., et al., "Multilayer feedforward networks are universal approximators." *Neural networks*, Vol. 2, No. 5, 1989, pp. 359–366.

[37] Goodfellow, I., Bengio, Y., and Courville, A., *Deep learning*, MIT press, 2016.

[38] Mathieu, M., Couprie, C., and LeCun, Y., "Deep multi-scale video prediction beyond mean square error," *CoRR*, Vol. abs/1511.05440, 2015. URL http://arxiv.org/abs/1511.05440.

[39] Elson, J., Douceur, J. J., Howell, J., and Saul, J., "Asirra: a CAPTCHA that exploits interest-aligned manual image categorization," 2007.

[40] Kim, T., Thürey, N., James, D., and Gross, M., "Wavelet turbulence for fluid simulation," *ACM Transactions on Graphics (TOG)*, Vol. 27, ACM, 2008, p. 50.