


RESEARCH ARTICLE 

Performance and accuracy assessments of an incompressible fluid solver coupled with a deep convolutional neural network

Ekhi Ajuria Illarramendi^{1,2,*} , Michaël Bauerheim¹ and Bénédicte Cuenot²

¹ISAE-SUPAERO, Université de Toulouse, Toulouse, France

²Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, CFD department, 31057 Toulouse, France

*Corresponding author. E-mail: ekhi.ajuria@cerfacs.fr

Received: 16 September 2021; **Revised:** 10 December 2021; **Accepted:** 10 January 2022


Keywords: Deep learning; hybrid strategy; partial differential equations; plume simulations; Poisson equation

Abstract

The resolution of the Poisson equation is usually one of the most computationally intensive steps for incompressible fluid solvers. Lately, DeepLearning, and especially convolutional neural networks (CNNs), has been introduced to solve this equation, leading to significant inference time reduction at the cost of a lack of guarantee on the accuracy of the solution. This drawback might lead to inaccuracies, potentially unstable simulations and prevent performing fair assessments of the CNN speedup for different network architectures. To circumvent this issue, a hybrid strategy is developed, which couples a CNN with a traditional iterative solver to ensure a user-defined accuracy level. The CNN hybrid method is tested on two flow cases: (a) the flow around a 2D cylinder and (b) the variable-density plumes with and without obstacles (both 2D and 3D), demonstrating remarkable generalization capabilities, ensuring both the accuracy and stability of the simulations. The error distribution of the predictions using several network architectures is further investigated in the plume test case. The introduced hybrid strategy allows a systematic evaluation of the CNN performance at the same accuracy level for various network architectures. In particular, the importance of incorporating multiple scales in the network architecture is demonstrated, since improving both the accuracy and the inference performance compared with feedforward CNN architectures. Thus, in addition to the pure networks' performance evaluation, this study has also led to numerous guidelines and results on how to build neural networks and computational strategies to predict unsteady flows with both accuracy and stability requirements.

Impact Statement

Recently, conventional neural networks (CNNs) have been used as efficient surrogate models of partial differential equations, substituting computationally expensive traditional iterative solvers. However, the networks do not guarantee any accuracy level, strongly limiting the spread of artificial intelligence (AI)-accelerated solvers in engineering applications. To tackle this issue, we introduce a hybrid strategy that combines the network predictions with traditional solvers, while still providing a significant speedup on the computational time. This hybrid strategy (a) ensures a user-defined accuracy level and (b) enables to generalize the network predictions to previously unseen conditions. Moreover, different CNN architectures are tested, where architectures with multiple scales consistently outperform simpler straightforward architectures, both in accuracy and inference time.

 This research article was awarded an Open Data badge for transparent practices. See the Data Availability Statement for details.

*The online version of this article has been updated since original publication. A notice detailing the change has also been published.

© The Author(s), 2022. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

1. Introduction

A wide variety of problems encountered in physics, engineering, and medicine, among others, can be described with partial differential equations (PDEs). These equations correspond to the mathematical translation of physical laws established from observable data. Among many examples, fluid flows can be described with the Navier–Stokes equations, electric and magnetic fields are modeled with the unified Maxwell’s equations, and the variation of concentration of chemicals can be modeled using the reaction–diffusion systems. Except for simplified configurations, PDEs cannot be solved analytically, and require numerical tools to approximate their solutions, which may result in computationally expensive calculations. Focusing on fluid mechanics, the Navier–Stokes equations are a nonlinear coupled PDE system which represents flows with a wide range of spatial and temporal scales, governed by forces depending on dimensionless numbers such as the Reynolds, Froude, Mach, and Richardson numbers, among others. The main difficulty arises from the nonlinear term of the momentum equation, which involves spatial gradients. Whereas this term generates multiple complex behaviors of the flow field (vorticity generation, transition to turbulence, etc.), it is by definition highly sensitive to the numerical setup: a small error on the gradient estimation may lead to very large differences in the solution. As a result, the computational fluid dynamics (CFD) community has produced an extreme effort over the last decades to improve numerical schemes, such as Deng et al. (2012), toward high-order discretization with high accuracy, most often at a price of a very high CPU cost. Such simulations have been made possible by massively parallel computing (high-performance computing [HPC]) combined with a constant amelioration of the computational hardware. More recently, the development and improvement of graphical processing units (GPUs) has also enabled to accelerate calculation codes, and has opened the path to machine learning (ML) techniques that were not yet developed in the CFD community.

The theoretical foundations of ML models were developed in the mid-twentieth century. In 1958, Rosenblatt (1958) developed an Artificial Neural Network (*Perceptron*), whereas the *backpropagation* was introduced in 1960 by Kelley (1960) for control applications (although the first successfully application was obtained by Rumelhart et al. (1986)). Convolutional neural networks (CNNs) appeared slightly later, when LeCun et al. (1998) created a network that extracted spatial features from 2D images. The performance of CNNs on image classification problems, as shown by Krizhevsky et al. (2012), and the development of more efficient GPUs then skyrocketed the use of ML in various domains. Mostly driven by the computer vision community, open-source frameworks were developed, which led to an even further spread of neural networks (NNs). This recent *data-driven* wave finally touched the CFD community, which made substantial efforts in the past 5 years to apply ML to flow problems.

ML techniques are optimization algorithms that extract patterns and structures from data, “generalizing” the valuable information found on its training data. When applied to Fluid dynamics, this usually translates into the creation of reduced-order models (ROMs), a simple representation of the information contained in the dataset. These simpler representations can be used in two different ways in CFD, as reviewed by Brunton et al. (2019): either discovering unknown models, or improving existing models. Discovering unknown models from data has been a long-standing problem in the scientific community. Throughout the history of humanity, people built models to describe the behavior of observed phenomena, in order to make future predictions. Nowadays, with the advance of technology, a time has reached where a lot of data of different processes are available, yet being still underexploited. This has promoted ML, with the aim to benefit from these data, by extracting features to develop reduced models. For instance, Brunton et al. (2016), followed by Rudy et al. (2017), established governing equations for the chaotic Lorenz system or for the vortex shedding behind a cylinder using a sparse dynamics model and a sparse regression model. Another example is the use of ML for problems where the governing equations are well known, but too expensive to resolve, as in turbulent flows, for example, Zhang and Duraisamy (2015) used NNs to predict subgrid-scale correction factors on turbulent flows. Fukami et al. (2019) applied CNNs to reconstruct turbulent flows that respect the energy spectrum, whereas other works, such as Xie et al. (2018), Kim et al. (2019), and Subramaniam et al. (2020), used generative adversarial

networks (GANs) to reconstruct turbulent fields. Similarly, CNNs have also been used in turbulent combustion to predict the subgrid flame wrinkling, as shown by Lapeyre et al. (2019).

When the resolution of PDEs is computationally expensive, ML techniques can be used to develop surrogate models that capture the essence of the studied phenomena with a reduced computational cost. Classical ROMs, as found in the works of Cizmas et al. (2008), Rowley and Dawson (2017), and Taira et al. (2017), include proper orthogonal decomposition or dynamic mode decomposition (DMD). These methods can be combined with ML techniques, such as autoencoders, as done by Otto and Rowley (2019) or long short-term memory (LSTM) NNs, by Pawar et al. (2019), in order to get satisfactory results on problems like the 2D Navier–Stokes equations. The first studies introducing ML to solve PDEs date back to the 1990s, with the work of Lee and Kang (1990) followed by Dissanayake and Phan-Thien (1994). Dissanayake’s team proposed a multilayer feedforward network to solve a linear Poisson equation and a thermal conduction problem on a nonlinear heat generation test case. Hornik et al. (1989) showed that NNs can be described as “universal approximator” functions, which enable to easily differentiate their inputs with respect to their output. That way, they constituted a novel approach where the classical finite-element or finite-difference discretization problem was substituted by an unconstrained optimization problem. Following the reduced-order-modeling philosophy, their objective was to obtain an easy-to-implement solver that could rapidly give accurate results. Lagaris et al. (1998) further exploited the idea, adding both Neumann and Dirichlet boundary conditions (BCs). Gonzalez-Garcia et al. (1998) developed a different approach, where they modeled the right-hand side of a PDE with an NN, in order to turn the problem into a more simple Ordinary Differential Equation. They showed that the network enables to generalize from experimental data which might be incomplete, or noisy, testing on the Kuramoto–Sivashinsky equation. All these works profit from the differentiable nature of NNs. However, these networks were rapidly limited by the available computational capability and were restricted to a single particular PDE, with not much margin for generalization.

With the increase of computational resources, larger and more complex networks started to be introduced, such as multilayer perceptrons (MLPs) used by Shirvany et al. (2009) or radial basis functions, by Mai-Duy (2005). Recently, Raissi et al. (2017a) introduced *physics-informed neural networks* (PINNs), where a priori physical knowledge is introduced into the training process of NNs. By using the residual of the physical equation to optimize the network’s trainable parameters, the network ensures physical laws and may even lead on to training processes which do not need a database, as shown by Sun et al. (2020). Such approach was tested on different PDEs, including the incompressible Navier–Stokes equations in the works of Raissi et al. (2017b) and Raissi and Karniadakis (2018), which have also been tackled by several works which employ networks to predict the future state of fluid flows, that is, as end-to-end solvers. Some recent works by Wiewel et al. (2019) include recurrent network architectures, where an LSTM (Hochreiter and Schmidhuber, 1997) network was used to encode fluid structures into a latent space, or more classical CNN by Wandel et al. (2020), where an unsupervised network training is evaluated to test the performance gain of CNNs. However, the majority of previously described works, that focused both on the Navier–Stokes or more general PDEs, handle the resolution of PDEs independently to traditional fluid solvers. Even if physical laws are embedded during the training procedure, the interaction between the NNs and the traditional fluid solvers still offers a wide variety of options for further studies. One of the study fields on which a large effort is devoted to further study the coupling networks with fluid solvers corresponds to the resolution of the Poisson equation in incompressible solvers. The problem was first tackled by Yang et al. (2016), who used an MLP, and Xiao et al. (2018) continued with the idea of using CNN in larger domains. Moreover, continuing the use of CNN for the resolution of the Poisson equation, the work of Özbay et al. (2021) focused on the resolution of both the homogeneous and the BC problems for the Taylor–Green Vortex test case. Along the same line, the work of Tompson et al. (2017) should be highlighted, where a CNN is coupled to an incompressible fluid solver. They introduced a physical loss similar to the one of PINN, based on the continuity equation. Similarly to the work of Tompson et al. (2017), Dong et al. (2019) used NNs to perform the pressure projection on Eulerian fluid simulations. Moreover, they trained an MLP to choose between multiple networks that better match the user’s defined constraints. Finally, recent work by Um et al. (2021) coupled a

differentiable fluid solver with a CNN in order to further encode flow dynamics during the network training.

Some works have also tackled different strategies to accelerate fluid simulations using NNs. For example, the work of Kochkov et al. (2021) accelerates direct numerical simulation (DNS) solvers by improving the convection step of a DNS solver. They use a learned interpolation, where an NN is used to predict the interpolation coefficients which vary depending on the flow topology, enabling to capture information lost on coarser domains. Thus, the same simulation is performed at higher precision.

This work further extends a previous work by Ajuria Illarramendi et al. (2020), which introduced a hybrid CNN-Jacobi method to solve incompressible flows, that guarantees a user-defined divergence level. The numerical setup and test cases are described in Section 2.2, and the CNN approach to solve the Poisson equation is described in Section 3. Different NNs are compared for the flow around a cylinder and plume test case. Particularly focusing on buoyant plumes, the network behavior and error distribution are compared and analyzed in Section 5. The methodology is then extended to 3D buoyant plumes in Section 6. Finally, Section 7 presents a performance assessment of the hybrid CNN approach with various network architectures. Results show that for the same error level, networks with multiple scales allow to perform accurate predictions faster than classical solvers. Thus, while a significant speedup is found, the present study has also produced multiple results on how to build efficient, accurate, and reliable AI-based solvers, guidelines that could be transferable to other problems tackled by AI-based surrogate models.

2. Configuration and Methodology

2.1. Flow around a 2D cylinder

As a first approach to evaluate the interaction between CNN and incompressible fluid solvers, the viscous flow around a 2D cylinder will be studied. The problem is governed by only one dimensionless number, the Reynolds number, that will be used to investigate the generalization capabilities of the AI-based solver (training will be performed on Euler simulations, i.e., $Re \rightarrow \infty$). When the Reynolds number is higher than a critical value ($Re_c \sim 47$), an instability develops in the cylinder wake resulting in a periodic vortex shedding. This canonical problem is related to a wide variety of engineering challenges, as it relates to any vibration problems caused on the wake of any bluff body (Zdravkovich (1981)). Although experimental studies on the subject date back to the beginning of the twentieth century, this test case is still widely used as benchmark in the CFD and Deep Learning community (Lee and You, 2019; Murata et al., 2020). The configuration studied in this work is found in Figure 1. To avoid problems related to the interaction between the BCs and the cylinder wake, the walls are located 20 diameters away from the cylinder upstream, as well as from the top and bottom walls. Due to the coupling of NNs with a DL-friendly framework based on Mantaflow (Thuerey and Pfaff, 2016) developed for GPUs, a uniform structured Cartesian mesh is

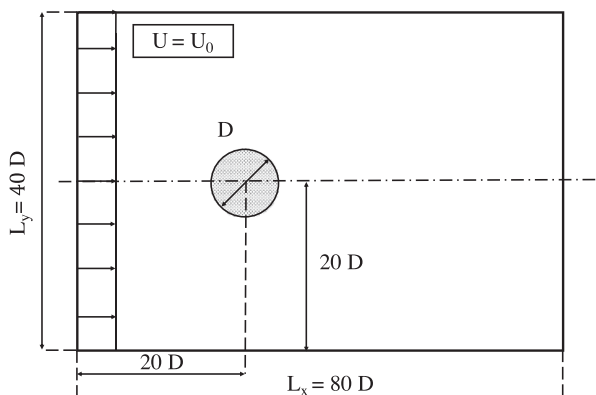


Figure 1. Von Karman test case configuration.

needed. This grid does not count with immersed boundary methods, and no mesh refinement around the cylinder is possible. Therefore, to ensure a good resolution around the cylinder, the size of the entire mesh increases dramatically. This same discretization is used for all of the following test cases. Note that while the present framework is limited by this Cartesian grid, the developed hybrid AI–CFD strategy and associated results could be transferable to more advanced network architectures, such as graph NNs.

Different Reynolds numbers will be analyzed, to study the network behavior in different flow regimes. Low Reynolds numbers will be kept to ensure the validity of the 2D simulations. As the training was done with no viscosity, this test case ensures network generalization to flows not seen during training.

2.2. Plume Case

Buoyancy forces are induced by density variations in the presence of gravity. The study of buoyancy-driven flows has been a long-standing problem for the fluid mechanics community, as shown by Turner (1979), as they correspond to a wide variety of remarkable geophysical flows, from volcanic eruptions to avalanches. The Earth’s climate and ocean’s behavior are strongly dependent of buoyancy, as the ocean’s currents and tides depend on the creation and distribution of water density variations. Buoyant coastal currents, for example, are responsible for redistributing the fresh water coming from rivers and other sources into the ocean, carrying as well sediments or pollutants which can strongly impact natural ecosystems. A thorough analysis of buoyancy-driven flows, especially focused on oceanic flows, can be found in the work of Chassignet et al. (2012). The civil engineering community also studies flows driven by buoyancy forces, which control the ventilation of buildings, as seen in the work of Allocca et al. (2003) or even entire cities, with direct effect on the propagation of pollutant particles, shown by Luo and Li (2011). The propagation of fires in buildings has also led to extended work with theoretical, numerical, and experimental models for smoke propagation, as shown by Ahn et al. (2019).

Usually, the term “buoyancy-driven flow” refers to the situation where a lighter moving fluid is surrounded by a heavier fluid, whereas a heavier moving fluid surrounded by a lighter fluid is called a density-driven flow. This work focuses on plumes, which result from injecting a light fluid into a heavier quiescent environment. Plumes can be found in a wide variety of physical phenomena, such as the flow escaping the chimney of a locomotive steam engine or even the smoke elevating from a cigarette. These types of flows have been studied for over 80 years, after the work of Zeldovich (1937), on similarity analysis, and Schmidt (1941), who obtained analytical expressions for the mean velocity and temperature profile in turbulent plumes. Despite the presence of a wide variety of experimental works by George Jr et al. (1977) and Shabbir and George (1994), and theoretical studies by Morton (1959) in this domain, the phenomenology underlying plume physics is still not completely mastered. Numerous works have used CFD to understand this type of flows, based either on the Reynolds-averaged Navier–Stokes equations, such as the works of Van Maele and Merci (2006) and Lou et al. (2019) or large eddy simulations (LES), by Zhou et al. (2001). Even for simplified incompressible and inviscid plumes, the interaction between the gravity source term and the momentum-driven forces results in an interesting and complex behavior. The misalignment of the density and pressure gradients causes the generation of vorticity and the development of instabilities, as introduced by Turner (1979). Thus, even with a simplified setup, a variety of phenomena with different spatial scales can be found. For further information, the reader can refer to the reviews of Turner (1969) and Hunt and Van den Bremer (2011).

The plume case has also been widely treated in the computer vision community, due to the difficulty of calculating realistic smokes in real time, which requires efficient computational solvers, and explains why ML was first introduced in this context. One can cite the works of Chu and Thurey (2017) with a CNN, Kim et al. (2019) using GAN with a novel stream function-based loss, and Tompson et al. (2017), who introduced the idea of a CNN coupled to an incompressible fluid solver.

The two configurations studied in this work are displayed in Figure 2. The flow is assumed inviscid (infinite Reynolds number) and adiabatic. Thus, the plume behavior is driven by only one dimensionless number, called the Richardson number (Equation (1)), which compares the buoyancy and inertia forces:

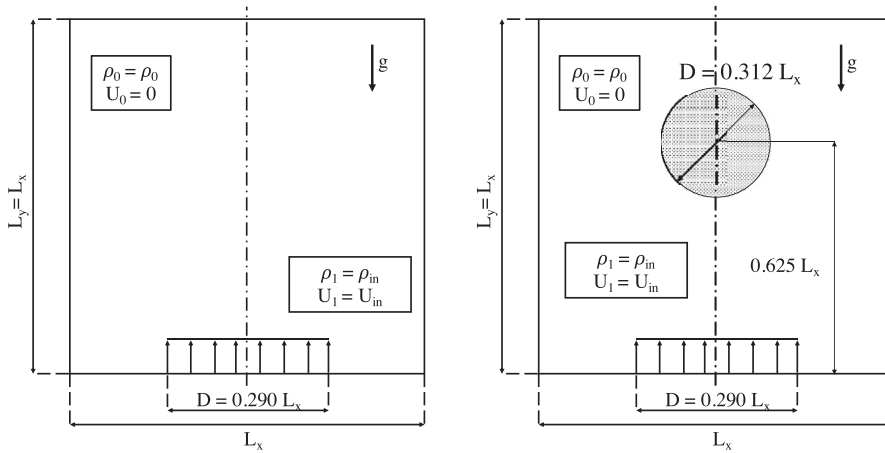


Figure 2. Plume configuration with and without cylinder.

$$R_i = \frac{\Delta\rho}{\rho_0} \frac{gL}{U^2}, \tag{1}$$

where $\Delta\rho/\rho_0$ is the density contrast between the light and heavy fluid, g is the gravity, L is a characteristic length, and U is the velocity of the injected lighter fluid. High Richardson numbers imply that the flow is piloted by the buoyant forces, whereas low Richardson numbers indicate that the flow is driven by momentum (also known as jets). In both cases, the domain is a square box of length L_X , with an inflow centered in the bottom face of the domain, of radius $D_{in} = 0.29L_X$. In the second configuration, a cylinder of diameter $D_c = 0.312L_X$ is added, aligned with the horizontal axis, and centered at $0.625L_X$ on the vertical axis (Figure 2). The objective is to evaluate the behavior of the proposed learning-based approach when the flow impinges objects and walls.

2.3. Numerical setup

For the CFD study, a fluid solver based on the open-source solver Mantaflow developed by Thuerey and Pfaff (2016) is used to solve the incompressible Navier–Stokes equations. A variable-density incompressible formulation is employed with external forces due to gravity. To model buoyancy-driven flows under the incompressibility assumption, the Boussinesq approximation as described by Gray and Giorgini (1976) is applied, which holds for small density variations compared to the mean density, that is, $\Delta\rho/\rho_0 \ll 1$. Note that the continuity equation with the Boussinesq approximation and the incompressibility constraint leads to the advection of the density variation $\Delta\rho$ as a passive scalar, that is, being constant along a streamline.

The incompressible Navier–Stokes equations are solved in two steps: (a) the advection step and (b) the pressure projection step. The computational domain is discretized on a uniform Cartesian grid with a finite-difference scheme on a marker-and-cell staggered grid, more extensively explained by Harlow and Welch (1965). First, the density is convected along streamlines, using an unconditionally stable, second-order accurate Maccormack semi-Lagrangian method, as described by Selle et al. (2008). Once the density field is advected, the momentum equation is solved through a splitting method, where the velocity field is first advected similarly to the density field, and then modified by the external forces with a forward Euler scheme, resulting in the velocity field u^* . The viscous effect is added as source term, the same way as gravity forces. The domain is discretized using a first-order finite-difference scheme. Note that the advection accuracy is a limitation of the fluid solver, and not from the developed deep learning (DL) approach proposed here. More sophisticated CFD frameworks capable of embedding DL solvers are

under development, but out of the scope of the present study. Since the pressure term in the momentum equation and the mass conservation equations have not yet been used, the velocity field u^* does not satisfy the incompressible kinematic constraint $\nabla \cdot u^* = 0$. To do so, this pressure term is used as a correction term to ensure a divergence-free velocity field, so that the corrected velocity reads:

$$\mathbf{u} \simeq \mathbf{u}^* - \Delta t \frac{1}{\rho_0} \nabla p. \quad (2)$$

Taking the divergence of both sides of Equation (2) and imposing the continuity equation $\nabla \cdot \mathbf{u} = 0$ yields the well-known Poisson equation:

$$\frac{\Delta t}{\rho_0} \nabla^2 p = \nabla \cdot \mathbf{u}^*. \quad (3)$$

This equation is solved to obtain the pressure correction, ensuring that the velocity field at the next time step satisfies both the momentum and the mass conservation equations. This correction is applied using Equation (2).

The Poisson equation is a second-order elliptical PDE, which is well known to be computationally expensive, taking up to 80% of the computational time of the incompressible solver, as shown by Ajuria Illarramendi et al. (2020). For this reason, this work only focuses on the resolution of this particular equation. This equation may be solved using direct methods, described in the works of Buzbee et al. (1970) and Quarteroni et al. (2010), such as the lower-upper (LU) factorization, which can be interpreted as a Gaussian elimination method (GEM). The idea behind these methods is to recast the A matrix of the linear system $Ax = b$ into an upper triangular and a lower triangular matrix $LUx = b$, which enables to decompose the problem into two successive GEMs, $Ly = b$ and $Ux = y$, respectively. These methods can provide solutions at the machine precision; however, they tend to be computationally expensive, especially for large computational domains since the cost of solving a single GEM is proportional to $2n^3/3$ operations, where n is the number of points in the mesh. Further acceleration toward fast Poisson solvers can be produced by dedicated methods, such as using Fast Fourier Transforms (McKenney et al., 1995), reducing the complexity to $n \log n$. However, these methods are usually restricted to specific applications (e.g., fully uniform Cartesian grids). Consequently, the CFD community traditionally uses iterative methods which can be easily parallelized in multiple CPU. Note that while the present AI-based hybrid method is validated on a fully uniform Cartesian grid and linear PDEs with constant coefficients because of the chosen fluid solver, the whole methodology can be extended to unstructured data and other linear or nonlinear PDEs. For such methods, a trade-off has to be chosen between accuracy and computational cost. Despite the need of multiple iterations to achieve convergence, those methods are usually more efficient than direct approaches since their computational cost is proportional to n^2 per iteration. Among the iterative methods, the Jacobi method, which was first developed by the German mathematician Carl Gustav (Jacobi, 1845), decomposes the matrix A into a diagonal matrix D and the upper and lower diagonal matrices L and U. The linear system can be expressed as $DX = b - (L + U)X$, which is equivalent to $X = D^{-1}(b - (L + U)X)$. An iterative process is obtained by solving the previous equation in the form $X_{t+1} = D^{-1}(b - (L + U)X_t)$. As any iterative method, the number of iterations required toward convergence strongly depends on the initial guess X_0 : an additional result of the present work compared with previous studies is to use the NN prediction as initial guess to speed up the Jacobi method. More advanced methods, such as the ones described by Saad (2003) and Hosseinverdi and Fasel (2018), exist for the Poisson equation, yet this work will focus only on the Jacobi method for the sake of stability, its easy implementation on GPUs, as well as its straightforward hybridization with NNs. Even if the Jacobi solver is not the fastest available method, other works have compared similar architectures with state-of-the-art linear solvers optimized for CPU parallelization (Cheng et al., 2021), which show that NNs can be competitive compared to traditional linear solvers, such as the conjugate gradient (CG) method with the HYPRE BoomerAMG preconditioner. Moreover, authors show how with little modifications CNNs can be applied to cylindrical coordinates, which are slightly more challenging for fast Fourier solvers (Moon et al., 2019).

3. A Deep-Learning Approach to Solve the Poisson Equation

To overcome the difficulties of iterative methods, the present study focuses on solving the Poisson equation with a CNN. Particular attention is put on its ability to tackle new cases, not seen during the training phase. To further ensure both a fast and reliable prediction of the corrected velocity field, this CNN method will be hybridized with the Jacobi method, using the CNN prediction as initial guess to significantly speed up the Jacobi approach.

CNNs perform multiple linear transformations, typically $y = w_i x + b_i$, followed by nonlinear activation functions to output the desired field. The *training* of the NN consists in the optimization of the weights w_i and biases b_i to minimize a user-defined objective function \mathcal{L} . CNNs differ from traditional MLP as they do not apply the weights and biases to the entire input fields. Indeed, CNNs apply the transformation locally through a kernel of limited size, typically 3×3 , which is then “slid” through the entire input field. This type of network is widely used for image recognition tasks, as the kernels can be interpreted as filters that extract patterns and structures from the input images. CNNs are therefore particularly adapted to fluid mechanics, since the underlying physics are mostly driven by flow structures (jets, vortices, stagnation points, etc.). In addition, compared with MLP, CNNs usually require a smaller number of tunable parameters (w_i and b_i), and are theoretically independent of the size of the input field: the same trained network can be reused on cases with different resolutions or meshes. However, a current limitation of CNNs is the need of perfectly uniform Cartesian grids, whereas most of today’s CFD simulations are performed on nonuniform, possibly unstructured, meshes. While interpolation from nonuniform meshes onto a uniform Cartesian grid is possible, recent works have efficiently adapted convolution operations to unstructured meshes, as shown by Baque et al. (2018) and Fukami et al. (2021). Further work by Pfaff et al. (2020) has extended the use of graph NNs for unstructured meshes. However, these methods are still complex and not yet mature, so classical CNNs are still widely used for deep learning on a CFD context. For the sake of simplicity, this study considers a perfectly uniform Cartesian grid. The spatial resolution will, however, be varied to validate the convergence properties of the method, as well as to evaluate its computational performance on different grid sizes. Thus, this study focuses on the development of a hybrid approach mixing CFD and CNNs, where results are expected to be transferable to the more advanced DL techniques for unstructured meshes, such as graph CNNs.

3.1. Network architectures

The choice of the NN architecture is usually driven by the user’s experience, and further improved by a trial-and-error approach. Since the network architecture is critical, tools for automatic architecture search have been also developed by Elsken et al. (2018). While discovering efficient network architectures, these automated data-driven search strategies require an extremely high computational time since multiple trainings have to be performed, as shown by Zoph et al. (2018), who used in parallel 500 Nvidia p100 GPUs during 4 days to find an optimal network for scalable image classification. Architectures are even more critical in physics, and especially fluid mechanics, since the studied problems usually involve a large range of different scales which need to be captured accurately by the network. Toward this objective, Geng and Wang (2020) have employed such a method to discover optimal NNs dedicated to multiscale analysis of seismic waves. However, despite those automatic strategies, the choice of the network architecture and its effect on accuracy and inference time is often ignored. In particular, guidelines still need to be established for physics-related tasks. As a first attempt to generate such guidelines, Morimoto et al. (2021) analyzed the influence of several network choices when using CNN for fluid dynamic-related problems. In particular, they studied the influence of the choice of several hyperparameters for CNNs for both CNN-based metamodeling and CNN autoencoders, such as the influence of the entry-point location of scalar inputs or the type of padding used within the convolutions. The present work further investigates different types of network architectures, in order to analyze their effect on the resolution of the Poisson equation. Note that the objective of this work is not to find the optimum architecture for the resolution of the Poisson equation, but rather to provide general trends and understanding of the architecture effect on the network accuracy and performance. For all architectures tested here, the pressure field is computed using a CNN

for which the inputs are the uncorrected velocity divergence $\nabla \cdot u^*$ and a Boolean field describing the object geometry.

First, a simple “feed-forward” convolutional network is considered, denoted hereafter MonoScale. This network (Figure 3) contains 395,601 tunable parameters distributed in nine layers, following a straightforward architecture. For convolution operations, a replication padding is used to ensure that all the feature maps keep the size of the original fields. This padding type outperforms networks trained with the zero-padding strategy, as the padding type better matches the flow BCs. This finding matches the work of Alguacil et al. (2021), which highlights the importance of the padding choice for spatiotemporal evolving problems. The replication padding creates *ghost cells* on the image boundaries with a value copied from the closest image pixel. This network is the most simple strategy to tackle the Poisson problem, as the input is just passed through a series of convolutional layers, without any further modification or postprocessing.

The second architecture is the MultiScale (Figure 4), introduced by Mathieu et al. (2016) for video image prediction, and introduced to solve the Poisson equation by Ajuria Illarramendi et al. (2020). It has also been employed in other studies on fluid mechanics, such as in Fukami et al. (2020) focusing on the super-resolution of turbulent flows, or applied to the propagation of acoustic waves by Alguacil et al.

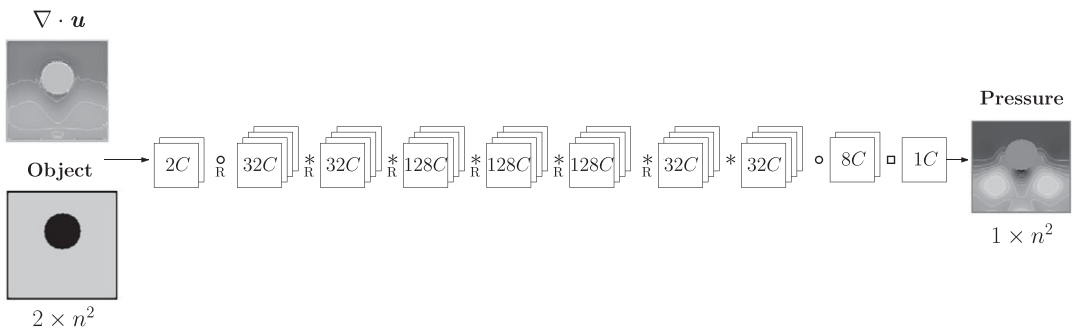


Figure 3. MonoScale network containing 395,601 parameters, where \circ corresponds to convolution with kernels of size 5×5 , $*$ to kernels of size 3×3 , and \square to kernels of size 1×1 . R corresponds to the ReLU activation function. Each box indicates the number of feature maps, or channels (C) present at each layer.

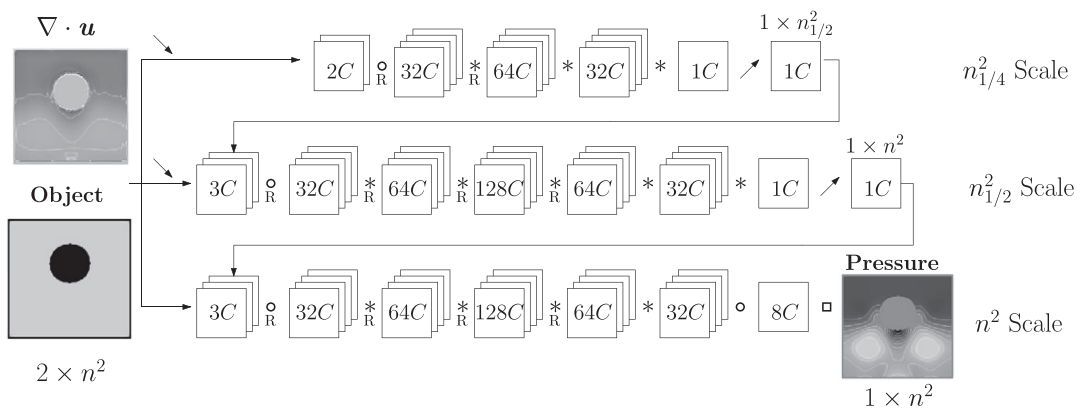


Figure 4. MultiScale network, with 418,640 parameters, where \circ corresponds to convolution with kernels of size 5×5 , $*$ to kernels of size 3×3 , and \square to kernels of size 1×1 . R corresponds to the ReLU activation function, \searrow indicates a bilinear downsampling operation, whereas \nearrow corresponds to the bilinear interpolation. Each box indicates the number of feature maps, or channels (C) present at each layer.

(2020). The idea behind this architecture is to feed the NN with several representations of the same inputs, focusing on different scales. To do so, the original input is interpolated on coarser meshes: while the largest resolution retains all scales, smaller resolutions focus only on the large scales. The network then encodes information linked to the different spatial scales at each level, which helps the network on the generalization task, and avoids spurious high-frequency oscillations on the large features of the image, as described by Mathieu et al. (2016). Here, the MultiScale architecture (Figure 4) has three scales, of sizes $n_{1/4}^2$, $n_{1/2}^2$, and n^2 , respectively. The first scale interpolates the original images on a quarter-cell size mesh. The resulting image is then interpolated on a half-cell size mesh to twice its size ($n_{1/2}^2$). The middle scale takes as an input both the interpolation of the original entries on a half-cell size mesh, and the output of the previous scale. Finally, the resulting image is interpolated to the original size n^2 , and is concatenated to the input fields. It is then fed to the final scale branch. This network architecture aims to progressively encode information from the coarsest scale to the finest; thus, the input features are first downsampled to the lowest resolution $n_{1/4}^2$ and the information is encoded to a feature map containing the information extracted from this coarsest scale. This process is repeated for each intermediate scale, downsampling the input fields to the corresponding scale and concatenating the information encoded from the previous coarser scale. The first scale contains five layers, whereas both the intermediate and final scales contain seven layers. Nonlinear Rectified Linear Unit (ReLU) activation functions are placed after each convolution layer, except on the last two ones, to enable the network to compute positive and negative outputs.

The MonoScale and MultiScale networks are then compared with the well-known Unet architecture (Figure 5), which was first developed by Ronneberger et al. (2015) for the segmentation of biomedical images. Apart from the initial biology-related applications, this network has also been used in many other fields, including for regression tasks in fluid mechanics. For instance, Lapeyre et al. (2019) employed this architecture on turbulent subgrid scale modeling for reacting flows. The structure of the Unet is similar to the MultiScale, as it also combines information extracted at different spatial scales. However, the Unet differs from MultiScale on the scale treatment: while several scales are managed in parallel in the MultiScale network and the input image is fed directly to each branch, the Unet acts as a simple

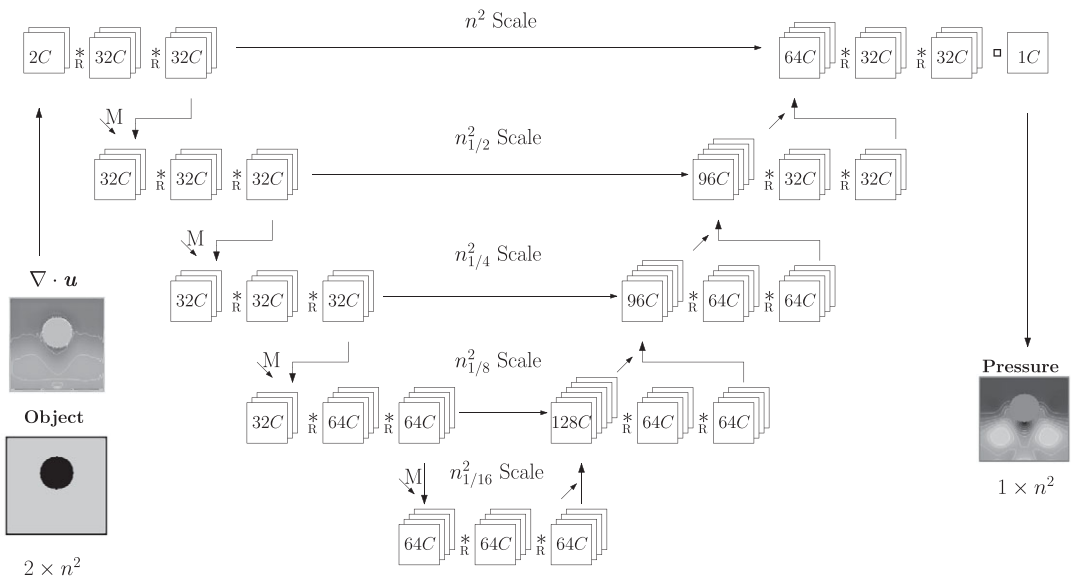


Figure 5. Unet with 443,521 parameters, where * corresponds to kernels of size 3×3 and \square to kernels of size 1×1 . R corresponds to the ReLU activation function, $\searrow M$ indicates a MaxP Pooling operation, whereas \nearrow corresponds to the bilinear interpolation. At each scale, the MaxPooling step reduces the image to half of its original size, whereas the interpolation upsamples the image to the double of its size. Each box indicates the number of feature maps, or channels (C) present at each layer.

feed-forward network with a decreasing–increasing resolution to encode information in a low-dimensional manifold (at the smaller scale). The image input is fed to the network only at the initial resolution. In addition, skip connections are imposed at each scale, allowing information to bypass the lower-resolution treatments and to avoid the vanishing gradient issue inherent to deep network architectures.

Note that those architectures are similar to multigrid and multifidelity approaches developed in CFD. Indeed, the MultiScale network corresponds to multifidelity approaches, as described by McCormick (1989), where the original problem is solved on a coarse grid, then the solution is used as an initial guess to a more refined grid, as done by Southwell (1956). The Unet, on the other hand, is similar to multigrid solvers, further detailed in the work of Wesseling (1995), where the initial problem is solved with a computationally nonexpensive solver, and the result is corrected with approximations computed on coarser grids.

In comparison with classical deep NNs employed for classification tasks which may contain 10^6 (e.g., ResNets, first developed by He et al. (2016)) to 10^8 (e.g., VGG-16, developed by Simonyan and Zisserman (2015)) tunable weights, the proposed architectures are relatively small, containing all about 4×10^5 parameters. Since the main goal of the present approach is to accelerate the classical iterative Poisson solvers, a limited number of parameters have been chosen. Note, however, that this cannot constitute a guideline since the performance of the networks will be evaluated at iso-level of accuracy: the number of parameters is, therefore, a nontrivial trade-off between accuracy and inference time, for which no rule exists in the literature. To investigate this accuracy-performance link, a smaller MultiScale containing 1.3×10^5 parameters is introduced in complement to the three previous networks. The structure is the same as the MultiScale, but only the number of filters per layer is changed, resulting in a network with around three times fewer parameters. This network will be denoted *SmallScale* in the following.

Consequently, four deep NNs will be analyzed on the two test cases introduced in Section 2.2. For each architecture, both their error levels and distributions, as well as their performances on inference time, will be assessed.

3.2. Training and loss function

The networks introduced in Section 3.1 are trained using a procedure similar to Tompson et al. (2017) and Ajuria Illarramendi et al. (2020). Compared with most studies on ML using a known output as target in the loss function \mathcal{L} for a supervised training, extensively described by LeCun et al. (1998), here a semisupervised learning strategy is used, where no “ground truth” field is needed. To do so, the residual of the continuity equation is used as a loss function, where the divergence of the velocity field is computed using a first-order forward finite-difference scheme. Thus, training the network is equivalent to minimizing this residual, that is, to enforce the mass conservation equation, which is the goal of solving the Poisson equation in an incompressible solver. Note that this approach can be considered as a physics-driven neural network (PDNN), demonstrating how a target physical equation can be introduced into the learning strategy. It differs from PINNs (developed by Raissi et al. (2017a)), which combines a semisupervised physics-driven approach with a standard supervised learning method in which a “ground truth” target is required. The main benefit of PDNN over PINN is the possibility of adding long-term loss effects during the training by constraining the network to produce consistent predictions in time.

To train the network, the widely used mean-squared-error (MSE) metric, as used by Hunt (1973), is introduced, which computes the L2 distance between two fields. As the loss function of the PDNN is the residual of the continuity equation, the MSE metric is applied directly to the divergence of the corrected velocity, which can be interpreted as the L2 distance to zero. Since the solution of the Poisson equation is the pressure correction term, the network output has to be coupled with the fluid solver to produce the desired corrected velocity and compute its divergence and L2 norm. Note that the gradient of this correction (Equation (2)) is known, so that a classical backpropagation can still be employed. However, if the network is only trained by evaluating the divergence level after a single time step, no information is given to the network on how its error can be amplified by the nonlinear advection at the next time step.

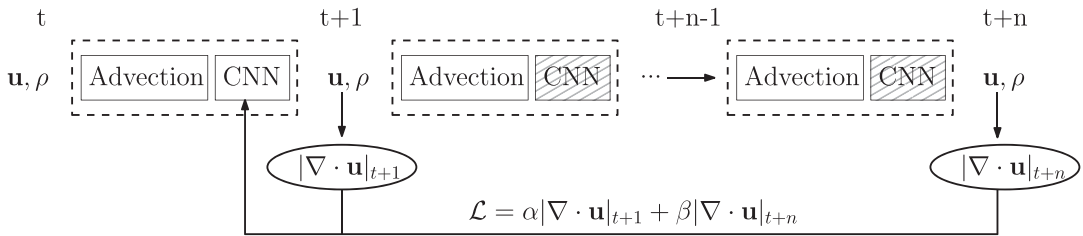


Figure 6. Physics-driven neural network learning strategy by combining a short-term loss and a long-term loss. The tiled box for the convolutional neural network indicates that the network parameters are frozen (i.e., they are the same as the one used in the network at time t).

To circumvent this issue, a new term is introduced into the PDNN loss. This new term, known as long-term loss, corresponds to the divergence field of the uncorrected velocity field obtained several time steps after the initial prediction (Figure 6). To do so, the network is trained inside the CFD solver: the prediction of the CNN at time t is fed through the advection and the CNN-based Poisson solver to compute the next n time steps. The resulting divergence is computed (*long-term* loss in Figure 6), and added to the initial divergence (*short-term* loss). The network weights are then updated using a gradient descent algorithm. Note that in the backpropagation algorithm, the chain rule described by Hecht-Nielsen (1992) is used, yet here the advection part is not differentiated, and therefore does not appear explicitly in the gradients. The number n of time steps to compute the long-term divergence term varies at each minibatch between 4 and 16. At each minibatch, a random number x is extracted from an uniform distribution $U(0, 1)$, 16 n time steps are performed when $x > 0.9$, whereas 4 iterations are performed if $x < 0.9$. This work fixed the number of iterations to 4 and 16 as a result of a trial-and-error process, although in other works the authors Ajuria-Illarramendi et al. (2021b) have looked at the influence of the number of *long-term* iterations. As a summary, the total loss function is:

$$\mathcal{L} = \frac{\alpha}{N} \sum |\nabla \cdot \mathbf{u}_{t+1}| + \frac{\beta}{N} \sum |\nabla \cdot \mathbf{u}_{t+n}|, \tag{4}$$

where α and β are two hyperparameters controlling the relative importance of the short-term and long-term losses and N is the number of mesh points in the domain.

As a semisupervised training procedure is considered, a complete training dataset is not necessary to train the network since no “ground truth” target is needed. However, physical initial conditions are still possible, in particular, to enforce the spatial coherence and structures relevant from a physical point of view, and to avoid overfitting. Thus, even for this PDNN, a training dataset was computed with the open-source code Mantaflow developed by Thuerey and Pfaff (2016). This dataset consists of closed boundary domains, with randomly initialized velocity fields and no density variations (i.e., the Richardson number is $R_i = 0$ for all training examples). Random geometries are placed in arbitrary locations, and velocity divergence sources are introduced in the domain to initialize the flow motion. (Further details about the training dataset can be found in Online Appendix D.) As a result, the dataset consists of 320 simulations on a 128×128 uniform Cartesian grid. Each simulation is run for 64 time steps. During training, a validation dataset consisting of 320 simulations with geometries that differ from the training dataset is used to ensure that overfitting is avoided. However, this validation dataset is still generated using the Euler equations ($Re \rightarrow \infty$) with constant density ($R_i = 0$), so that Sections 4 and 5 will assess the generalization capabilities on new flow regimes at finite Reynolds numbers (Section 4) and nonnull Richardson numbers (Section 5).

4. Flow Past a Cylinder

The four networks described in Section 3.1 will be compared on the Von Karman vortex street configuration. Training was performed with simulations with no viscosity (Euler equations), and domains

with 128^2 mesh points. Therefore, the flow around a cylinder represents an adequate test case to check the network generalization. Three flow regimes will be studied to characterize the network behavior ($Re = 100, 300,$ and $1,000$). The domain size is set to a $1,000 \times 2,000$ cell mesh, the input velocity U_0 is set to 1.0 m/s, and the kinematic viscosity (ν) is used to obtain the desired Reynolds numbers. Vorticity fields of the four studied networks, as well as for a reference Jacobi solver, are found in Figure 7 for $Re = 100$ (top), 300 (middle), and $1,000$ (bottom).

A first qualitative analysis of the results shows that networks with multiple scales are capable of generating *accurate* simulations, whereas the MonoScale network is not capable of correctly capturing the vortex shedding, as it dissipates the near-wake of the cylinder. Moreover, the SmallScale network also slightly differs from the other networks, particularly at low Reynolds numbers, as it underestimates the vortex magnitude. To quantify the network errors, this work will focus on the vortex shedding frequency. This frequency is usually expressed as a dimensionless frequency, noted as the Strouhal number:

$$St = \frac{fD}{U_0}. \quad (5)$$

The Strouhal number for the three studied Reynolds numbers is calculated with the fluid solver using the four studied networks and the Jacobi solver with 400 iterations. Results are displayed in Table 1.

These quantitative results confirm the conclusion drawn from the qualitative analysis of Figure 7. The multiple scale networks considerably outperform the MonoScale network, which struggles to correctly represent the vortex shedding. The MultiScale and Unet networks yield better results than the SmallScale for the studied flow regimes. This highlights the importance of choosing a sufficient amount of training

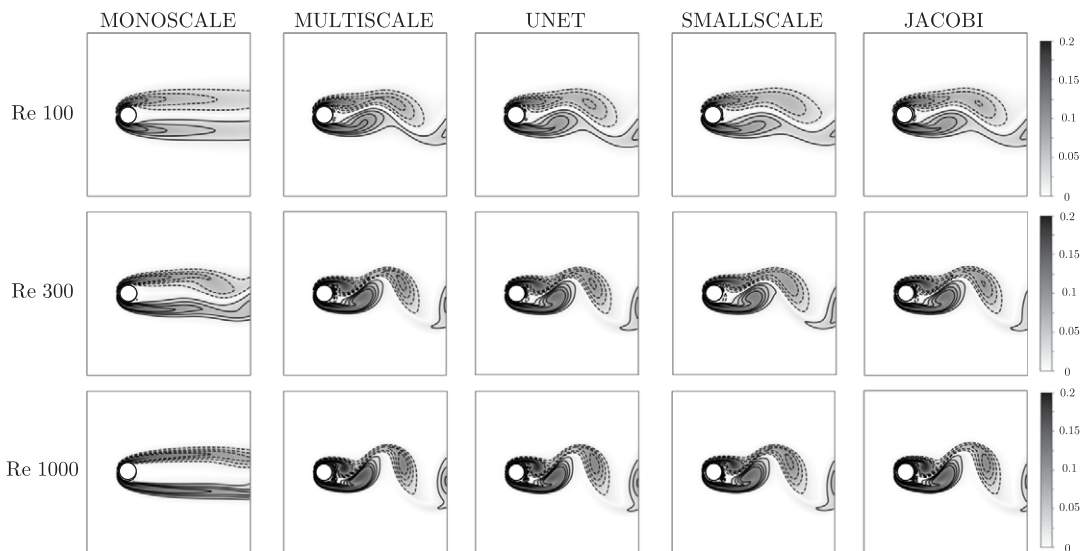


Figure 7. Iso-contours of vorticity in the z axis, for the Von Karman test case at Reynolds 100, 300, and 1,000, for the four studied networks and the reference Jacobi 400 solver. Dashed lines correspond to negative iso-contours, and the continuous lines correspond to the positive iso-contours.

Table 1. Strouhal values and relative error compared to the experimental values.

Reynolds	MonoScale	MultiScale	SmallScale	Unet	Jacobi
100	0.116	0.146	0.135	0.142	0.141
300	0.132	0.168	0.161	0.167	0.167
1,000	0.119	0.177	0.171	0.177	0.175

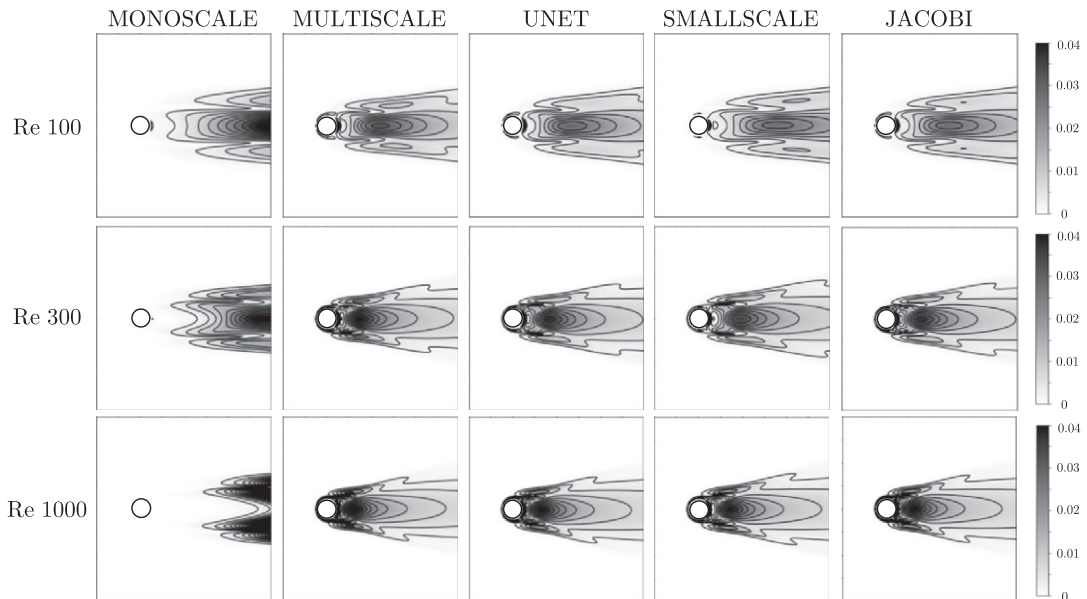


Figure 8. Iso-contours of the amplitude of the fundamental mode, for the Von Karman test case at Reynolds 100, 300, and 1,000, for the four studied networks and the reference Jacobi 400 solver.

parameters, as both MultiScale and Unet networks have almost three times more parameters. Results for the Unet and MultiScale networks are almost identical concerning the Strouhal number, with a small deviation on the Reynolds 100 test case. The predicted Strouhal number is similar to the reference value taken from the Jacobi solver, with a maximum difference of 3%.

Following previous work by Ajuria-Illarramendi et al. (2021a), a DMD, first introduced by Schmid (2010), is performed on the snapshots of the Von Karman vortex street. When analyzing the spatial modes, particular attention is paid to the fundamental mode, whose frequency corresponds to the vortex shedding Strouhal numbers displayed in Table 1. Figure 8 shows the amplitude of this first mode for the four studied networks, as well as the reference Jacobi solver. These modal structures confirm that the MonoScale network is not capable of capturing the vortex shedding, as no fluctuation occurs in the wake near the cylinder. Differences are more subtle between the rest of the networks, especially for the Reynolds 300 and 1,000 cases. However, the Unet network better matches the Jacobi 400 solver for the Reynolds 100 test case, showing a very similar spatial distribution in the near wake of the cylinder. As expected, differences between network architectures are more pronounced for lower Reynolds numbers ($Re = 100$), since it corresponds to cases further away from the training dataset at finite Reynolds number. Overall, the CNN approach provides accurate predictions in time, even for flow simulations far from the training dataset, where no viscous effects and unsteady wake were present. Yet, small discrepancies still exist, which calls for a new method ensuring a user-defined accuracy level of the CNN-based solver: this method is developed in Section 5.2, and is the core of the present study.

5. Buoyant Plumes

The analysis of the flow around a cylinder test case showed that networks can *successfully* interact with incompressible fluid solvers, yielding accurate results. However, to further assess the generalization capabilities of the approach, and to better understand the behavior of the different studied network architectures, this work will now investigate the buoyant plume configuration introduced in Section 2.2. As observed in a previous work (Ajuria Illarramendi et al. (2020)), buoyant plumes at high Richardson numbers can be tricky to solve with NNs, as small errors can lead to noncoherent fluid flows. To make the

configuration governed by only one dimensionless parameter (to be used as a generalization metric), no viscous effects are considered here (Euler equations), and only the density variations are considered through the Richardson number.

Thus, in this section, there will be no viscosity in the simulation, thus solving the Euler equations. Moreover, in order to compare the performance of the studied architectures, a detailed analysis of the network errors is required. Indeed, evaluating and comparing the time of inference of a method, in particular with NNs, is relevant only if performed at a fixed error level. Thus, this section intends to characterize the accuracy of each architecture, so that a fair comparison of performances can be achieved in Section 7.

5.1. Preliminary results without hybrid approach

First, the four networks described in Section 3.1 are tested on a 2D plume test case with a Richardson number $R_i = 14.8$, in order to evaluate the network accuracy and its generalization capabilities (since the training dataset was obtained for $R_i = 0$). Following the previous work of Ajuria Illarramendi et al. (2020), flows at high Richardson numbers can become numerically unstable and yield nonsymmetric simulations. Thus, $R_i = 14.8$ is chosen, corresponding to such sensible regime, where the network prediction can become critical for a stable and reliable flow simulation. As the Boussinesq approximation is used, the density variation should remain small and is set here to $\Delta\rho/\rho_0 = 0.01$. The lighter fluid is injected at the inlet boundary with the velocity $U_{in} = 0.01\text{m/s}$. The CFD domain size is $L_x = L = 512\text{m}$, discretized on a uniform Cartesian grid of 512×512 cells. The gravity is set to $g = 0.002\text{m/s}^2$, and the inlet radius is $R = 7,425\text{m}$, taken as the characteristic length to define the Richardson number. The divergence is normalized with the inlet velocity and the characteristic length, and time is normalized as well by the characteristic time needed for the plume to reach the top boundary at $R_i = \infty$, that is, in the pure jet configuration with no buoyancy. These normalized quantities are therefore defined as

$$\tilde{u} = Ru/U_{in} \text{ and } \tilde{t} = tU_{in}/L \quad (6)$$

The plume head position is defined as the highest point of the plume at a time t (Figure 9). On the vertical axis, this position is noted $\tilde{h}_y = h_y/L$. In the case of the plume impinging the cylindrical obstacle, the plume head position along the horizontal axis is also measured, denoted by $\tilde{h}_x = h_x/L$ (Figure 9). The vertical position \tilde{h}_y allows the identification and comparison of the plume velocity when rising due to both

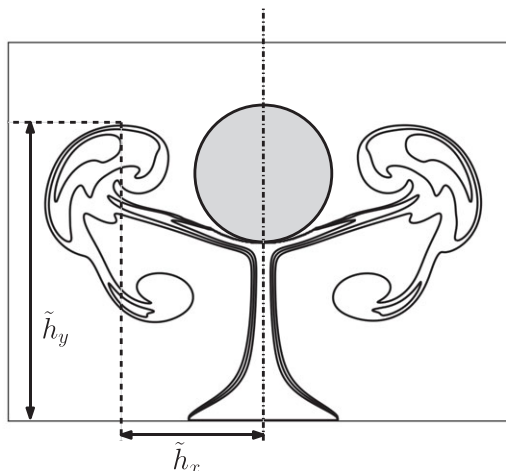


Figure 9. Sketch of the plume-cylinder configuration. \tilde{h}_x and \tilde{h}_y are the coordinates of the plume head location.

the advection by the jet and the buoyancy forces. The horizontal position \tilde{h}_x measures the flow deviation induced by the cylinder. This measurement is performed in both the left and right sides of the CFD domain in order to quantify the asymmetry of the flow field.

Analyzing the plume head's position for the several networks on the two test cases presented in Figure 2 reveals that deep NNs are able to solve the Poisson equation in an Euler incompressible solver with good precision, even in cases not seen during training (Figure 10; tested at $R_i = 14.8$). However, slightly different plume developments are obtained depending on the network architecture. As introduced in the previous work of Ajuria Illarramendi et al. (2020), the Jacobi solver is taken as the reference for the two studied test cases. The accuracy of the Jacobi solver solution increases with the number of iterations, but as the convergence rate does not follow a linear behavior, a trade-off between the number of iterations and the desired accuracy is usually necessary. The gray zones in Figure 10 represent the solution range obtained with between 200 and 10,000 iterations. Note that the plume development is slower with higher number of iterations. While the solution deviation of the test case without obstacle remains narrow, the cylinder test case shows a larger dispersion. Due to computational cost, it was, however, not possible to perform more than 10,000 iterations of the Jacobi solver.

For the no-cylinder case, the MonoScale network results in a faster head propagation, which falls outside the confidence interval of the Jacobi solvers, whereas all other network architectures produce an overall good physical behavior of the plume. The same trend is observed in the case with the obstacle, with a faster propagation and an early flow deviation for the MonoScale. Note that the symmetry is preserved by all networks since no difference is observed between left and right horizontal positions \tilde{h}_x . The final horizontal length of MonoScale does not match the ones of the other networks. This is due to a wrong prediction of the flow structure and roll-up of the plume by the MonoScale network. This is highlighted in Figure 11, showing the density iso-contours (white lines) and the percentiles of the divergence error (gray background field) at $\tilde{t} = 0.29$ (no-cylinder case) and $\tilde{t} = 0.41$ (case with obstacle). These percentiles show the spatial distribution of the error, without indicating the absolute value. The percentiles correspond to the divergence level of each snapshot individually, where the plotted value \tilde{h} represents the relative value of each spatial point normalized by the maximum value in the domain. A significant difference in the plume shape is observed between the MonoScale and the other networks. The MonoScale network predicts larger vortices, probably due to an inaccurate pressure field prediction which negatively affects the baroclinic torque. However, if the divergence error is analyzed, the MonoScale network follows the most regular distribution, whereas the rest of the networks seem to struggle in the surroundings of the cylinder. Moreover, the Unet network shows a quite irregular divergence distribution, with problems at boundaries.

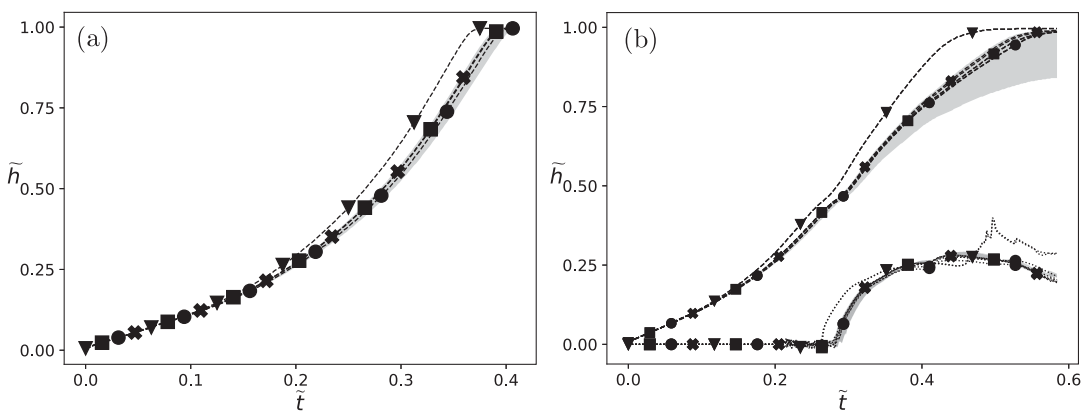


Figure 10. Plume head coordinates \tilde{h}_x (.....) and \tilde{h}_y (- - -) for the case without (a) and with (b) obstacle at $R_i = 14.8$ obtained by several networks: \blacktriangledown MonoScale, \blacksquare MultiScale, \bullet Unet, and \times SmallScale. The gray zones show the range of the plume head position obtained with the Jacobi solver, using between 200 and 10,000 iterations.

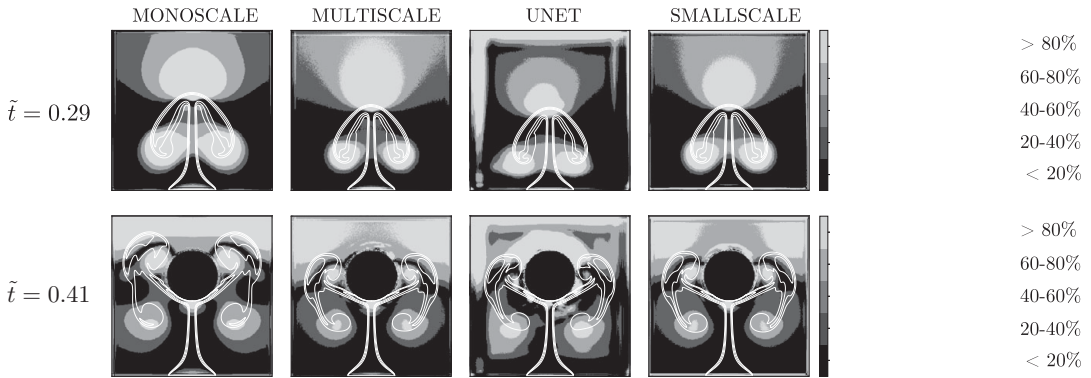


Figure 11. Divergence error percentiles and density iso-contours (in white) of the four studied networks and a reference Jacobi solver with 400 iterations, at time $\tilde{t} = 0.29$ for the case with no cylinder (top row) and time $\tilde{t} = 0.41$ for the cylinder case (bottom row).

The complete evolution of the plume for both cases are provided in Online Appendix A (Figures 23 and 24 in the Online Appendix).

5.2. Hybrid methodology

The main difficulty when building an ML-based CFD solver is to guarantee its precision and robustness, in particular for cases far from the learning database. To do so, Ajuria Illarramendi et al. (2020) introduced a hybrid strategy, where the network error is tracked over time. Depending on this error level, the network prediction is used as an initial guess for the Jacobi method to further improve its quality. Note that a similar procedure was also proposed by Hsieh et al. (2019), where an iterative solver is coupled to a deep linear network to guarantee the accuracy level of the network prediction. An overview of the hybrid method is shown in Figure 12: after the advection step, the pressure correction term is predicted by the deep NN architectures proposed in Section 3.1. At this point, no guarantee is given on the precision of the method, possibly leading to large errors or even numerical instabilities, as shown by Ajuria Illarramendi et al. (2020). The hybrid strategy is then applied, based on an error associated with the divergence velocity field, denoted \mathcal{E}_∞ . Ajuria Illarramendi et al. (2020) proposed to use the maximum of the divergence velocity field, that is, $\mathcal{E}_\infty = \max |\nabla \cdot \mathbf{u}|$, showing a significant improvement of the solver accuracy and robustness. Note, however, that no physical evidence of this choice is given. In particular, there is no formal proof that reducing the maximum of the velocity divergence leads to a more physical behavior of the simulation. One objective of this paper is to propose a proper choice of \mathcal{E} that guarantees a physical behavior of the simulation, toward a fast, reliable, and robust CNN-based incompressible solver.

Previously, the head location has been employed for measuring the error, yet it is case-dependent and requires a reference simulation, which prevents this type of error to be used in new unknown simulations.

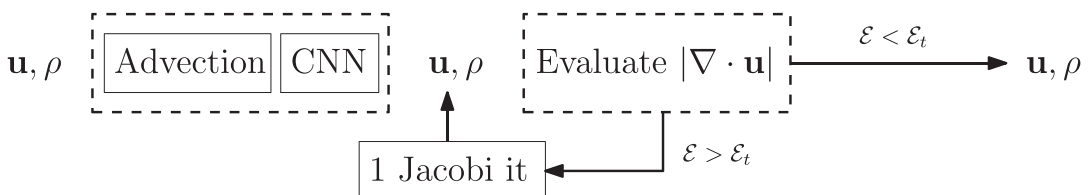


Figure 12. Sketch of the hybrid method, which is activated depending on the error \mathcal{E} compared with the threshold value \mathcal{E}_t .

The percentiles of the divergence error displayed in Figure 11 highlight the locations in the flow field where the NNs have difficulties, but it cannot be used as an absolute error level as well. Thus, the objective is to define an absolute unsupervised error \mathcal{E} , which can be evaluated for any plume case without reference, and is consistent with the physical-based error given by the head position. Here, two measures are considered: (a) the maximum of the divergence velocity field $\mathcal{E}_\infty = \max(|\nabla \cdot \mathbf{u}|)$, and (b) the mean of this divergence field, $\mathcal{E}_1 = \text{mean}(|\nabla \cdot \mathbf{u}|)$. This type of error control is typically used in iterative algorithms, such as the Jacobi method or the CG method, further described in the work of Hestenes et al. (1952). This assumes a uniform error distribution, which implies that the maximum or mean error value is representative of the overall flow behavior. However, this hypothesis does not necessarily hold for NNs, since their error distribution is usually nonuniform: NNs may lead to a very high localized error, as displayed in Figure 11, near a BC for example.

Figure 13 shows these two errors \mathcal{E}_1 and \mathcal{E}_∞ for both test cases without (a) and with (b) obstacle, for the various network architectures. Interestingly, the “best” network is different for the two types of errors. With the error based on \mathcal{E}_∞ , the MultiScale network outperforms the other architectures, in particular for the case with cylinder, whereas the SmallScale leads to the highest error. Regarding \mathcal{E}_1 , the Unet network gives a low error in both cases, similar to the MultiScale network. However, based on this error, this is the MonoScale network, which is the worst network in both situations, with the highest mean divergence. These test cases reveal how critical is the choice of the absolute error \mathcal{E} when comparing and then choosing network architectures.

5.3. Controlling the error level using the hybrid approach

Before assessing the performance of each NN in Section 7, it is essential to ensure that the hybrid approach is able to control the physical targets \tilde{h}_x and \tilde{h}_y . To do so, this section intends to determine if \mathcal{E}_1 or \mathcal{E}_∞ is able to control the physical behavior of the simulation. In other words, a proper definition of the error \mathcal{E} should guarantee that all networks hybridized with the same threshold $\mathcal{E} < \mathcal{E}_t$ lead to the exact same evolution of \tilde{h}_x and \tilde{h}_y in time.

First, a test is carried out in order to choose between \mathcal{E}_1 and \mathcal{E}_∞ as a proper indicator of the flow behavior, characterized by \tilde{h}_x and \tilde{h}_y . The lowest values obtained in Figure 13 are chosen as threshold values. With these thresholds, the four different networks are evaluated and compared with the Jacobi method using a number of iterations also driven by the same threshold. Results on the divergence errors and head’s position are displayed in Figures 14 and 15 for the two test cases. First, it can be noticed that the specified thresholds are correctly followed by the networks and the Jacobi methods, since for the maximum error (Figures 14a and 15a) and mean error (Figures 14b and 15b), the divergence curves

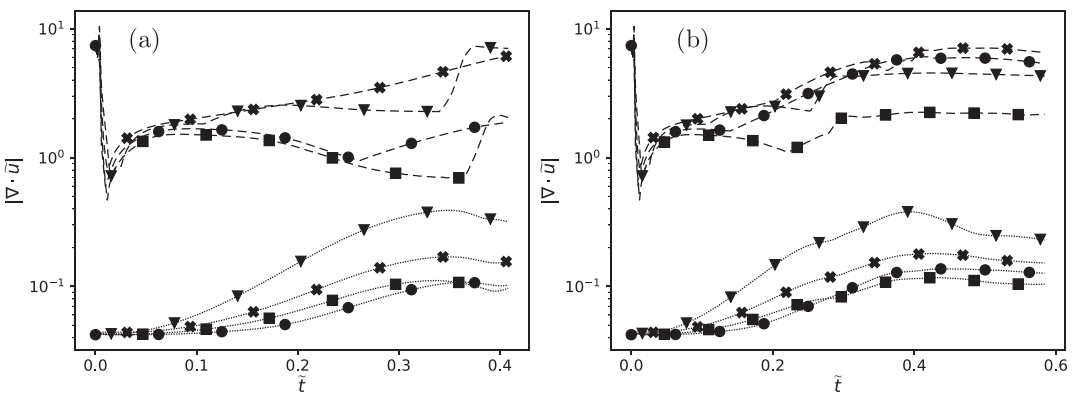


Figure 13. \mathcal{E}_∞ (---) and \mathcal{E}_1 (.....) for the case without (a) and with (b) cylinder obtained by several architectures: \blacktriangledown MonoScale, \blacksquare MultiScale, \bullet Unet, and \times SmallScale.

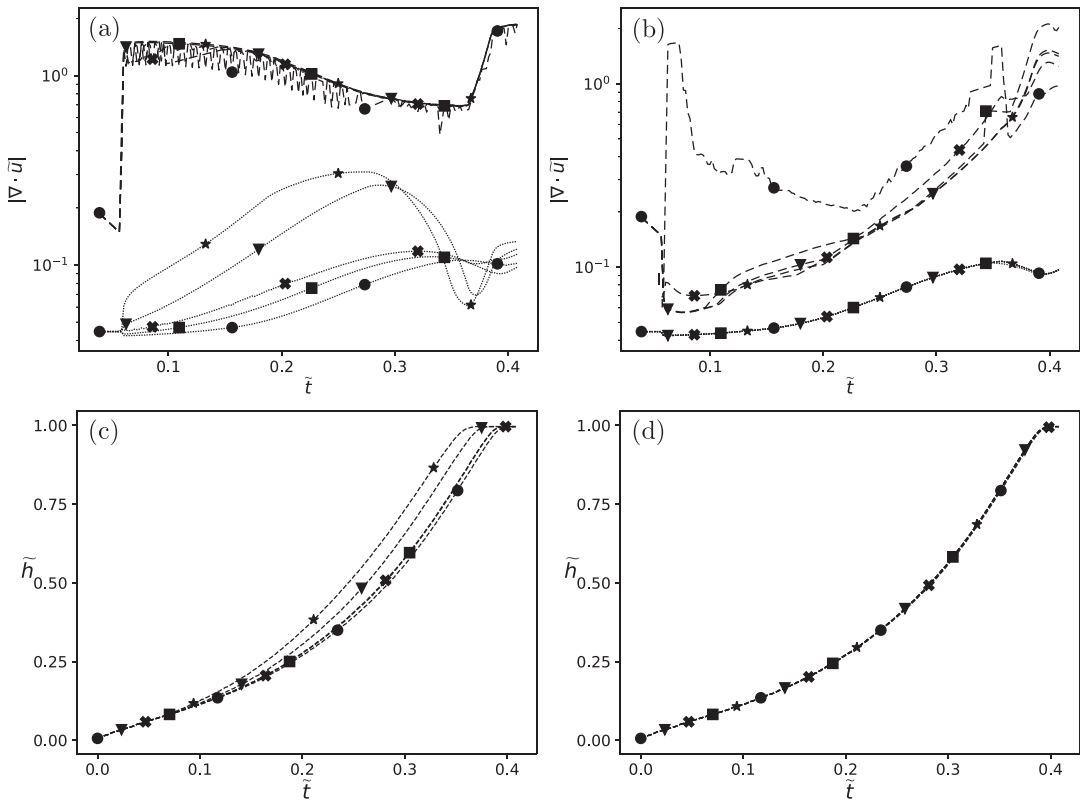


Figure 14. \mathcal{E}_∞ (---) and \mathcal{E}_1 (.....), and plume head position \tilde{h}_y (bottom), where (a,c) correspond to $\mathcal{E}_t = \min(\mathcal{E}_\infty)$ and (b,d) $\mathcal{E}_t = \min(\mathcal{E}_1)$ evaluated on the no cylinder test case, at an $R_i = 14.8$ obtained by several networks: \blacktriangledown MonoScale, \blacksquare MultiScale, \times SmallScale, and \bullet Unet, as well as the \star Jacobi solver.

are superimposed. This demonstrates how the hybrid approach is able to guarantee a level of accuracy for the NN predictions, which are otherwise unreliable. The resulting head positions \tilde{h}_x and \tilde{h}_y , representative of the physical behavior of the simulations, are also provided (Figs. 14c,d and 15c,d). Interestingly, it is found that imposing the same mean error to all methods yields exactly the same time evolution of the physical targets, for both cases with and without obstacles. However, imposing the maximum error as a threshold leads to different, yet close, simulation behaviors. Typically, the same trend as in Figure 10 is observed, with a too rapid rise of the plume when predicted by the Jacobi method (\star) and the MonoScale network (\blacktriangledown). This faster convection speed is highlighted in the second case, where the plume is impinging the cylinder obstacle at shorter times for these two methods, resulting in an early flow deviation characterized by a nonnull \tilde{h}_x position (Figure 15c). Note, however, that prescribing a lower maximum error, typically $\mathcal{E}_t = 0.18$, yields also a similar plume evolution for all networks (Online Appendix B). The choice of the optimum threshold value \mathcal{E}_t is not known a priori, although a small iterative process can rapidly show a value where the error is low enough to stop seeing any variations on the flow structures. Note that the question of choosing \mathcal{E}_t , or similarly choosing the number of iterations, is not restricted to AI-based solvers, but to all iterative solvers such as Jacobi or CG methods.

5.4. Analysis of the error distribution

To further understand how the error definition (\mathcal{E}_1 or \mathcal{E}_∞) is driving the overall behavior of the simulations, the time evolution of the spatial distribution of the error is studied. To do so, at a given

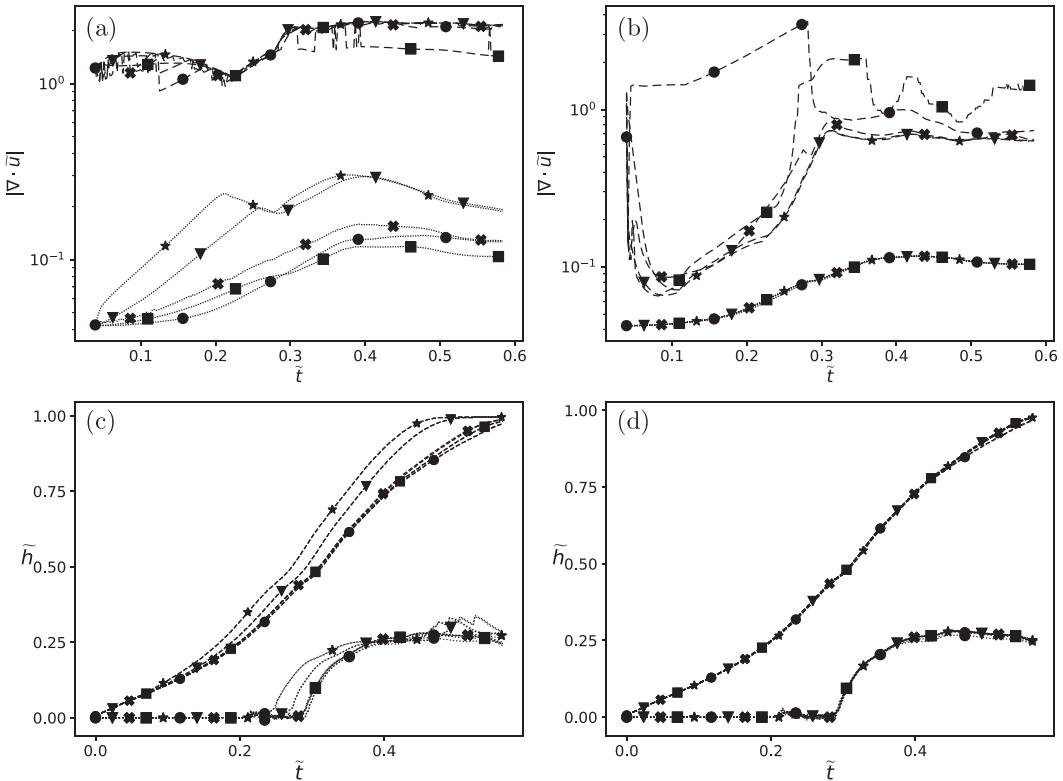


Figure 15. \mathcal{E}_∞ (---) and \mathcal{E}_1 (·····), and plume head position (\tilde{h}_y, \tilde{h}_x) (bottom), where (a,c) correspond to $\mathcal{E}_t = \min(\mathcal{E}_\infty)$ and (b,d) $\mathcal{E}_t = \min(\mathcal{E}_1)$ evaluated on the cylinder test case, at an $R_t = 14.8$ obtained by several networks: \blacktriangledown MonoScale, \blacksquare MultiScale, \times SmallScale, and \bullet Unet, as well as the \star Jacobi solver.

normalized time \tilde{t} , the distribution of the divergence error is computed by a kernel density estimation (KDE; as described in the work of John and Langley (1995)). The evolution of the KDE for the plume case without obstacle is displayed in Figure 16. The same analysis on the second test case corresponding to the plume impinging the cylinder is performed in Online Appendix C. Results are displayed at four time steps: $\tilde{t} = 0.1, 0.2, 0.29$, and 0.39 . Both \mathcal{E}_∞ (top) and \mathcal{E}_1 (bottom) have been tested. The corresponding threshold values \mathcal{E}_t have been set to the lowest values of the error obtained by all networks without hybrid approach (Section 5.1). When following the maximum divergence threshold (top), it can be noticed that the error distributions differ depending on the method employed. A significant difference is observed at $\tilde{t} = 0.1$ for the Jacobi iterative method, for which the error distribution is spreading over a wide range, from 0 to 0.3. In comparison, all NNs at this normalized time produce an error close to 0.05, resulting in a sharp unimodal density function. Note that such behavior is ideal since the error is well controlled in space, yet the threshold value ($\mathcal{E}_t = 1.5$ at $\tilde{t} = 0.1$; Figure 14a) is not directly linked to the most probable error ($\mathcal{E} \approx 0.05$ at $\tilde{t} = 0.1$). At the same time $\tilde{t} = 0.1$, using \mathcal{E}_1 (bottom part of Figure 16) generates exactly the same error distribution for all approaches, including the Jacobi iterative method. This result generalizes the conclusions established from Figures 14 and 15: when specifying the error as the spatially averaged \mathcal{E}_1 error on the CFD domain, the hybrid approach yields exactly the same error distribution (whatever the method used or the network architecture), and consequently produces the same evolution of the plume head position \tilde{h} , that is, the same physical behavior of the simulation. Interestingly, the error distribution is similar to the case following \mathcal{E}_∞ : the probability density function of the divergence error is unimodal, with a peak located at $|\nabla \cdot \tilde{u}| \approx 0.05$. However, compared with the use of the maximum error, here the threshold value ($\mathcal{E}_t \approx 0.05$ at $\tilde{t} = 0.1$; Figure 14b) directly corresponds to the most probable error.

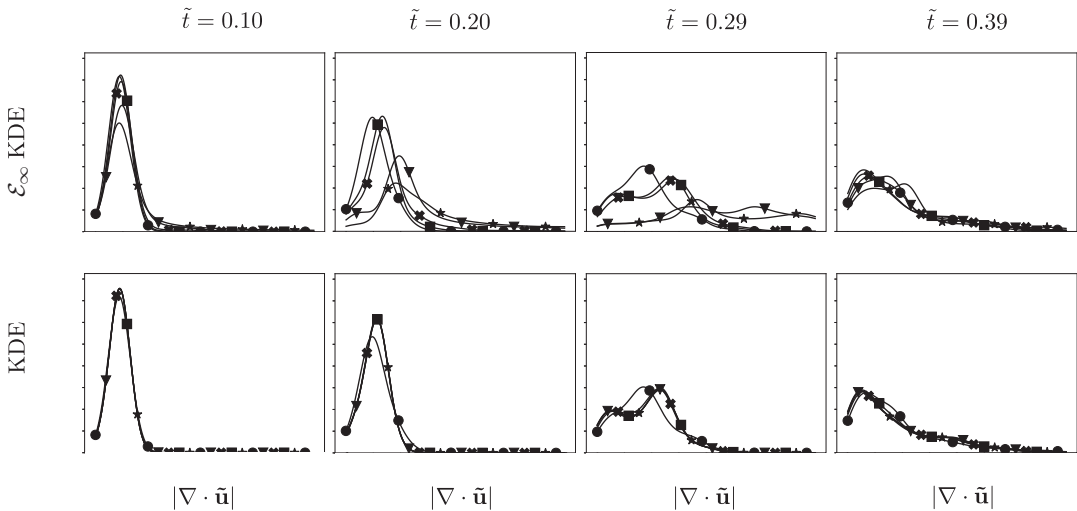


Figure 16. Kernel density estimation at four times ($\tilde{t} = 0.10, 0.20, 0.29,$ and 0.39) of the cases where $\mathcal{E} = \mathcal{E}_\infty$ (top) and $\mathcal{E} = \mathcal{E}_1$ (bottom) of the no cylinder test case, at an $R_i = 14.8$ obtained by several networks: \blacktriangledown MonoScale, \blacksquare MultiScale, \times SmallScale, and \bullet Unet, as well as the \star Jacobi solver.

As the simulation is running, a clear advantage of using \mathcal{E}_1 can be observed at $\tilde{t} = 0.2$. Not only it yields the same distribution error for all methods, but this distribution is close to the one obtained at previous times (e.g., $\tilde{t} = 0.1$), revealing that the hybrid approach is consistent in time. On the contrary, the use of \mathcal{E}_∞ leads to very different error distributions. It is worth noting that the Jacobi method (\star) and the MonoScale (\blacktriangledown) exhibit a distribution shift toward higher error values, thus explaining their higher mean values (Figure 14a). This result is consistent with the unphysical behavior of these two simulations (too fast rising of the plume), since the evolutions of \bar{h} are strongly correlated with the mean value error (Figure 14b,d).

At larger time steps ($\tilde{t} = 0.29$ and 0.39), the error distribution is spreading for all cases toward higher error levels (from 0 to 0.3 for most cases). Again, defining the error and threshold values using \mathcal{E}_∞ (top) is not able to control the whole error distribution. The Unet (\bullet) produces the most unimodal distribution with a peak at a low error of 0.05, whereas the Jacobi method (\star) and the MonoScale network (\blacktriangledown) generate a flat error distribution with larger errors up to 0.6. However, following \mathcal{E}_1 (bottom) is still producing consistent error distributions, with low discrepancies between the various methods, including the MonoScale and Jacobi methods. A slight shift of the error distribution can be observed, with a most probable error close to 0.1 (instead of 0.05 for early times). This is due to the fact that the present test case is performed with a varying threshold value $\mathcal{E}_t(\tilde{t})$: this value is chosen as the lowest error value obtained without the hybrid approach, which is therefore not fixed in time. Figure 14b reveals that the threshold value of the mean error is actually increasing in time, from 0.05 to early time steps toward 0.1 at the end of the simulation ($\tilde{t} = 0.6$).

Overall, the description of the error \mathcal{E} and threshold value \mathcal{E}_t using the spatially averaged error over the CFD domain has shown a remarkable ability to control the whole distribution error in time and space, whatever the method used. For all times, the distribution is unimodal, with a most probable error in the range 0.05 – 0.1, in good agreement with the threshold value of the mean error employed for the hybrid approach. This is not surprising since the error distribution is a unimodal symmetric probability function, for which the most probable and mean values are equal. In other words, controlling the mean error is constraining the whole error distribution, whereas following the maximum value \mathcal{E}_∞ only constrain the distribution tail. This reduced constrain over the error distribution, seems insufficient to control the physics of the system, in particular here the rising speed of the plume head. Note that similar trends and conclusions are observed with the plume impinging the cylindrical obstacle (Figure 30 in Online Appendix C), even if the error distributions are no more unimodal: the hybrid approach based on the

mean error is still able to control the whole distribution error whatever the network architecture. For completeness, the whole spatial distributions of the error are displayed for the two cases (with and without obstacle) and both error definitions (\mathcal{E}_1 and \mathcal{E}_∞) in Online Appendix A.

Finally, this study reveals that the NNs hybridized with a Jacobi method are able to guarantee the accuracy level of the whole simulation in time, in particular when choosing an error defined as the spatially averaged divergence error in the CFD domain. Consequently, in the following, the hybrid approach based on \mathcal{E}_1 is chosen to ensure that all networks have a similar accuracy with the same plume head evolution in time, a pre-requisite to perform a fair comparison of the network performances.

6. 3D Plume Test Case

The same methodology is now applied to 3D cases. In this section, the plume test case without a cylinder at the Richardson number 14.8 is shown in Figure 17. The domain consists of 128^3 mesh points, with a similar layout as the 2D case.

For this configuration, a new network based on the Unet architecture with 3D convolutions (resulting in 1.34×10^6 trainable parameters) has been trained on a similar dataset as for the 2D case, but with the domain sizes of 64^3 mesh points. Looking at the plume evolution in the middle part of Figure 17, the Unet network simulations qualitatively match the reference Jacobi solver (with 400 iterations; top part of Figure 17). However, some asymmetries at the plume head can be appreciated, especially at $\tilde{t} = 0.39$ (displayed by the black arrow). To correct these numerical instabilities, the hybrid strategy is used with the threshold $\mathcal{E}_t = \mathcal{E}_1$, where \mathcal{E}_1 corresponds to the mean divergence of the reference Jacobi solver with 400 iterations at each simulation time step.

When applying the hybrid strategy, Jacobi iterations are just needed in the first iterations, which correspond to the most difficult time steps for the network, whereas for the rest of the simulation, none or

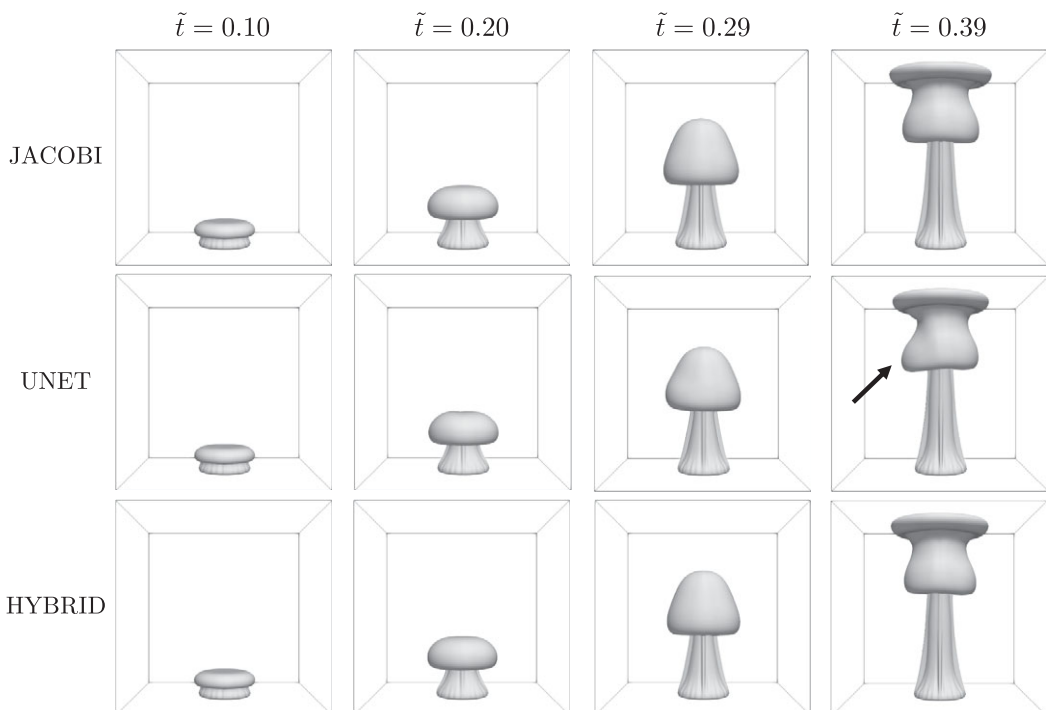


Figure 17. Comparison between the Unet network and a reference Jacobi 400 solver for a 3D plume without obstacle at Richardson 14.8.

one Jacobi iteration is needed. Thus, the asymmetries found on the plume surface disappear, matching the reference Jacobi simulations. Note that this hybrid strategy, while guaranteeing the accuracy and stability of the simulation, has an extra cost of only 14% compared with the Unet network for this case, yielding an overall simulation around 10 times faster than the Jacobi method. This exemplifies that the present hybrid strategy has significant benefits in many different configurations (2D, 3D, flows unseen during training, etc.) by providing both fast and reliable simulations. To better understand this performance, a detailed analysis of the acceleration produced by the four previously studied network architectures on various mesh sizes is provided in Section 7.

7. Performance Assessment

In this section, the performance of NNs is assessed for the time of inference, that is, the time needed to produce the pressure correction. The analysis is innovative in two ways: (a) a fair performance comparison is performed using the hybrid approach, allowing the assessment of the time of inference at a fixed error level whatever the method (Jacobi or NN architecture), and (b) these performances are evaluated for several network architectures and grid sizes. Note that even if CNNs are theoretically capable of dealing with CFD domains of arbitrary size and resolution, they have in practice difficulties which such cases, probably because of boundary effects, as hinted by Kayhan and van Gemert (2020): here again, the hybrid approach is useful to guarantee the error level of the solution when changing the grid size. To do so, as suggested in Section 5.3, the mean divergence error is used to define the threshold \mathcal{E}_t . All simulations were performed on the same GPU card, namely here an NVIDIA Tesla V100 with 32 GB of memory. Note that the performance evaluation on multiple GPUs is out of the scope of the present study.

To measure the fluid solver performance, the following times are defined:

- t_{inf} : Inference time taken by an NN or the Jacobi solver to output one pressure field when the divergence field is inputted, that is, not taking into account the time to correct the velocity field or the time spent on the extra Jacobi iterations of the hybrid solver.
- t_p : Time taken to perform the entire pressure projection step. This includes the time to perform the first pressure inference, the extra Jacobi iterations in the hybrid process, as well as the correction of the velocity field.
- t_{it} : Time taken to perform an entire iteration of the fluid solver, that is, the advection and pressure projection steps.

7.1. Network comparison

Using the hybrid strategy, a nontrivial trade-off has to be made between accuracy (usually implying more complex and deeper architectures) and fast inference time (requiring small networks). In practice, a less accurate network with a fast inference time will require more Jacobi iterations to reach the desired accuracy level, thus limiting drastically its performance. Comparing the performance of the several network architectures used in this work will provide a first insight on this trade-off, in order to establish guidelines for future developments of AI-based solvers.

Consequently, the performance assessment of the NNs is split into two steps: (a) assess the inference time t_{inf} of each CNN to produce the output, and (b) evaluate the number of Jacobi iterations needed by each network, using the CNN prediction as initial guess, to reach the target accuracy. Table 2 shows the inference time of each network, and reveals that t_{inf} is not directly associated with the number of parameters of the network. Indeed, whereas MonoScale, MultiScale, and Unet networks have the same number of parameters, they also have very different inference times. Similarly, the SmallScale network contains three times fewer parameters compared with the Unet network, but still requires more time to output its prediction. This is due to the specific architectures of the Unet and MultiScale networks which involve several scales: filters and their associated parameters at low resolution (capturing the large scales of the flow field) are applied on reduced inputs, therefore limiting the number of operations to perform, and thus reducing t_{inf} (Figure 18b). Thus, the more parameters at lower scale, the faster the network

Table 2. Inference time of each network to produce a 2D (512² grid) and 3D (128³ grid) pressure field without Jacobi iterations, as well as the inference time of a Jacobi solver.

	Time	MonoScale	MultiScale	Unet	SmallScale	Jacobi (One iteration)
2D	t_{inf} (ms)	14.6	11.3	5.25	5.35	2.31
	$t_{\text{inf}}/t_{\text{it}}$ (%)	29.1	24.1	12.9	13.1	6.10
3D	t_{inf} (ms)			107.8		6.9
	$t_{\text{inf}}/t_{\text{it}}$ (%)			59.8		8.7

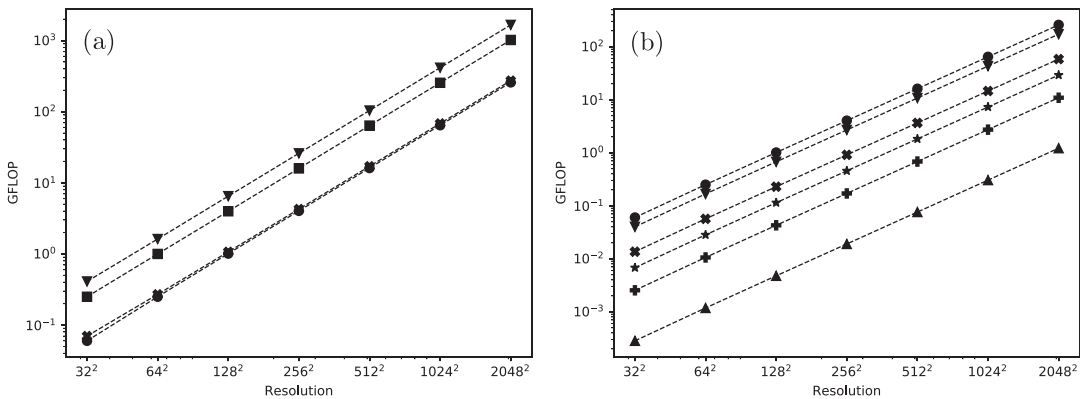


Figure 18. Evolution of the number of floating-point operations (in giga units) with the 2D domain size (varying from 1,024 to 4.2×10^6 cells) needed in a single network inference for the four studied networks (a): \blacksquare MultiScale, \times SmallScale, \blacktriangledown MonoScale, and \bullet Unet, and for the scales composing the Unet network (b): \bullet Unet, \blacktriangledown n^2 , \blacktriangledown $n^2_{1/2}$, \times $n^2_{1/2} \times n^2_{1/2}$, \star $n^2_{1/4} + n^2_{1/8}$, and \blacktriangle $n^2_{1/16}$.

is. This is confirmed by the associated number of floating-point operations (FLOP) for each network architecture, displayed in Figure 18 for each network and several grid sizes, from 1,024 to 4.2×10^6 cells. It reveals that the FLOP number depends linearly on the grid resolution and is directly correlated with the inference times of Table 2: the five scales of the Unet network result in a low FLOP number, and therefore a reduced t_{inf} , comparable to the SmallScale network with three times fewer parameters. The MultiScale network also contains three scales, but most parameters are concentrated at the larger resolution, which is therefore dominating the FLOP number and inference time. Only a small gain is observed for this architecture compared with a standard MonoScale network.

Since the several studied networks have different error levels, the number of Jacobi iterations to reach the desired accuracy level is decisive in the overall performance of the AI-based code. Figure 19 shows the number of Jacobi iterations needed to reach the threshold \mathcal{E}_t , for both 2D plume cases without (Figure 19a) and with (Figure 19b) obstacle. As expected, the classical Jacobi solver (\star) and the MonoScale network (\blacktriangledown) need numerous iterations, typically between 300 and 400. Using a multiscale architecture (either MultiScale or SmallScale) improves significantly this situation, since only 100 to 200 iterations are needed for the same accuracy level. Note that for the case with obstacle, the MultiScale network needs almost no Jacobi iteration after the plume has impinged the cylindrical obstacle ($\tilde{r} > 0.3$). In contrast, the Unet outperforms all networks, with almost no Jacobi iteration for all cases. However, it requires 50 – 100 iterations after the plume has impinged the cylinder ($\tilde{r} > 0.3$), which is consistent with the previous finding that the Unet has difficulties close to walls (Figure 11). As a general conclusion, all networks outperform the classical Jacobi solver, proving that using NNs as an initial guess to the iterative solver is relevant.

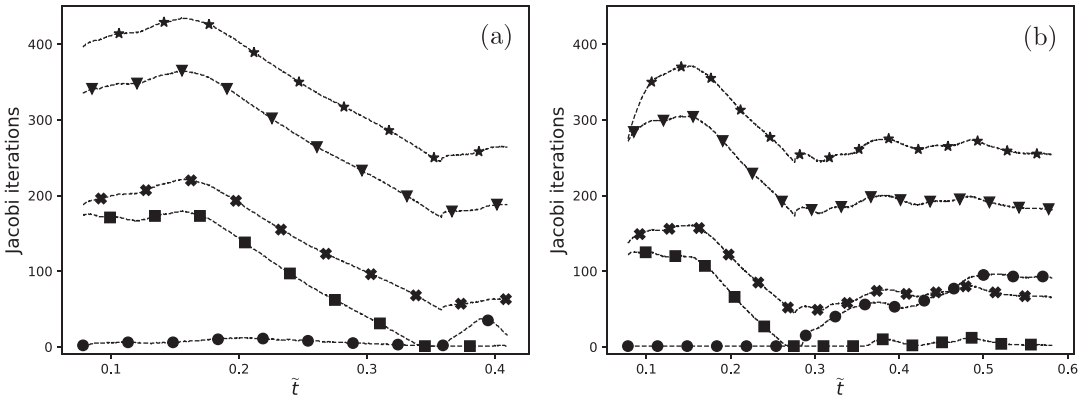


Figure 19. Number of Jacobi iterations needed to ensure $\mathcal{E}_t = \min(\mathcal{E}_1)$ on the noncylinder (a) and the cylinder (b) 2D test cases with various networks: \blacktriangledown MonoScale, \blacksquare MultiScale, \times SmallScale, and \bullet Unet, as well as the \star Jacobi solver.

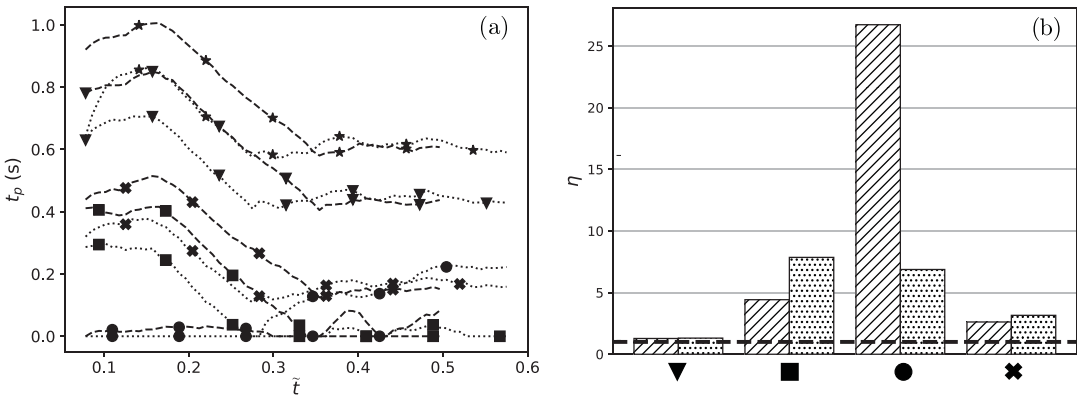


Figure 20. Time t_p (a) and acceleration factor η (b) for the noncylinder (---) and cylinder 2D test case (.....), for the four studied networks: \blacktriangledown MonoScale, \blacksquare MultiScale, \times SmallScale, and \bullet Unet, as well as the \star Jacobi solver.

However, this AI-based computational strategy is efficient only if the time needed to produce the initial guess by the CNN leads to a total time lower than using the Jacobi solver alone.

To verify this, the time to solve the Poisson equation (t_p) is now compared for all methods in Figure 20a. As a comparison, the computational time to solve the advection equation is $t_{ad} = 35.5\text{ms}$. The bar plot (Figure 20b) highlights the acceleration factor $\eta = t_p^{jacob}/t_p^{network}$ for each network architecture, where t_p^{jacob} and $t_p^{network}$ are the computational times t_p of the Jacobi method and the hybridized network, respectively. First, it can be noticed that all AI-based approaches are faster than the Jacobi solver alone, which proves the efficiency of the proposed computational strategy using CNN to predict the solution of the Poisson equation. As expected, the MonoScale only provides a small gain, about 1.2 – 1.3, since (a) it needs numerous iterations of the Jacobi solver to achieve the desired accuracy level, and (b) it has the largest inference time (Table 2). Note that the latter is evaluated at 14.6ms, which indicates that even for this network with a large inference time, it is still negligible compared to the computational time of Jacobi iterations. Interestingly, the large (MultiScale) and small (SmallScale) multiscale networks achieve a comparable acceleration factor, respectively, $\eta \approx 3.7$ and 2.4 without obstacle, but differ when the plume impinges the cylinder, with $\eta \approx 7.5$ and $\eta \approx 3.1$. Indeed, the larger network achieves a better accuracy, thus limiting the number of Jacobi iterations. However, the accuracy-inference time trade-off is

not trivial when dealing with deep NNs embedded in a CFD solver, as in test cases where few Jacobi iterations are needed, the SmallScale network could outperform the larger MultiScale network, thanks to the smaller network inference time. Finally, the Unet architecture outperforms all other methods in the case without obstacle with an acceleration factor $\eta=26.1$. As the Unet has difficulties handling boundaries, its performance drops in the case with obstacle, yet still accelerating the CFD solver by a factor $\eta=6.6$, slightly below the MultiScale network performance. Note, however, that for longer simulations, the MultiScale network would further outperform the Unet since requiring no Jacobi iteration for times larger than $\tilde{t}=0.3$.

Similar conclusions can be drawn from the 3D performance test, where the advection time is $t_{ad}=72.1\text{ms}$. Looking at the results shown in Table 2, the network inference time is higher compared to a single Jacobi iteration, due to the addition of further trainable parameters. Note that for the 3D network, no particular optimization of the Unet architecture has been performed, so that a reduction of the network inference time is still possible. Nevertheless, when using the hybrid strategy following the reference Jacobi solver, the network only needs to perform iterations in the initial simulation time steps (needing up to 1,000 Jacobi iterations in the first two time steps), whereas none or one Jacobi iteration is needed for the rest of the simulation. This results in an acceleration factor $\eta=12.9$ comparing the hybrid strategy with the reference Jacobi solver with 400 iterations. Again, this is a demonstration that a hybrid strategy coupling AI and CFD can provide significant speedup, even for 3D cases which require larger NNs.

7.2. Evaluation and analysis of the performance on various grid sizes

The previous performance assessment was made at a fixed resolution of 2.6×10^5 cells. Recent work by Kayhan and van Gemert (2020) revealed that networks can encode spatial location and resolution on the training dataset, which can lead to poor performance when applied to a different resolution than the one used during the training phase (2.6×10^5 cells in this work). This drawback is here tackled by the hybrid approach, which ensures the desired accuracy level whatever the grid size. It is also found in this work that the present CNN architectures are capable of generalizing to different grid sizes, even if no clear explanation can be given for this specificity. As a reminder, Figure 18 showed that the number of operations per network evolves linearly with the grid size. However, Figure 21a shows a different behavior when looking at t_{inf} , with two distinct regimes: (a) the network performances follow the same behavior as in Figure 18 when the domain size is large enough, with a linear increase with grid size, and (b) t_{inf} remains constant for small grids. This phenomenon is related to the CPU overhead, as the time taken by the GPU kernels is inferior to the command transmission between the CPU and the GPU, which depends only on the network architecture but not the grid size. Therefore, networks with a simpler architecture result in a lower CPU overhead, and so

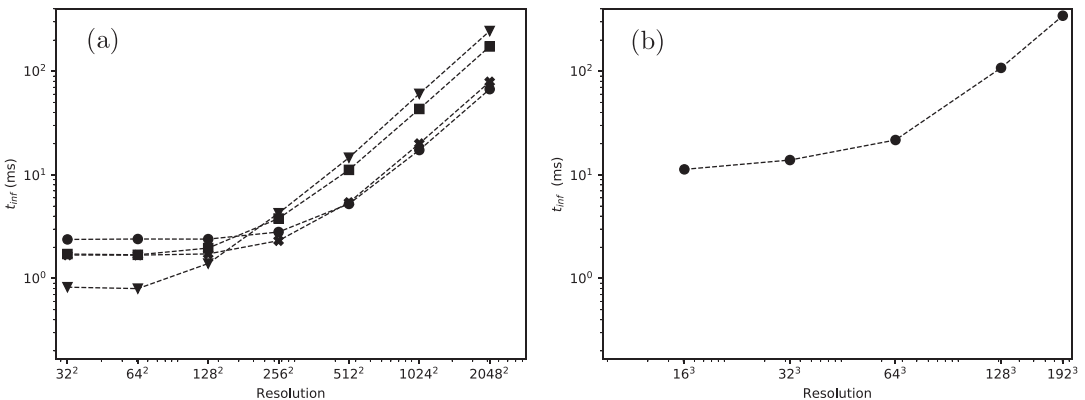


Figure 21. Time taken to perform the network prediction t_{inf} (a) for the four studied networks: \blacktriangledown MonoScale, \blacksquare MultiScale, \times SmallScale, and \bullet Unet, on a 2D grid size varying from 1,024 to 4.2×10^6 and (b) for the 3D Unet network (\bullet) on a 3D grid size varying from 16^3 to 192^3 .

a faster inference. It should be noted how, for example, the MultiScale and SmallScale networks show a similar inference time when the GPU kernels are saturated, as the architecture and layer sizes are the same. This saturation should be taken into account for small resolution test cases, for instance, by designing dedicated simple architectures. A similar behavior is found on the 3D network (Figure 21b) with the distinct linear and saturated behaviors. However, due to the 3D convolutions, the overhead regime saturates at higher inference times. Interestingly, the transition between these two regimes appears for the Unet network at approximately the same number of cells: 65×10^3 cells in 2D, corresponding to 256^2 and 33×10^3 cells, corresponding to 32^3 . Note that because of memory capability of the V100 GPU card, no 3D meshes with more than 7×10^6 cells (192^3) can be tested. In addition, the inference on multiple GPUs is out of the scope of the present study, but would be a crucial step to demonstrate the potential of this AI-CFD hybrid method in an HPC context.

In order to better understand the link between the network architecture and its performances, the time taken for different scales is displayed in Figure 22, for both the MultiScale (a) and Unet (b) networks when the spatial resolution is varying. This figure shows the evolution of the three main scales of the MultiScale network, depending on the spatial resolution. As expected, the largest scale takes the major part of the inference time for both networks, performing most of the operations due to the larger size of the domain. Moreover, it is shown that the smaller scales are more prone to the command overhead, which makes the saturated regime occur on a larger resolution range, for instance, the scale $n/4$ of the MultiScale is saturated until the resolution 512×512 , and the scale $n/16$ of the Unet until $1,024 \times 1,024$. Exploiting smaller scales with more parameters could open the path to new computational strategies with large computational domains divided into multiple smaller subdomains of the appropriate size, for example, adapted to each scale of the network.

8. Conclusions

This work focuses on the resolution of the Poisson equation with CNNs applied to incompressible fluid flows. First, the well-known MultiScale and Unet architectures are trained on Euler constant-density flows, and then analyzed, and compared to a simpler architecture as well as with a traditional Jacobi solver. These networks are tested on the viscous flow around a cylinder, where the multiple scale networks (Unet and MultiScale) are found capable of correctly simulating the vortex shedding at various Reynolds numbers, thus revealing generalization capabilities to unseen flow physics. To further study the network-solver interactions, a plume test case, with and without obstacle, is studied for various Richardson numbers. For this case, asymmetries and numerical instabilities may appear when applying the trained network at $R_i = 0$ to nonnull Richardson flows. In order to ensure a user-defined accuracy level and to

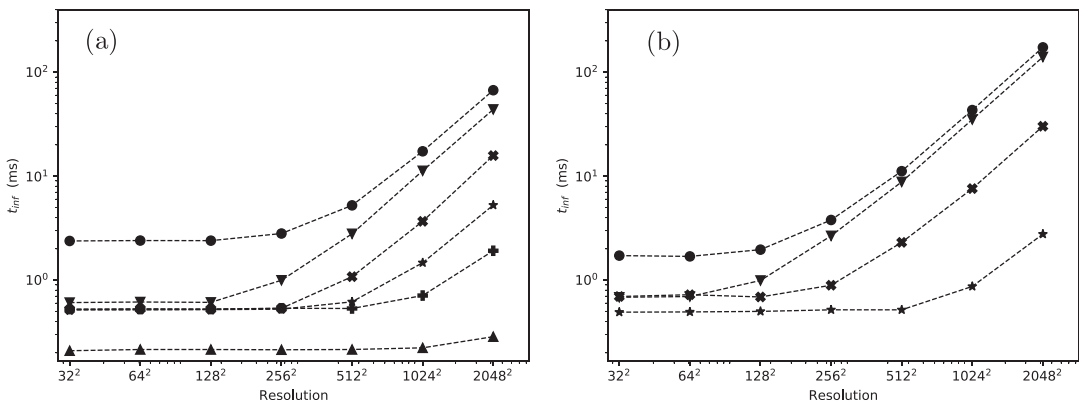


Figure 22. Time taken by each scale ($\bullet n^2$, $\blacktriangledown n_{1/2}^2$, $\times n_{1/2}^2$, $\star n_{1/4}^2$, $+ n_{1/8}^2$, and $\blacktriangle n_{1/16}^2$) for the MultiScale (a) and Unet (b) networks, to perform a single inference on a 2D grid size varying from 1,024 to 4.2×10^6 cells.

avoid numerical instabilities, a hybrid strategy is used where the NNs are coupled with a Jacobi solver when a threshold error criterion is not satisfied. This work shows that a threshold based on the mean divergence of the flow field ensures consistent physical behavior of the fluid flow, which guarantees both accuracy and reliability of the AI-based solver. Moreover, a 3D plume is computed as well, showing how the introduced methodology can be extrapolated to more realistic test cases: again, the hybrid approach allows fast and reliable 3D simulations, whereas the network alone yields numerical asymmetries in the flow. The various networks are compared in both accuracy and inference time when using the hybrid strategy. It is found that the network behavior varies with the studied case. When there is no obstacle in the domain, the Unet network outperforms the other networks both in accuracy and performance. However, when an obstacle is introduced, its accuracy decreases because of difficulties close to walls, resulting in a simulation time superior to the MultiScale network. For both cases, the MultiScale and Unet networks provide speedups of order 5–7 in 2D and 10 in 3D, demonstrating the potential of such a method. Moreover, it is shown that the multiple scales of the Unet and MultiScale networks result in a faster inference time, as long as the resolution is larger than 256×256 . For lower resolutions, the CPU overhead controls the inference time, enabling the simpler MonoScale network to obtain the fastest inference time. In conclusion, this work has revealed the potential of coupling CNN with multiple scales to a CFD solver to produce fast and reliable flow predictions, as well as providing useful insights on the performances of NNs depending on their architecture, to be used as guidelines for future works coupling CFD and AI.

Acknowledgments. The calculations were performed using HPC resources from CALMIP on Olympe (Grant 2020-p20035). The presented fluid solver was initially developed by Antonio Alguacil in collaboration with the Jolibrain company, who also offered their technical support throughout this project. Finally, the Physics-Based Simulation group (Thuerey group) of the Technical University of Munich should be acknowledged for their support.

Supplementary Materials. To view supplementary material for this article, please visit <http://doi.org/10.1017/dce.2022.2>.

Data Availability Statement. Readers interested in replicating the results in this work may find the code available at https://gitlab.isae-supaero.fr/daep/fluidnet_supaero.git, which has been tested to work on NVIDIA Tesla V100 32-GB GPUs.

Author Contributions. Conceptualization: E.A.I., M.B., and B.C.; Data curation: E.A.I.; Formal analysis: E.A.I. and M.B.; Funding acquisition: M.B. and B.C.; Investigation: E.A.I.; Methodology: E.A.I. and M.B.; Project administration: M.B.; Resources: M.B. and B.C.; Software: E.A.I.; Supervision: M.B. and B.C.; Validation: E.A.I.; Visualization: E.A.I.; Writing—original draft: E.A.I. and M.B.; Writing—review and editing: E.A.I., M.B., and B.C.

Funding Statement. This research was supported by grants from the CERFACS and the ISAE-SUPAERO.

Competing Interests. The authors declare no competing interests.

References

- Ahn C-S, Bang B-H, Kim M-W, James SC, Yarin AL and Yoon SS (2019) Theoretical, numerical, and experimental investigation of smoke dynamics in high-rise buildings. *International Journal of Heat and Mass Transfer* 135, 604–613.
- Ajuria Illarramendi E, Alguacil A, Bauerheim M, Misdariis A, Cuenot B and Benazera E (2020) Towards an hybrid computational strategy based on deep learning for incompressible flows. In *AIAA AVIATION 2020 FORUM*, p. 3058. Virtual event.
- Ajuria-Illaramendi E, Bauerheim M and Cuenot B (2021a) Analysis of downscaled branches and receptive field on a CNN-based incompressible solver. In *74th Annual Meeting of the APS Division of Fluid Dynamics*. Phoenix, AZ.
- Ajuria-Illaramendi E, Bauerheim M and Cuenot B (2021b) Embedding temporal error propagation on CNN for unsteady flow simulations. In *35th Conference on Neural Information Processing Systems (NeurIPS)*. Machine Learning and the Physical Sciences Workshop. Virtual event.
- Alguacil A, Bauerheim M, Jacob MC and Moreau S (2020) Predicting the propagation of acoustic waves using deep convolutional neural networks. In *AIAA AVIATION 2020 FORUM*, p. 2513.
- Alguacil A, Pinto WG, Bauerheim M, Jacob MC and Moreau S (2021) Effects of boundary conditions in fully convolutional networks for learning spatio-temporal dynamics. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 102–117. Virtual event.
- Allocca C, Chen Q and Glicksman LR (2003) Design analysis of single-sided natural ventilation. *Energy and Buildings* 35(8), 785–795.
- Baque P, Remelli E, Fleuret F and Fua P (2018) Geodesic convolutional shape optimization. In *International Conference on Machine Learning (ICML '18)*, pp. 472–481. Stockholm, Sweden.

- Brunton SL, Noack BR and Koumoutsakos P** (2019) Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics* 52, 477–508.
- Brunton SL, Proctor JL and Kutz JN** (2016) Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences* 113(15), 3932–3937.
- Buzbee BL, Golub GH and Nielson CW** (1970) On direct methods for solving Poisson's equations. *SIAM Journal on Numerical Analysis* 7(4), 627–656.
- Chassignet EP, Cenedese C and Verron J** (2012) *Buoyancy-Driven Flows*. Cambridge University Press. Verron, Cambridge
- Cheng L, Ajuria Illarramendi E, Bogopolsky G, Bauerheim M and Cuenot B** (2021) Using neural networks to solve the 2D Poisson equation for electric field computation in plasma fluid simulations. *Preprint*, arXiv:2109.13076.
- Chu M and Thuerey N** (2017) Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics* 36(4), 1–14.
- Cizmas PG, Richardson BR, Brenner TA, O'Brien TJ and Breault RW** (2008) Acceleration techniques for reduced-order models based on proper orthogonal decomposition. *Journal of Computational Physics* 227(16), 7791–7812.
- Deng X, Mao M, Tu G, Zhang H and Zhang Y** (2012) High-order and high accurate CFD methods and their applications for complex grid problems. *Communications in Computational Physics* 11(4), 1081–1102.
- Dissanayake M and Phan-Thien N** (1994) Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering* 10(3), 195–201.
- Dong W, Liu J, Xie Z and Li D** (2019) Adaptive neural network-based approximation to accelerate Eulerian fluid simulation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–22. Denver, CO.
- Elsken T, Metzger JH and Hutter F** (2019) Neural architecture search: a survey. *Journal of Machine Learning Research*, 20(1), 1997–2017.
- Fukami K, Fukagata K and Taira K** (2019) Super-resolution reconstruction of turbulent flows with machine learning. *Journal of Fluid Mechanics* 870, 106–120.
- Fukami K, Fukagata K and Taira K** (2020) Assessment of supervised machine learning methods for fluid flows. *Theoretical and Computational Fluid Dynamics*, pp. 1–23. Elsevier.
- Fukami K, Maulik R, Ramachandra N, Fukagata K and Taira K** (2021) Global field reconstruction from sparse sensors with Voronoi tessellation-assisted deep learning. *Preprint*, arXiv:2101.00554.
- Geng Z and Wang Y** (2020) Automated design of a convolutional neural network with multi-scale filters for cost-efficient seismic data classification. *Nature Communications* 11(1), 1–11.
- George Jr WK, Alpert RL and Tamanini F** (1977) Turbulence measurements in an axisymmetric buoyant plume. *International Journal of Heat and Mass Transfer* 20(11), 1145–1154.
- Gonzalez-Garcia R, Rico-Martinez R and Kevrekidis I** (1998) Identification of distributed parameter systems: a neural net based approach. *Computers & Chemical Engineering* 22, S965–S968.
- Gray DD and Giorgini A** (1976) The validity of the Boussinesq approximation for liquids and gases. *International Journal of Heat and Mass Transfer* 19(5), 545–551.
- Harlow FH and Welch JE** (1965) Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids* 8(12), 2182–2189.
- He K, Zhang X, Ren S and Sun J** (2016) Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778. Las Vegas, NV.
- Hecht-Nielsen R** (1992) Theory of the backpropagation neural network. In *Neural Networks for Perception*. Elsevier, pp. 65–93. Fairfax, Virginia, USA.
- Hestenes MR and Stiefel E** (1952) Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards* 49(6), 409–436.
- Hochreiter S and Schmidhuber J** (1997) Long short-term memory. *Neural Computation* 9(8), 1735–1780.
- Hornik K, Stinchcombe M, White H, et al.** (1989) Multilayer feedforward networks are universal approximators. *Neural Networks* 2(5), 359–366.
- Hosseini S and Fasel HF** (2018) An efficient, high-order method for solving Poisson equation for immersed boundaries: combination of compact difference and multiscale multigrid methods. *Journal of Computational Physics* 374, 912–940.
- Hsieh JT, Zhao S, Eismann S, Mirabella L and Ermon S** (2019) Learning neural PDE solvers with convergence guarantees. In *International Conference on Learning Representations*. Vancouver, BC, Canada.
- Hunt BR** (1973) The application of constrained least squares estimation to image restoration by digital computer. *IEEE Transactions on Computers* 100(9), 805–812.
- Hunt G and Van den Bremer T** (2011) Classical plume theory: 1937–2010 and beyond. *IMA Journal of Applied Mathematics* 76(3), 424–448.
- Jacobi CG** (1845) Ueber eine neue auflösungsart der bei der methode der kleinsten quadrate vorkommenden lineären gleichungen. *Astronomische Nachrichten* 22(20), 297–306.
- John GH and Langley P** (1995) Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 338–345. Montréal, Québec, Canada.

- Kayhan OS and van Gemert JC** (2020) On translation invariance in CNNs: convolutional layers can exploit absolute spatial location. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14274–14285. Seattle, WA.
- Kelley HJ** (1960) Gradient theory of optimal flight paths. *ARS Journal* 30(10), 947–954.
- Kim B, Azevedo VC, Thuerey N, Kim T, Gross M and Solenthaler B** (2019) Deep fluids: a generative network for parameterized fluid simulations. In *Computer Graphics Forum*, Vol. 38. pp. 59–70. Wiley Online Library.
- Kochkov D, Smith JA, Alieva A, Wang Q, Brenner MP and Hoyer S** (2021) Machine learning-accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences* 118(21), e2101784118.
- Krizhevsky A, Sutskever I and Hinton GE** (2012) ImageNet classification with deep convolutional neural networks. In Pereira F, Burges CJC, Bottou L and Weinberger KQ (eds), *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., pp. 1097–1105. Vancouver, BC, Canada.
- Lagaris IE, Likas A and Fotiadis DI** (1998) Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks* 9(5), 987–1000.
- Lapeyre CJ, Misdariis A, Cazard N, Veynante D and Poinso T** (2019) Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates. *Combustion and Flame* 203, 255–264.
- LeCun Y, Bottou L, Bengio Y, Haffner P, et al.** (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11), 2278–2324.
- Lee H and Kang IS** (1990) Neural algorithm for solving differential equations. *Journal of Computational Physics* 91(1), 110–131.
- Lee S and You D** (2019) Data-driven prediction of unsteady flow over a circular cylinder using deep learning. *Journal of Fluid Mechanics* 879, 217–254.
- Lou Y, He Z, Jiang H and Han X** (2019) Numerical simulation of two coalescing turbulent forced plumes in linearly stratified fluids. *Physics of Fluids* 31(3), 037111.
- Luo Z and Li Y** (2011) Passive urban ventilation by combined buoyancy-driven slope flow and wall flow: parametric CFD studies on idealized city models. *Atmospheric Environment* 45(32), 5946–5956.
- Mai-Duy N** (2005) Solving high order ordinary differential equations with radial basis function networks. *International Journal for Numerical Methods in Engineering* 62(6), 824–852.
- Mathieu M, Couprie C and LeCun Y** (2016) Deep multi-scale video prediction beyond mean square error. In *4th International Conference on Learning Representations*. San Juan, Puerto Rico.
- McCormick SF** (1989) *Multilevel Adaptive Methods for Partial Differential Equations*. SIAM. Philadelphia, PA, USA.
- McKenney A, Greengard L and Mayo A** (1995) A fast Poisson solver for complex geometries. *Journal of Computational Physics* 118(2), 348–355.
- Moon S, Kim W-T and Ostriker EC** (2019) A fast Poisson solver of second-order accuracy for isolated systems in three-dimensional Cartesian and cylindrical coordinates. *The Astrophysical Journal Supplement Series* 241(2), 24.
- Morimoto M, Fukami K, Zhang K, Nair AG and Fukagata K** (2021) Convolutional neural networks for fluid flow analysis: toward effective metamodeling and low dimensionalization. *Theoretical and Computational Fluid Dynamics* 35(5), 633–658.
- Morton B** (1959) Forced plumes. *Journal of Fluid Mechanics* 5(1), 151–163.
- Murata T, Fukami K and Fukagata K** (2020) Nonlinear mode decomposition with convolutional neural networks for fluid dynamics. *Journal of Fluid Mechanics* 882, A13. <https://doi.org/10.1017/jfm.2019.822>
- Otto SE and Rowley CW** (2019) Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems* 18(1), 558–593.
- Özbay AG, Hamzehloo A, Laizet S, Tzirakis P, Rizos G and Schuller B** (2021) Poisson CNN: convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh. *Data-Centric Engineering* 2, E6. <https://doi.org/10.1017/dce.2021.7>
- Pawar S, Ahmed SE, San O and Rasheed A** (2019) Data-driven recovery of hidden physics in reduced order modeling of fluid flows. *Physics of Fluids*, 32(3), 036602.
- Pfaff T, Fortunato M, Sanchez-Gonzalez A and Battaglia P** (2020) Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*. Virtual event.
- Quarteroni A, Sacco R and Saleri F** (2010) *Numerical Mathematics*, Vol. 37. Springer Science & Business Media. Luxembourg, Luxembourg.
- Raissi M and Karniadakis GE** (2018) Hidden physics models: machine learning of nonlinear partial differential equations. *Journal of Computational Physics* 357, 125–141.
- Raissi M, Perdikaris P and Karniadakis GE** (2017a). Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations. *Preprint*, arXiv:1711.10561.
- Raissi M, Perdikaris P and Karniadakis GE** (2017b) Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations. *Preprint*, arXiv:1711.10566.
- Ronneberger O, Fischer P and Brox T** (2015) U-net: convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, pp. 234–241. Munich, Germany.
- Rosenblatt F** (1958) The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* 65(6), 386–408.
- Rowley CW and Dawson ST** (2017) Model reduction for flow analysis and control. *Annual Review of Fluid Mechanics* 49, 387–417.

- Rudy SH, Brunton SL, Proctor JL and Kutz JN** (2017) Data-driven discovery of partial differential equations. *Science Advances* 3(4), e1602614.
- Rumelhart DE, Hinton GE and Williams RJ** (1986) Learning representations by back-propagating errors. *Nature* 323(6088), 533–536.
- Saad Y** (2003) *Iterative Methods for Sparse Linear Systems*, Vol. 82. SIAM. Philadelphia, PA, USA.
- Schmid PJ** (2010) Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics* 656, 5–28.
- Schmidt W** (1941) Turbulent propagation of a stream of heated air. *Zeitschrift für Angewandte Mathematik und Mechanik* 21, 265–278.
- Selle A, Fedkiw R, Kim B, Liu Y and Rossignac J** (2008) An unconditionally stable Maccormack method. *Journal of Scientific Computing* 35(2–3), 350–371.
- Shabbir A and George WK** (1994) Experiments on a round turbulent buoyant plume. *Journal of Fluid Mechanics* 275, 1–32.
- Shirvany Y, Hayati M and Moradian R** (2009) Multilayer perceptron neural networks with novel unsupervised training method for numerical solution of the partial differential equations. *Applied Soft Computing* 9(1), 20–29.
- Simonyan K and Zisserman A** (2015) Very deep convolutional networks for large-scale image recognition. *Preprint*, arXiv:1409.1556.
- Southwell RV** (1956) *Relaxation Methods in Theoretical Physics*. Clarendon Press. London, UK.
- Subramaniam A, Wong ML, Borker RD, Nimmagadda S and Lele SK** (2020) Turbulence enrichment using physics-informed generative adversarial networks. *Preprint*, arXiv:2003.01907.
- Sun L, Gao H, Pan S and Wang J-X** (2020) Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering* 361, 112732.
- Taira K, Brunton SL, Dawson ST, Rowley CW, Colonius T, McKeon BJ, Schmidt OT, Gordeyev S, Theofilis V and Ukeiley LS** (2017) Modal analysis of fluid flows: an overview. *AIAA Journal*, 4013–4041.
- Thurey N and Pfaff T** (2016) Mantaflow. Available at <http://mantaflow.com/>.
- Tompson J, Schlachter K, Sprechmann P and Perlin K** (2017) Accelerating Eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, Vol. 70, pp. 3424–3433. Available at <https://www.jmlr.org/>. Sydney, Australia.
- Turner J** (1969) Buoyant plumes and thermals. *Annual Review of Fluid Mechanics* 1(1), 29–44. San Mateo, CA, USA.
- Turner JS** (1979) *Buoyancy Effects in Fluids*. Cambridge University Press.
- Um K, Holl P, Brand R, Thurey N, et al.** (2020) Solver-in-the-loop: learning from differentiable physics to interact with iterative PDE-solvers. In *Thirty-fourth Conference on Neural Information Processing Systems*. Virtual event.
- Van Maele K and Merci B** (2006) Application of two buoyancy-modified $k-\epsilon$ turbulence models to different types of buoyant plumes. *Fire Safety Journal* 41(2), 122–138.
- Wandel N, Weinmann M and Klein R** (2020) Learning Incompressible Fluid Dynamics from Scratch—Towards Fast, Differentiable Fluid Models that Generalize. *Preprint*, arXiv:2006.08762.
- Wesseling P** (1995) Introduction to multigrid methods. *Technical Report*. Hampton, VA: Institute for Computer Applications in Science and Engineering.
- Wiewel S, Becher M and Thurey N** (2019) Latent space physics: towards learning the temporal evolution of fluid flow. In *Computer Graphics Forum*, Vol. 38, pp. 71–82. Wiley Online Library
- Xiao X, Zhou Y, Wang H and Yang X** (2018) A novel CNN-based Poisson solver for fluid simulation. *IEEE Transactions on Visualization and Computer Graphics* 26(3), 1454–1465.
- Xie Y, Franz E, Chu M and Thurey N** (2018) tempoGAN: a temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Transactions on Graphics* 37(4), Article no. 95, 1–15.
- Yang C, Yang X and Xiao X** (2016) Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds* 27(3–4), 415–424.
- Zdravkovich M** (1981) Review and classification of various aerodynamic and hydrodynamic means for suppressing vortex shedding. *Journal of Wind Engineering and Industrial Aerodynamics* 7(2), 145–189.
- Zeldovich YB** (1937) Limiting laws of freely rising convection currents. *Zhurnal Eksperimentalnoii Teoreticheskoy Fizika* 7, 1463–1465.
- Zhang ZJ and Duraisamy K** (2015) Machine learning methods for data-driven turbulence modeling. In *22nd AIAA Computational Fluid Dynamics Conference*, p. 2460. Dallas, TX.
- Zhou X, Luo KH and Williams JJ** (2001) Large-eddy simulation of a turbulent forced plume. *European Journal of Mechanics—B/Fluids* 20(2), 233–254.
- Zoph B, Vasudevan V, Shlens J and Le QV** (2018) Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710. Salt Lake City, UT.

Cite this article: Ajuria Illarramendi E, Bauerheim M and Cuenot B (2022). Performance and accuracy assessments of an incompressible fluid solver coupled with a deep convolutional neural network. *Data-Centric Engineering*, 3: e2. doi:10.1017/dce.2022.2