Coupling Machine Learning Local Predictions with a Computational Fluid Dynamics Solver to Accelerate Transient Buoyant Plume Simulations

Clément Caron, Philippe Lauret, Alain Bastide

Twelfth International Conference on Computational Fluid Dynamics (ICCFD12), Kobe, Japan, July 14-19, 2024

Coupling Machine Learning Local Predictions with a Computational Fluid Dynamics Solver to Accelerate Transient Buoyant Plume Simulations

C. Caron*,**, P. Lauret* and A. Bastide*
Corresponding author: clement.caron@univ-reunion.fr * Department of Sustainable Built Environment, PIMENT lab, University of Reunion, La Réunion, France.
** Research & Development team, INTEGRALE Ingénierie, Saint-Gilles les Hauts, La Réunion, France.

Abstract:
Data-driven methods demonstrate considerable potential for accelerating the inherently expensive computational fluid dynamics (CFD) solvers. Nevertheless, pure machine-learning surrogate models face challenges in ensuring physical consistency and scaling up to address real-world problems. This study presents a versatile and scalable hybrid methodology, combining CFD and machine learning, to accelerate long-term incompressible fluid flow simulations without compromising accuracy. A neural network was trained offline using simulated data of various two-dimensional transient buoyant plume flows. The objective was to leverage local features to predict the temporal changes in the pressure field in comparable scenarios. Due to cell-level predictions, the methodology was successfully applied to diverse geometries without additional training. Pressure estimates were employed as initial values to accelerate the pressure-velocity coupling procedure. The results demonstrated an average improvement of 94% in the initial guess for solving the Poisson equation. The first pressure corrector acceleration reached a mean factor of 3, depending on the iterative solver employed. Our work reveals that machine learning estimates at the cell level can enhance the efficiency of CFD iterative linear solvers while maintaining accuracy. Although the scalability of the methodology to more complex cases has yet to be demonstrated, this study underscores the prospective value of domain-specific hybrid solvers for CFD.

Keywords: Computational fluid dynamics, Machine learning, buoyant plume, incompressible flow, Poisson equation.

1 Introduction

Computational fluid dynamics (CFD) techniques enable the simulation of fluid flow, which is valuable for a wide range of scientific challenges. This flexible numerical approach can model various physical phenomena across different time and space scales based on the governing equations. As a result, CFD finds applications in many areas, such as aerodynamics, turbomachinery, hydrology, chemical processes, meteorology, and buildings [1, 2]. However, the computational time required to solve unsteady large-scale real-world problems remains a significant bottleneck. Despite the development of efficient numerical approaches and the growth of computational power, CFD is inaccessible for numerous applications [3, 4]. In this context, the CFD community is paying greater attention to machine learning algorithms to accelerate solvers by creating cost-effective models [3, 5, 6, 7, 8]. The machine learning field encompasses various algorithms that extract valuable information from data, leading to applications such as pattern recognition or surrogate modeling [9]. Data-driven models can provide fast predictions, rendering them a desirable alternative to high-fidelity physics-based simulations. Nevertheless, further research is required to identify effective ways to combine CFD with machine learning to speed up simulations while maintaining accuracy [10].

Although machine learning algorithms are not novel, they have substantially advanced recently. A combination of factors, including abundant data, more capable hardware, increased computational power, and the development of efficient algorithms, has revealed the potential of machine learning. Deep learning, which relies on deep neural networks, has gained significant attention due to its demonstration of state-of-the-art capabilities in addressing numerous intricate challenges, such as image or speech recognition [11, 12]. The *scientific machine learning* domain aims to bridge the gap between machine learning

and scientific computing. Concerning CFD, a primary focus is the development of data-driven models that achieve a more favorable balance between accuracy and computational cost [6, 10]. However, several challenges must be overcome, including model generalizability, interpretability, scalability, and data efficiency [8, 10, 13]. Therefore, the research effort is shifting toward frameworks that enforce the underlying physics [13]. For example, physics-informed neural networks [14] and neural operators [15] hold significant potential for accelerating traditional scientific computing. Additionally, *hybrid approaches*, which combine physics-based solvers with machine learning, are considered a more practical strategy than completely replacing traditional numerical methods [10]. While hybrid approaches may be more complex to develop and yield less impressive computational advantages than data-driven black-box models, they could guarantee essential physical consistency.

The existing literature suggests several strategies for hybridizing numerical and neural architectures for CFD. For instance, it is advisable to employ machine learning models to refine a cost-effective physical approximation rather than predict an absolute value [16]. In this context, super-resolution techniques have been employed to reconstruct high-fidelity fields from low-fidelity data obtained, e.g., from coarse-grid simulations [17]. However, despite model accuracy, transient simulations present an additional challenge due to the inevitable accumulation of errors when making consecutive data-driven predictions [18, 19, 20, 21]. While unrolling over multiple time steps in training enhances model performance, a more robust hybridization is required for long-term predictions [16, 17, 22]. For example, a hybrid methodology alternating traditional CFD and machine-learning-based time series has been demonstrated to control this error effectively [19].

An alternative approach to achieve consistent results is to use the model output as the initial condition for a numerical solver, ensuring convergence constraints [23]. Providing better initial guesses for costly solver sub-components, such as the solution of the pressure Poisson equation for incompressible fluids, is also decisive. Thus, data-driven predictions have demonstrated effectiveness as initial values for traditional iterative methods used in solving linear systems [22, 24, 25]. However, these studies involved complex deep learning architectures predicting a field of interest for the entire domain simultaneously, which may struggle to scale up to real-world cases.

Concurrently, some studies have concentrated on developing *local approaches*. These strategies entail training models to map a local flow description to a local prediction. Consequently, models are typically simpler as they operate within a constrained dimensional space. Also, they can be applied to any domain size at inference time. Local approaches are therefore promising for improving scalability, generalizability, and data efficiency [17, 18, 19, 20].

In light of the aforementioned readings, the next step is to combine local and hybrid methodologies to cumulate their benefits for transient flow simulations [26, 27]. In particular, it would be crucial to examine whether local approaches can effectively accelerate the various iterative methods employed for solving the Poisson equation.

To address this research gap, this study presents a hybrid methodology combining machine learning and CFD to accelerate traditional solvers for long-term incompressible fluid flow simulations. The framework is designed to be versatile, scalable, accurate, robust, and data-efficient. We used two-dimensional (2D) buoyant plume simulations to evaluate this strategy. Specifically, we implemented a local approach to predict the pressure field. A fully connected neural network was trained offline to map physics-based cell-level features to the evolution of the cell pressure between consecutive time steps. Then, the predicted field was utilized as an initial guess to solve the Poisson equation. We compared the performance of the hybrid approach using several state-of-the-art iterative solvers. The evaluation was performed on various geometries and initial conditions without additional training.

This paper is structured as follows. Section 2 presents the buoyant plume problem and the generic numerical method to approximate the solution. Then, the hybrid solver methodology, combining machine learning and CFD, is described in section 3. Next, the numerical experiment designed to assess the feasibility of our strategy is detailed in section 4. The test case, the numerical setup, and the model training procedure are explained. Finally, the model performance and the acceleration provided by the hybrid method are discussed in section 5.

2 Buoyant plume numerical modeling

This work focuses on the numerical simulation of a buoyant plume. Buoyancy effects occur due to density variations within a fluid under the influence of gravity, causing a lighter fluid to rise when surrounded by a denser fluid. Since temperature directly affects density, buoyancy drives numerous natural flows,

such as those found in oceans and the atmosphere. A plume is generated when a fluid ascends, driven by buoyancy or momentum. Plume flows are a subject of extensive study due to their complex behaviors, presenting challenges for simulation [28, 29].

2.1 Simple plume equations

We start with the assumption of a single-phase, incompressible Newtonian fluid. The non-isothermal Navier-Stokes equations, with the Boussinesq approximation, are used to model a simple plume. The momentum (1), continuity (2), and energy (3) equations are written as follows in Cartesian coordinates:

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} = -\dfrac{1}{\rho_0}\nabla p + \nu\nabla^2\boldsymbol{u} + \dfrac{\rho}{\rho_0}\boldsymbol{g} & (1) \\ \nabla\cdot\boldsymbol{u} = 0 & (2) \\ \dfrac{\partial T}{\partial t} + (\boldsymbol{u}\cdot\nabla)T = \alpha\nabla^2 T & (3) \end{cases}$$

where $\boldsymbol{u}$, p, and T are respectively the fluid velocity, pressure, and temperature fields. ρ represents the variable fluid density, and ρ_0 is the constant reference density. ν denotes the kinematic viscosity, α the thermal diffusivity, and $\boldsymbol{g}$ the gravitational field.

The energy equation (3) represents the standard unsteady advection-diffusion heat transfer process without a heat source. The incompressibility assumption holds under the condition that (i) the velocities involved are low (i.e., subsonic flow) and that (ii) density variations remain moderate. The latter point refers to the Boussinesq approximation, where density variations are considered only in the gravitational term of the momentum equation. In addition, a linear dependence is maintained between density variation and temperature variation (Equation 4).

$$\rho_0 - \rho = \beta\rho_0(T - T_0) \tag{4}$$

β denotes the coefficient of thermal expansion, T_0 is the constant reference temperature related to ρ_0, and T denotes the temperature related to ρ. Thus, ρ can be substituted in Equation (1).

If the temperature differences are small enough, this modeling correctly captures the movements due to the buoyancy. Thus, the system of governing equations (1)-(2)-(3)-(4), along with initial and boundary conditions, provide us with a reasonable approximation for representing a simple plume flow. The Boussinesq approximation can even extend to hot smoke control studies for small-scale fires [30]. It should be noted, however, that this approximation may yield qualitatively wrong flows when temperature differences are significant [1].

2.2 Numerical procedure

The buoyant plume problem is solved numerically using a traditional CFD procedure based on the finite volume method [1, 2]. The finite volume method is well suited for complex geometries and is inherently conservative, making it a popular choice in engineering. First, the grid generation preprocessing step divides the computational domain into a finite number of contiguous cells (control volumes). Then, the underlying equations are integrated over each cell. Next, the integral forms of the conservation equations are discretized into a system of algebraic equations. Finally, a numerical procedure solves the algebraic equations (see, e.g., Figure 1).

The governing equations constitute an interdependent pressure-velocity system, solved using the Pressure Implicit with Splitting of Operators (PISO) algorithm [31, 32]. The equations (1)-(2) are decoupled and solved sequentially for each time step. Figure 1 presents a simplified flowchart of the PISO algorithm variant we utilized in this study. t_n denotes the current time step, corresponding to the time index n. In this paper, the quantities designated by a superscript $n-1$ represent those from the preceding time step. For the sake of simplicity, the absence of a time index denotes the current time step. At time t_n, the principal PISO stages are as follows:

(I) *Predictor step*
The discretized momentum equation is solved using an estimate of the pressure field denoted p^*. This calculation yields the intermediate velocity field $\boldsymbol{u}^*$, which does not satisfy the continuity equation (unless p^* is correct). Usually, the previous time step field is employed as an initial guess for pressure: $p^* = p^{n-1}$.

(II) The temperature field is computed by solving the energy equation.

(III) *Corrector step 1*
One can derive the pressure equation by applying the divergence operator to the rearranged discretized momentum equation and using the incompressibility hypothesis. This pressure-correction Equation is a Poisson equation that is particularly computationally expensive to solve. The pressure p^* is corrected, followed by the velocity field $\boldsymbol{u}^*$. The latter is updated explicitly, ensuring the velocity adheres to the continuity equation. However, the momentum balance may not be satisfied after this step. Thus, PISO performs a second corrector step.

(IV) *Corrector step 2*
The Poisson equation is assembled following the same procedure with the updated fields. Therefore, pressure and velocity are corrected again. After this second corrector, pressure and velocity are considered to be correct.

(V) The additional discretized transport equations, such as species, temperature, or turbulence quantities, are solved.

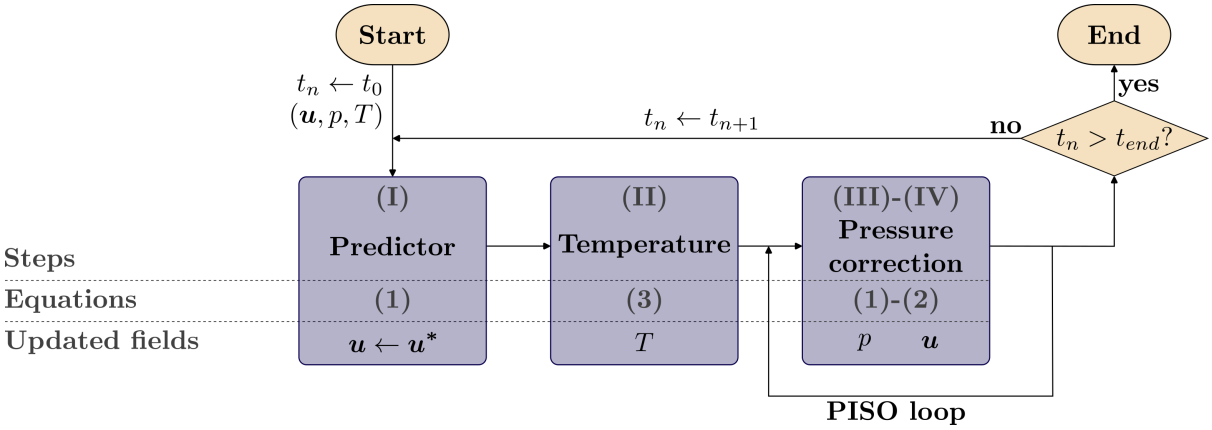


Figure 1: Simplified flowchart of the Pressure Implicit with Splitting of Operators (PISO) algorithm variant employed to address the buoyant plume case. We considered an incompressible flow with a fixed orthogonal mesh and without turbulence modeling.

2.3 Iterative solvers

Each stage of the numerical procedure described above entails solving large-scale sparse linear systems whose size depends on the number of cells in the domain. Solving the Poisson equation for pressure correction can consume up to 80% of the computational time required to simulate incompressible fluid flow [33]. Consequently, it is a significant challenge for CFD to accelerate linear solvers. Standard direct methods, including Gauss elimination and LU decomposition, are unsuitable for solving large-scale CFD problems due to their excessive computational cost. Therefore, CFD traditionally relies on iterative methods, which are generally much more efficient for solving such systems [1, 2].

Let us denote the discretized Poisson equation in matrix notation: $\mathbf{Ap} = \mathbf{b}$. $\mathbf{A}$ is the sparse coefficient matrix, $\mathbf{b}$ is the right-hand side vector, and $\mathbf{p}$ is the unknown vector containing the pressure values for each cell. Iterative methods start with an initial guess $\mathbf{p}^{(0)}$, which is improved iteratively through the repeated application of a cost-effective procedure. $\mathbf{A}$ is decomposed, and the system is rearranged to define an iterative procedure $\mathbf{Mp}^{(k+1)} = \mathbf{Np}^{(k)} + \mathbf{c}$, with k as the iteration index, $\mathbf{M}^{-1}\mathbf{N}$ as the iteration matrix, and $\mathbf{c}$ as a constant vector. Each iterative method employs a distinct decomposition strategy to achieve low-cost iterations and fast convergence. These linear solvers possess the advantage of memory efficiency and can be stopped when the desired tolerance is reached, which is convenient for adjusting the speed-accuracy tradeoff. As the error $\mathbf{e}^{(k)} = \mathbf{p} - \mathbf{p}^{(k)}$ is unknown during the solving process, the residual $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{Ap}^{(k)} = \mathbf{Ae}^{(k)}$ is used to define the stopping criterion. The number of iterations required to converge toward an acceptable level of accuracy cannot be predicted in advance. However, it is closely related to the spectral radius of the iteration matrix and the initial guess.

Point-iterative techniques, such as Jacobi, Gauss-Seidel, or successive over-relaxation (SOR), are easy to implement but tend to exhibit a slow convergence rate for large systems. In particular, these methods encounter difficulties in eliminating low-frequency errors, which results in poor convergence performance on fine meshes. Multigrid acceleration techniques have been developed to address this convergence issue. Multigrid approaches use several resolution levels within the iterative process, following a cycle of coarsening and refinement. Point-iterative methods, called smoothers in this context, are efficient in eliminating high-frequency errors on a coarse grid, which correspond to the low-frequency errors of the finer grid. Thus, multigrid methods exhibit accelerated convergence and represent state-of-the-art techniques for solving linear systems in CFD. Krylov subspace methods, such as the Generalized Minimum Residual (GMRES) or the conjugate gradient method, are also popular in CFD. Concerning conjugate gradient, the quadratic form of the linear system is used to convert it into a minimization problem. Consequently, gradient-based minimization methods, such as the conjugate gradient method, can be executed. In practice, the problem is modified with preconditioners to improve convergence properties. In the case of non-symmetric systems, the biconjugate gradients variant adapts the methodology.

This study aims to accelerate iterative solvers for the Poisson equation by providing a better initial guess $\mathbf{p}^{(0)}$. Although we do not provide any mathematical guarantee that an initial guess closer to the solution will significantly speed up the convergence, its impact can be critical.

3 Hybrid solver methodology

This section introduces the hybrid CFD/machine-learning methodology designed to accelerate transient simulations without compromising accuracy. The overall approach involves training a data-driven model to predict the future pressure field and then incorporating the prediction into the pressure-velocity coupling algorithm to expedite the pressure correction step.

3.1 Prediction workflow

The initial objective is establishing a workflow that estimates the pressure field $p^n := p(\mathbf{x}, t_n)$ for each simulation time step t_n, with $\mathbf{x}$ as the position vector. We aim to link the current state of the simulation, denoted as S^n, to the pressure field p^n. S^n encompasses all the available information at t_n before the pressure correction step, such as the simulated quantities $\boldsymbol{u}(\mathbf{x}, t_{n-1})$, $p(\mathbf{x}, t_{n-1})$, or $T(\mathbf{x}, t_{n-1})$. Note that S^n may also include information from the previous time steps $\{t_0, \ldots, t_{n-1}\}$, given that we have access to the complete time history of a simulation. Figure 2 depicts the prediction workflow for generating the pressure field. The prediction strategy is founded upon three fundamental pillars.

(i) *Cell-level predictions.* The approach involves making predictions at the cell level rather than attempting to produce an estimate for the entire domain at once. For all cells $i \in \{1, \ldots, N_{cell}\}$ of the domain, the unique model maps a local state S_i to a local pressure p_i. The local strategy is designed to scale to large domains without increasing model complexity. Additionally, the same model can make predictions for variable geometries. We also expect to enhance model generalizability by learning a local operator [17] and to train models with a small amount of data [19], i.e., data-efficient learning. While some studies coupling machine learning and CFD preferred a multi-scale approach for accurate predictions [22, 24], others have shown that a local approach can achieve reasonable accuracy [19, 20]. Thus, we hypothesize that cell-level predictions can be sufficiently accurate for our hybrid solver.

(ii) *Pressure correction learning.* Instead of predicting the absolute pressure, the model learns to correct the pressure from the previous time step. In other words, the model output is the local pressure temporal variation $p_i^n - p_i^{n-1}$. It has been proven that learning a correction can significantly improve model performance [16, 18].

(iii) *Physics-based features.* The model inputs are handcrafted predictors based on physical quantities. For example, local features such as the divergence or the gradient of a quantity can be computed from the local state S_i. It should be noted that S_i provides information about cell i and its neighborhood, the size of which can be flexible. A few carefully chosen features may be adequate for accurate predictions. This strategy could lead to simple models, offering improved interpretability and faster inference. We posit that engineered features are worth considering before transitioning to more complex architectures, such as those typically utilized in deep learning.

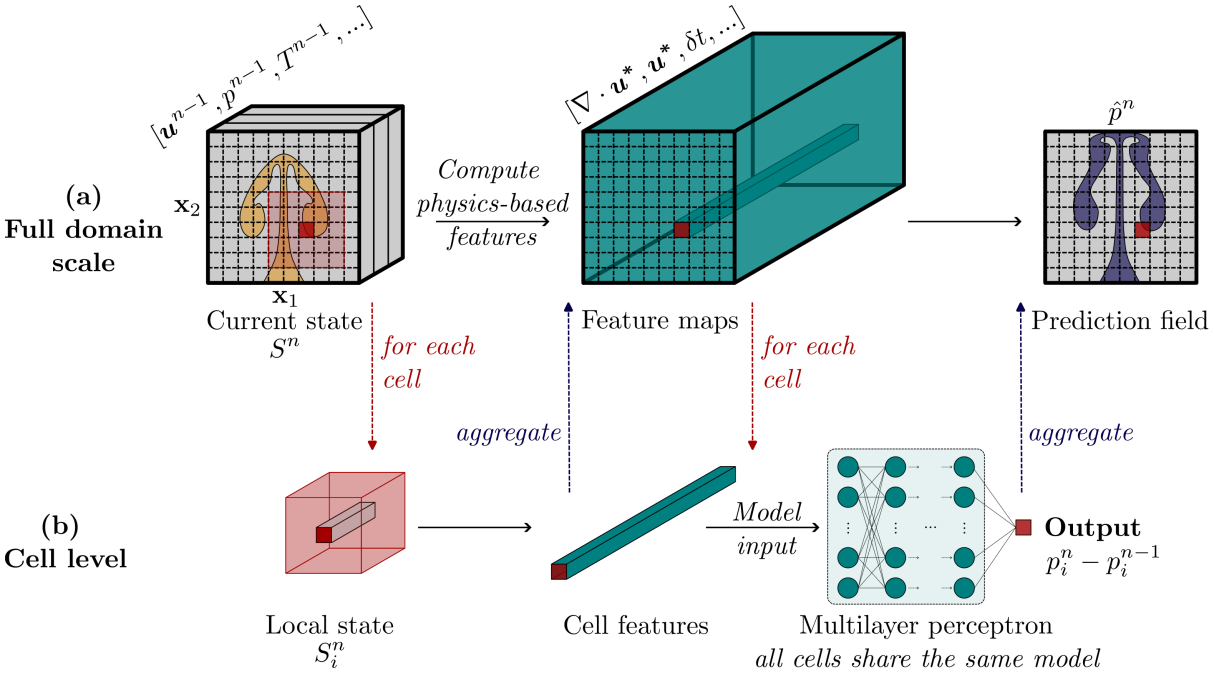


Figure 2: Illustration of the prediction workflow. At a given time step t_n, we aim to predict the pressure field p^n. **(a)** Local physics-based features are computed for the entire domain, enabling the model to estimate the pressure field. **(b)** Features are calculated similarly for each cell i of the domain. Some features incorporate information from the cell's neighborhood. The model infers the cell pressure variation from one time step to the next.

3.2 Model learning strategy

The data-driven model is trained using a supervised learning approach, illustrated in Figure 3. The training database contains CFD simulation examples. For each simulation time step and domain cell, input/output couples $(\boldsymbol{x}, \boldsymbol{y})$ can be stored in matrices (X_{train}, Y_{train}). X_{train} is a $N_{train} \times N_{input}$ matrix, and Y_{train} a $N_{train} \times N_{output}$ matrix. N_{train} denotes the number of points in the training set, N_{input} is the input size, and N_{output} is the output size. Then, the model is trained to approximate the function $f : \boldsymbol{x} \mapsto \boldsymbol{y}$. In this study, the input vector $\boldsymbol{x}$ corresponds to the physics-based features computed from the local state S_i, and the scalar output y corresponds to the cell pressure correction $p_i^n - p_i^{n-1}$. Machine learning algorithms, such as linear regressions, decision trees, or neural networks, are designed to learn the best possible approximation $f_\theta : \boldsymbol{x} \mapsto \hat{\boldsymbol{y}}$ of f from data. The parameters θ are automatically adjusted to minimize a loss function $\mathcal{L}$, which quantifies the discrepancy between the model predictions $\hat{\boldsymbol{y}}$ and the ground truth $\boldsymbol{y}$.

The objective is to develop a model that can be trained offline and provide satisfactory interpolation capabilities. Despite the time-consuming phases of data generation and training, the model remains advantageous as it can be deployed in a multitude of novel scenarios. We believe it is preferable to specialize the model in a particular domain to achieve the highest possible prediction performance. Indeed, developing a local machine-learning model for general-purpose CFD acceleration is a highly ambitious undertaking. In our case, we concentrated on buoyant plume simulations, but this versatile hybrid approach could also be effectively applied to other domains.

3.3 Model coupling approach

In addressing transient simulations, it is anticipated that one-time-step machine learning predictions may not be sufficient to expedite long-term simulations. Although a data-driven model may demonstrate strong predictive capabilities, it is likely to accumulate errors over successive time steps, which could result in a significant divergence from the physical flow [18, 19, 20, 21]. To guarantee long-term consistency, it becomes imperative to integrate data-driven outcomes within a traditional CFD solver.

Given that the pressure correction steps (III)-(IV) represent the most time-intensive phase for incompressible flows, efforts are directed toward enhancing the efficiency of this component. Hence, we focus

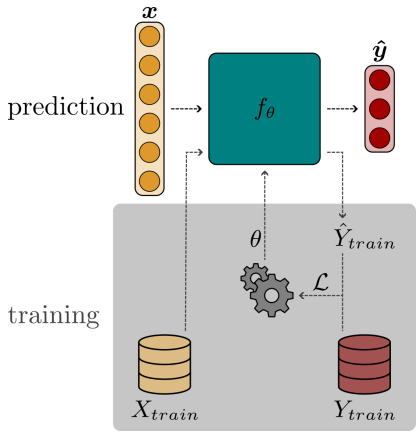


Figure 3: Supervised learning diagram. The machine learning model f_θ provides a prediction $\hat{\boldsymbol{y}} \approx \boldsymbol{y}$ from an input vector $\boldsymbol{x}$. A database (X_{train}, Y_{train}) is used to train the model. The training process aims to find optimal parameters θ, according to some metric $\mathcal{L}$, to approximate the function f mapping inputs $\boldsymbol{x}$ to outputs $\boldsymbol{y}$.

on pressure field predictions to build a Poisson equation solver surrogate model. Instead of substituting the Poisson equation iterative solver, we follow the idea of using model predictions as an initial guess, which has demonstrated encouraging results [22, 24, 25, 26, 27].

In the PISO procedure (Figure 1), the coupling strategy involves running the prediction workflow (Figure 2) for each time step before the pressure correction (III). The approach implies additional calculations for each time step, which must be considered when comparing the overall computing time. Nevertheless, these extra computations are intended to be rapid, independent, and scalable to any domain size.

According to the specified coupling strategy, it could be argued that it is preferable to use the intermediate pressure correction (after the first corrector) as the model output rather than the final pressure (after the second corrector). Nonetheless, the estimated outcome remains valid as the first corrector converges closely to the final pressure. Additionally, this opens the door for the supplementary integration of a machine learning-based velocity correction, which would enhance the initial pressure-velocity coupling before pressure correction.

Thus, this coupling strategy guarantees the same level of accuracy as traditional CFD methods and is specifically designed to reduce the number of iterations for pressure correction, thereby reducing the overall computational time required.

4 Experiment setup

A numerical experiment was conducted to assess the potential of the methodology presented in section 3. The implementation of this innovative hybrid solver introduces, to the best of our knowledge, novel elements to the existing literature, including the following points:

(i) custom local physics-based features for data-driven pressure field estimation are evaluated,

(ii) cell-level machine learning predictions are used as initial guesses for iterative linear solvers,

(iii) the methodology is applied to a non-isothermal case with variable domain geometries and initial conditions without model retraining,

(iv) the hybrid strategy performance is compared for several state-of-the-art iterative solvers.

4.1 Reference case

A reference case was defined and then adapted with diverse geometries and initial conditions to generate a training database. Inspired by similar studies [22, 34], the reference case was a 2D buoyant air plume in a $L_{\mathbf{x}_1} \times L_{\mathbf{x}_2}$ rectangular enclosure, as illustrated in Figure 4. We simulated a one-square-meter box, closed with adiabatic walls and filled with air at a reference temperature of T_0. A unique inlet was positioned

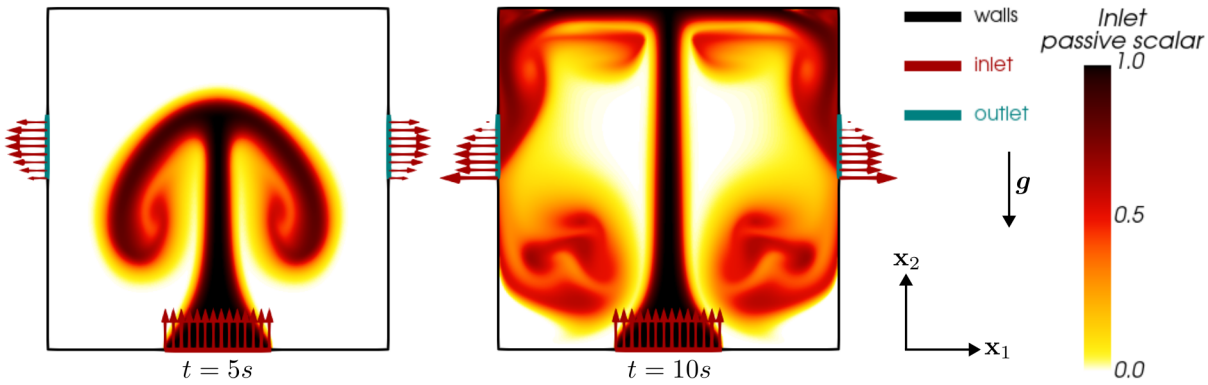


Figure 4: Reference case simulation at two different time steps. Inside the domain, colors represent the concentration of a passive scalar introduced at the inflow. The orthogonal projections of the velocity vectors at the boundaries are displayed to visualize the airflow rates. For this reference simulation, we set $L_{\mathbf{x}_1} = L_{\mathbf{x}_2} = 1\,\mathrm{m}$, $T_{inlet} = T_0 + 10$, and $U_{inlet} = 0.125\,\mathrm{m.\,s^{-1}}$.

at the bottom of the box, through which warm air at a temperature T_{inlet} was blown at a fixed uniform vertical velocity U_{inlet}. Two outlets were located on the vertical walls. The airflow could be directed either inwards or outwards at these openings according to the prevailing pressure conditions. The reverse flow temperature was fixed to T_0. The simulation was conducted for 10 seconds, which allowed sufficient time for the plume to reach the ceiling due to the buoyancy force and for the emergence of various flow patterns (see Figure 4). To mimic air properties, we set $T_0 = 293.15\,\mathrm{K}$, $\nu = 15.06 \times 10^{-6}\,\mathrm{m^2/s}$, $\beta = 3.43 \times 10^{-3}\,\mathrm{K^{-1}}$, the Prandtl number to 0.7 and $g = 9.81\,\mathrm{m/s^2}$.

4.2 Physics solver setup

We aimed to establish a robust physics solver configuration capable of solving 2D plume cases under various conditions with reasonable accuracy. This CFD solver was used to produce the training database, develop the hybrid solver, and serve as a benchmark for evaluating the hybrid approach. We were not striving to produce precise Direct Numerical Simulation (DNS) results, as it is unnecessary to demonstrate the feasibility of our methodology.

The open-source CFD software `OpenFOAM` (`v2312`) was employed to simulate the transient flows. Specifically, we used the pressure-based solver `buoyantBoussinesqPimpleFoam`. The solver was configured in PISO mode following the flowchart depicted in Figure 1. A uniform Cartesian grid comprising 256 cells per meter was defined for spatial discretization. Although the methodology employed did not impose such constraints on the mesh, this grid provided a simple starting point. The boundary conditions employed are presented in Table 1. Simulations were run without turbulence modeling, which might introduce numerical errors. The use of stable numerical schemes was favored. Therefore, a first-order backward Euler time integration was applied in conjunction with a first-order upwind scheme for the convective terms. The other spatial schemes relied on the default linear interpolation. The temporal evolution of the flow was simulated using a variable time step. Indeed, δt was adjusted according to the maximum Courant number, set to 0.9. The linear systems were solved iteratively (see Section 2.3) with a tolerance of 10^{-7} on the normalized residuals. The multigrid solver `GAMG` with the Gauss-Seidel smoother was employed for the pressure, while conjugate gradient solvers were utilized for all other quantities.

Table 1: Boundary conditions employed in `OpenFOAM` to solve the buoyant plume cases.

	Velocity	Pressure	Temperature
Wall	`noSlip`	`fixedFluxPressure`	`zeroGradient`
Inlet	`fixedValue`	`fixedFluxPressure`	`fixedValue`
Outlet	`pressureNormalInletOutletVelocity`	`prghTotalPressure`*	`inletOutlet`

* The total pressure was fixed to $\boldsymbol{g} \cdot \mathbf{h}$

It is worth mentioning that, in `OpenFOAM`, the momentum equation is rearranged to include the hydrostatic pressure contribution $\rho(\boldsymbol{g} \cdot \mathbf{h})$ into the pressure gradient, which is numerically convenient.

$\mathbf{h}$ denotes the height vector in the opposite direction to gravity (our reference height is 0). Thus, by defining the alternative pressure $p_{rgh} = \frac{1}{\rho_0}(p - \rho(\boldsymbol{g} \cdot \mathbf{h}))$, Equation 1 becomes:

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = -\nabla p_{rgh} + \nu \nabla^2 \boldsymbol{u} - (\boldsymbol{g} \cdot \mathbf{h})\nabla(\frac{\rho}{\rho_0}) \tag{5}$$

Indeed, based on the relationship

$$\nabla(\frac{\rho}{\rho_0}\boldsymbol{g} \cdot \mathbf{h}) = \boldsymbol{g} \cdot \mathbf{h}\nabla(\frac{\rho}{\rho_0}) + \frac{\rho}{\rho_0}\underbrace{\nabla(\boldsymbol{g} \cdot \mathbf{h})}_{=\boldsymbol{g}}$$

the gravity term can be substituted in Equation 1, leading to Equation 5. Consequently, the target pressure in this experiment was not p, but p_{rgh}. Nevertheless, for the sake of readability, we continue using the term pressure throughout this document.

4.3 Local features

The machine learning model utilized physics-based local features as input to predict pressure evolution (see Figure 2). These predictors were designed to be computationally inexpensive and to provide the model with insightful information for accurate predictions. We deliberately opted for handcrafted features to keep the machine-learning process simple while providing baseline results. However, identifying relevant features represents a significant challenge, contributing to the popularity of deep learning algorithms. In light of this consideration, it is prudent to anticipate that representation learning algorithms have the potential to enhance our workflow by automatically extracting relevant features.

Table 2: List of cell-level features used as model input at time t_n.

Feature	Description
$\boldsymbol{u^*}$	Intermediate velocity. Computed at the predictor step (I).
T^*	Intermediate temperature. Computed with the intermediate velocity $\boldsymbol{u^*}$ at step (II).
p^*	Intermediate pressure. The pressure estimate used for the predictor step (I) and as initial guess $\mathbf{p}^{(0)}$ for step (III). In this experiment $p^* = p_{rgh}^{n-1}$.
δt	Time step between time t_{n-1} and t_n.
d_{inlet}	Distance from cell to inlet.
d_{outlet}	Distance from cell to closest outlet.
d_{wall}	Distance from cell to closest wall.
$\delta\tilde{p}$	Filtered pressure. Difference between cell pressure and mean adjacent cell pressure. Here, $\delta\tilde{p} = p_{rgh}^{n-1} - \tilde{p}_{rgh}^{n-1}$ with $\tilde{p}$ the mean pressure on cell faces.
$\nabla \cdot \boldsymbol{u^*}$	Intermediate velocity divergence.
$\nabla \cdot (T^*\boldsymbol{u^*})$	$T^*\boldsymbol{u^*}$ product divergence field.
$\nabla \cdot (p^*\boldsymbol{u^*})$	$p^*\boldsymbol{u^*}$ product divergence field.
∇T^*	Intermediate temperature gradient.
$\nabla \cdot (\nabla T^*)$	Divergence of the temperature gradient.

See Figure 1 and Section 2.2 for PISO algorithm step numbering.

Table 2 outlines the features selected for this experiment. It is worth noting that all features were grid-independent, which preserved the flexibility of the methodology. Most features depended only on the cell itself and the adjacent cells. The distances to the boundaries, although requiring a more global perspective, were included because they might strongly impact the local fluid behavior. Considering the model target and the use of an adjustable time step, δt emerged as an essential global feature.

4.4 Dataset generation

The training data were generated by running a series of 49 simulations, all of which were variants of the reference case described in Section 4.1. The mesh size was maintained at 256 cells per meter, and each simulation was 10 s long.

The following parameters were randomly assigned according to a uniform distribution.

- Domain size $(L_{\mathbf{x}_1}, L_{\mathbf{x}_2}) \in [1, 2]^2$ m
- Inlet temperature $T_{inlet} \in [T_0 + 5, T_0 + 15]$
- Inlet velocity $U_{inlet} \in [0.1, 0.15]$ m. s^{-1}
- Inlet position and size.
- Outlet number (a maximum of one per boundary), positions, and sizes.
- Number of obstacles in the domain (up to three), positions, and sizes.

In addition, other rules were observed to ensure reasonable scenarios. Obstacles, inlets, and outlets had minimum and maximum lengths, depending on the domain size. We also prevented the positioning of elements that were too close without overlapping. Figure 5 shows an extract of the generated database.

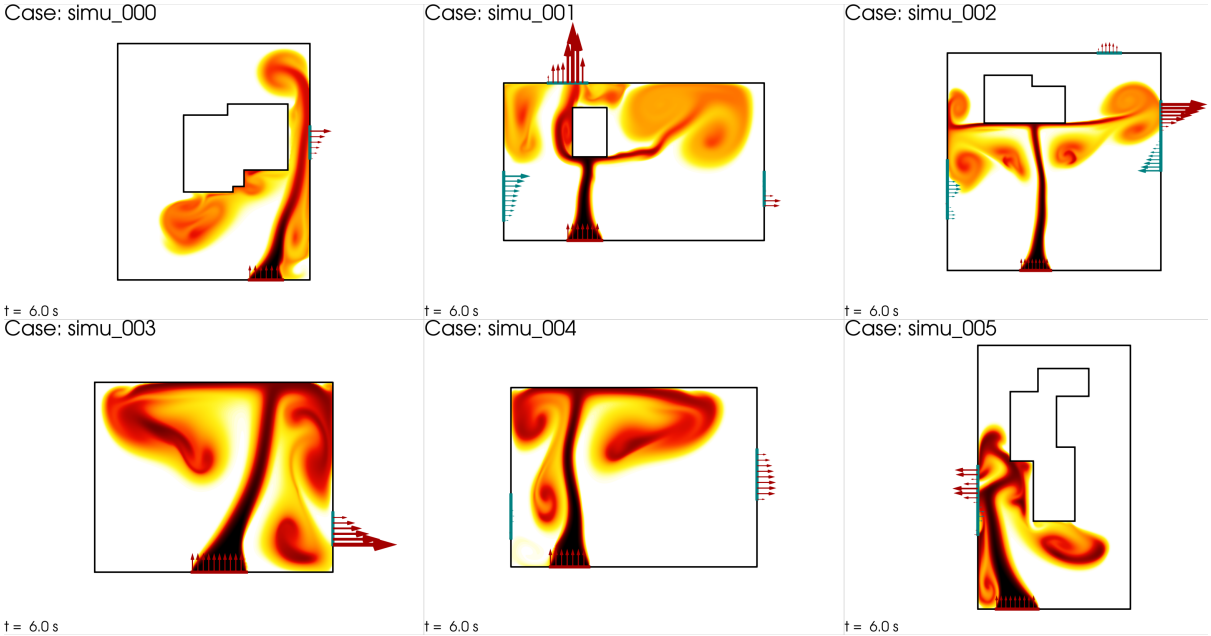


Figure 5: Database extract. Visualization of 6 simulations at the same time step $t = 6$ s. The domain size, initial conditions, and obstacles were randomly sampled. The legend (colors, vectors) is the same as in Figure 4.

The data were only gathered after 1 s of simulation, which was arbitrarily chosen. The rationale behind this was to avoid the initial phase of the simulation when the flow might exhibit some instabilities. Consequently, using the model only after 1 s of simulation in the evaluation phase was also preferable. It should be noted that this measure was introduced to facilitate model learning, although it might be unnecessary.

The cell-level procedure yields a large amount of data, as each domain cell and time step can produce a data point $(\boldsymbol{x}, \boldsymbol{y})$. In our reference simulation, 65,536 cells could generate over 90 million data points within a 10 s simulation. Thus, data points were sampled before aggregation in the database to manage a reasonable data volume while preserving a broad range of scenarios. At a given time step, we set a 5% probability of data generation. Data from 0.5% randomly selected domain cells were collected during these time steps. Our database was segmented into three subsets, following established machine learning principles. The test set comprised 7 complete simulations, while data from the remaining 42 simulations were shuffled and split. 80% of the data was designated as the training set, with the other 20% assigned to the validation set.

4.5 Model training

The model was trained offline based on the generated training set described in Section 4.4. Before training, standard preprocessing steps were carried out. First, the output pressure was modified by subtracting the pressure from the previous time step to learn a correction (see Section 3.1). Then, the inputs and outputs were scaled, based on the training data, by removing the mean and scaling to unit variance (Equation 6).

$$\mathcal{T}(x) = \frac{x - \mu_{train}}{\sigma_{train}} \tag{6}$$

μ_{train} and σ_{train} represent, for the variable of interest x, the mean and the standard deviation of the training samples. We denote the inverse transformation $\mathcal{T}^{-1}$ such that $\mathcal{T}^{-1}(\mathcal{T}(x)) = x$.

Although the presented framework can be applied with any machine learning algorithm, this experiment focused on neural networks. Indeed, neural networks have proven to be excellent universal approximators capable of handling large amounts of high-dimensional data. Moreover, using such models in fluid mechanics has led to promising results, including accurate local predictions [19, 20]. Finally, neural networks provide interesting extensions for scientific machine learning applications (e.g., physics-informed neural networks, graph neural networks, or neural operators).

A neural network comprises artificial neurons that linearly combine input components and apply a final nonlinear activation function to produce a scalar output. Subsequently, the neurons are organized into layers. In this study, we chose the multi-layer perceptron (MLP) basic architecture as a starting point. We anticipated that cell-level predictions do not necessitate a complex deep-learning architecture. MLP layers are fully connected, as illustrated in Figure 2. The optimization of the network involves adjusting the weights and biases of each unit. These parameters, denoted by θ, are optimized by minimizing a loss function with gradient-based optimization techniques.

The open-source machine-learning library `scikit-learn` [35] was employed to train the model, with the following implementation details. The MLP consisted of three hidden layers with 64 units each ($64 \times 64 \times 64$). It should be noted that the model architecture is typically fine-tuned as it can significantly impact performance. However, the objective of this investigation was not to achieve optimal accuracy but rather to illustrate the viability of our methodology. Therefore, this simple architecture was selected, which aligns with the optimal configuration identified in a comparable cell-level learning study [18]. The activation function employed was the rectified linear unit (ReLU). The common squared error loss function described in Equation (7) was minimized with the Adam optimizer [36].

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = ||\boldsymbol{y} - \hat{\boldsymbol{y}}||_2^2 \tag{7}$$

Other hyper-parameters were tuned with a grid search approach. Thus, the tolerance for the optimization was set to 10^{-6}, the learning rate to 5×10^{-4}, and the batch size to 1024. An L_2 regularization term with a coefficient of 10^{-4} was incorporated into the Equation 7 loss function. The maximum number of iterations (i.e., epochs) was set to 150. Based on the validation set, an early-stopping strategy was implemented to prevent overfitting.

4.6 Model incorporation in physics solver

We explored the feasibility of incorporating a machine learning model into a state-of-the-art CFD solver for research and industrial applications. Thus, the model was embedded into the `OpenFOAM` physics solver, which is implemented in C++. In its current prototype stage, our hybrid solver performed model inference in Python, with data exchanges occurring through an inefficient file read/write process. While a direct comparison of overall computation time would require a complete C++ implementation, we could still evaluate the model's impact on pressure correction. To reduce the evaluation computational cost and assess the model's integration in varying scenarios, we used the model predictions at specific time intervals during the test simulations. Specifically, the model was activated every 10 time steps, for a total of $1,045$ time steps across the 7 test simulations.

The effectiveness of the hybrid solver was assessed using the same time steps with three iterative solvers, encompassing the main families outlined in Section 2.3.

(i) `symGaussSeidel`. Symmetric Gauss-Seidel implementation, performing forward and reverse sweeps for each iteration.

(ii) `PCG-DIC`. Preconditioned conjugate gradient, with simplified Diagonal-based Incomplete Cholesky preconditioner.

(iii) `GAMG-GS`. Multigrid solver using a V-cycle, with Gauss-Seidel smoother.

The same absolute tolerance was employed to solve the Poisson equation to ensure a fair comparison between the hybrid solver and the physics solver. Convergence was reached when the residuals were below the 10^{-6} threshold, i.e., $||\mathbf{r}^{(k)}||_1 = ||\mathbf{b} - \mathbf{A}\mathbf{p}^{(k)}||_1 < 10^{-6}$. It should be noted that the residual normalization procedure was deactivated to guarantee the same level of convergence, as this factor depends on the initial solution $\mathbf{p}^{(0)}$.

5 Results and discussion

5.1 Single-step predictions

Since the loss function (see Equation 7) does not provide any relative performance regarding model predictions, we define the skill score SS_{MSE}, which is expressed in Equation 8.

$$SS_{MSE} = 1 - \frac{MSE(Y_{val}, \hat{Y}_{val})}{MSE(Y_{val}, Y_{ref})} \tag{8}$$

- MSE denotes the mean squared error between two vectors.
- Y_{val} vector, with a size of N_{val}, represents the validation data ground truth before dataset preprocessing. It contains the target pressure p_{rgh} for each data point.
- Y_{ref} refers to the reference estimation for pressure, corresponding to the pressure at the previous time step p_{rgh}^{n-1}.
- $\hat{Y}_{val}$ is the model prediction. The predicted pressure is given by $\hat{p}_{rgh} := p_{rgh}^{n-1} + \mathcal{T}^{-1}(\hat{y})$, for each validation point. The model raw prediction $\hat{y}$ is a scalar corresponding to the scaled alternative pressure variation for a given cell, i.e., $y = \mathcal{T}(p_{rgh} - p_{rgh}^{n-1})$.

This skill score is a quantitative measure of the performance of the model in comparison to a reference. The reference is the pressure at the previous time step, the standard initial guess for pressure correction (see Section 2.2). A positive skill score indicates that the model outperforms the reference for the selected metric (MSE). SS_{MSE} maximum value is 1, or 100%.

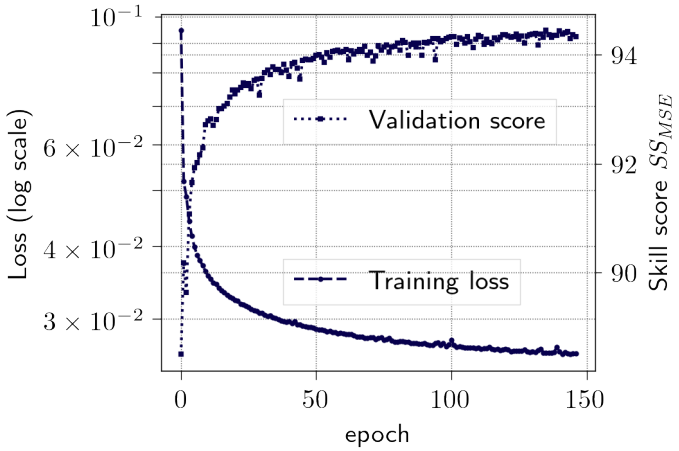


Figure 6: Model performance monitoring during the training phase. The decreasing curve (left Y-axis) shows the evolution of the loss function. The increasing curve (right Y-axis) indicates the skill score SS_{MSE} on validation data.

Figure 6 exhibits the training phase monitoring. The loss decreased from 9.48×10^{-2} at the end of the first epoch to 2.60×10^{-2} at the end of the training, indicating that the learning procedure was effective. In addition, the model demonstrated satisfactory performance on the validation set. At the end of the training procedure, SS_{MSE} reached 94.4%, demonstrating that the mean squared error on the validation data was significantly lower with the machine learning model than with the reference

estimation. Although further improvements may be achievable, the considerable enhancements provided by these predictions over the reference should be noted. The monotonic trend of the validation curve indicates that the selected model did not exhibit overfitting of the training data. Nevertheless, the training was terminated before the maximum number of epochs was reached, as the skill score did not improve further.

To provide a more comprehensive evaluation of the model's performance, we computed the skill score SS_{Max} with another discrepancy metric. The MSE in Equation 8 was replaced by the metric $Max(\boldsymbol{y}, \hat{\boldsymbol{y}}) = ||\boldsymbol{y} - \hat{\boldsymbol{y}}||_\infty$. The resulting value for SS_{Max} was 75.6%. The presented score indicates that the maximum error of the reference model was considerably larger than the maximum error of the proposed model on the validation set. As the reference model is an acceptable physics-based approximator, the value of SS_{Max} suggests that the machine learning model was stable in the sense that the predictions on unseen data did not deviate excessively from the ground truth.

At this juncture, we have demonstrated the viability of the prediction workflow and the learning strategy presented in Section 3. A single cell-level model could provide reasonable pressure field predictions for the next time step, which were more accurate than the reference when considering the 2-norm and the infinity norm. The trained model exhibited notable performance on validation cases, indicating robust interpolation capabilities across diverse scenarios.

However, we still need to quantify the benefits that these predictions may bring to the iterative solvers (this point is discussed in Section 5.3). Furthermore, it is important to acknowledge that the model was designed for single-time-step predictions and is likely inadequate for successive time steps without a hybrid strategy. In addition, this initial demonstration is limited to the specified buoyant plume use case.

Further investigation is necessary to gain a more comprehensive understanding of the performance of the surrogate model. First, it would be valuable to assess the model's extrapolation capabilities. For example, testing the model with larger domains or initial conditions beyond database distributions could yield valuable insights into its generalization performance. In addition, it is crucial to determine whether the methodology can be extended to different grid types and resolutions. Next, the learning approach could be applied to various quantities, such as the velocity, the temperature, or the pressure gradient. Finally, optimizing the hyperparameters, including the model architecture, would enhance the prediction performance.

Nevertheless, this study provides encouraging evidence of the potential for local data-driven prediction capabilities. This strategy could lead to simple machine learning architectures theoretically scalable to any domain size. The adaptability of the methodology could enable the development of domain-specific models across a wide range of fields.

5.2 Feature selection

The physics-based features presented in Section 4.3 were selected without a precise physical justification. This raises the question of which quantities are relevant and whether other features might be more appropriate. The first point was elucidated through the implementation of a feature selection algorithm.

A forward sequential feature selection algorithm was employed to determine the relative importance of inputs. The algorithm incorporates new features stepwise, selecting the one that yields the greatest improvement in the validation skill score SS_{MSE} (Equation 8). Therefore, the number of training instances to launch to rank N_{input} features equals $\sum_{k=1}^{N_{input}} k$. In our case, we identified 15 scalar inputs, which led to the training of 120 models. Given the considerable time investment required for this strategy, we set a maximum of 50 training iterations, which might allow for further improvements in some models. It should be noted that this greedy methodology does not provide the optimal combinations of features to maximize the skill score. However, it does provide valuable insights with a reasonable algorithmic complexity.

Figure 7 depicts the evolution of the skill score SS_{MSE} as features were sequentially incorporated into the input vector, following the greedy procedure. First, we observe that a limited number of features can provide reasonable accuracy. The skill score exceeded 90% with just five input features. Then, divergence fields were paramount in enhancing the model's accuracy. Indeed, it seems plausible to suggest that the intermediate velocity divergence $\nabla \cdot \boldsymbol{u}^*$ is crucial, as it helps identify the continuity error when using the uncorrected pressure p^* to solve the momentum equation. Next, the relevance of the time step and distance features was established. Finally, it seems that the raw quantities T^*, p^*, and $\boldsymbol{u}^*$ at the cell level provided little helpful information to improve the model. It should be noted that, for vectors, the second component ($\mathbf{x}_2$) proved to be more pertinent than the first ($\mathbf{x}_1$), likely due to its alignment with

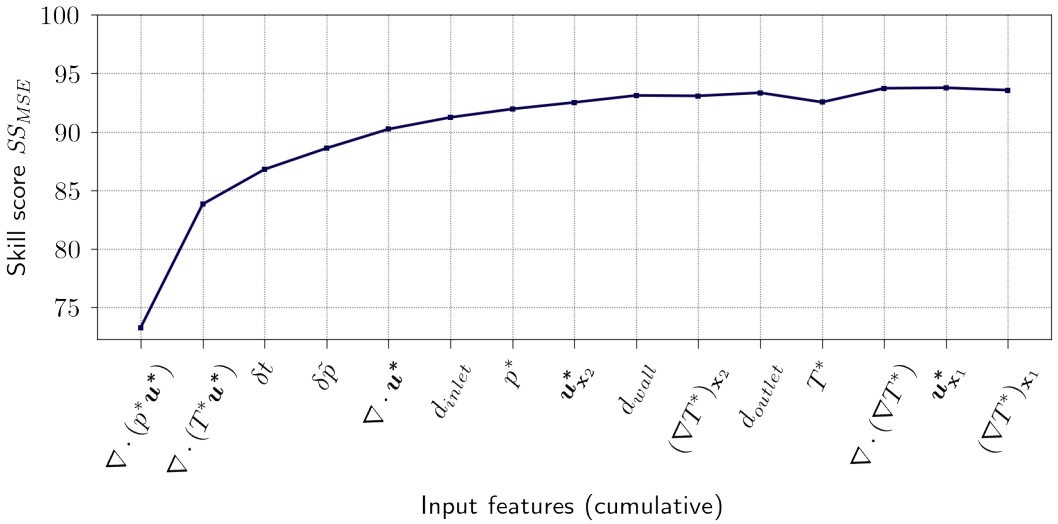


Figure 7: Forward sequential feature selection results. The skill score SS_{MSE} was evaluated for each new feature added to the input vector.

the buoyancy force. The findings may inform the development of parsimonious and fast models based on a few judiciously chosen predictors.

Further feature engineering could lead to more relevant inputs, improving the model's performance. In light of the temporal nature of the problem at hand, it would be appropriate to use time series as input so that the model may comprehend the dynamics of flow. Additionally, it may be beneficial to increase the local state S_i size, i.e., the model receptive field. Indeed, a less restrictive view of the problem could lead to more informed and accurate predictions. Finally, feature extraction techniques, including representation learning, constitute the logical next step in improving model features.

5.3 Hybrid solver evaluation

The acceleration factor $\eta(x) = \frac{x_{\text{CFD}}}{x_{\text{ML+CFD}}}$ is defined for each time step to assess the hybrid solver performance. x denotes the metric of interest:

- k_{corr1}: the number of iterations to solve the first pressure corrector (step (III), see Section 2.2).
- τ_{corr1}: the processor time measured for the first pressure corrector.
- τ_{corr}: the processor time measured for the entire pressure correction process (steps (III)-(IV), see Section 2.2).

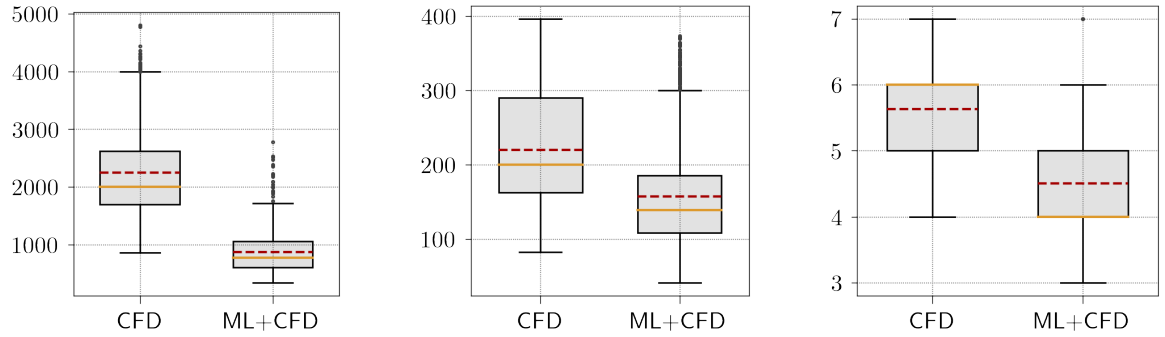
Note that all the simulations were carried out under the same conditions with the same machine. "CFD" refers to the physics solver using a traditional iterative method to solve the Poisson equation (see Section 4.6): Gauss-Seidel, preconditioned conjugate gradient, or multigrid. The term "ML+CFD" refers to the hybrid solver presented in this paper, where the machine learning model prediction is used as an initial guess to solve the Poisson equation.

Table 3 summarizes the acceleration factors obtained with the hybrid approach. All configurations were successfully accelerated on average, i.e., $\eta > 1$. As expected, the number of iterations required to solve the first Poisson equation was reduced. The mean acceleration factors were approximately equal to 3.0, 1.6, and 1.3 for Gauss-Seidel, preconditioned conjugate gradient, and multigrid solvers. In the case of Gauss-Seidel, all the evaluation time steps were strictly improved, which is a highly encouraging outcome. This proportion fell to 90% for the conjugate gradient and 80% for the multigrid solver. Therefore, the machine learning predictions offered a stable improvement to the iterative processes. The boxplots in Figure 8 represent the number of iterations required for each iterative method. Significant distribution shifts are observed, favoring the hybrid method.

Our hybrid methodology demonstrated a distinct advantage when using the Gauss-Seidel algorithm, a more moderate benefit for the preconditioned conjugate gradient techniques, and a less prominent advantage for the multigrid algorithm. Although we cannot provide a straightforward explanation for

Table 3: Acceleration factor results. Comparison between the hybrid solver and the CFD solver. k denotes the number of iterations, τ the processor time, corr1 the first pressure corrector, and corr the entire pressure correction process. Results are based on a sample of 1,045 time steps distributed across 7 test simulations.

	Iterative solver		
	Gauss-Seidel `symGaussSeidel`	Conjugate gradient `PCG-DIC`	Multigrid `GAMG-GS`
$\eta(k_{\text{corr1}})$ mean value	2.976	1.594	1.264
$\eta(k_{\text{corr1}}) > 1$ ratio	100.0%	90.4%	80.0%
$\eta(\tau_{\text{corr1}})$ mean value	2.967	1.580	1.248
$\eta(\tau_{\text{corr}})$ mean value	2.461	1.349	1.079



(a) Gauss-Seidel (b) Conjugate gradient (c) Multigrid

Figure 8: Number of iterations to solve the first pressure correction. Comparison between the hybrid solver (ML+CFD) and the CFD solver. Results are based on the same sample of 1,045 time steps distributed across 7 test simulations. Solid line: median. Dashed line: mean.

this result, we notice that the approach performed better when coupled with iterative methods exhibiting a slow convergence rate. As illustrated in Figure 8, the number of required iterations to achieve convergence varied significantly depending on the iterative method employed. Considering the baseline CFD solver, the mean number of iterations was approximately 2250 for Gauss-Seidel, 220 for the conjugate gradient, and 5.6 for the multigrid approach. Given the limited number of iterations required for the multigrid solver, larger-scale problems are worth considering. For example, an expanded domain, a three-dimensional (3D) problem, or a more refined grid might result in more favorable accelerations.

Regarding the computational time, a positive correlation was confirmed between the number of iterations k and the computing time τ. As anticipated, the acceleration factors for the global pressure correction procedure $\eta(\tau_{\text{corr}})$ were less important than those for the first corrector $\eta(\tau_{\text{corr1}})$ since the second corrector was not enhanced. This remark paves the way for further improvements, such as providing a more accurate initial estimate for the second corrector or enhancing the intermediate velocity $\boldsymbol{u}^*$ with machine learning to achieve a faster pressure-velocity coupling.

This initial demonstration of the hybrid solver methodology is highly encouraging. Our work demonstrates that cell-level machine-learning corrections of the pressure field can significantly accelerate CFD simulations while maintaining accuracy. What may appear unexpected is that the model was not fine-tuned. A simple MLP architecture was employed, with basic physics-based local quantities as inputs. In addition, simulations were run with a variable time step, which improves the approach flexibility.

A primary limitation is that the accelerations presented did not account for the time spent on model inference and feature computations, which must be performed for each cell in the domain. Consequently, the actual computational time savings are likely lower. Nevertheless, given the simple neural network architecture, the inference should be fast. Additionally, the model predictions and feature computations can be parallelized, and their associated cost is expected to increase linearly with the number of cells. Thus, the methodology should benefit problems above a certain size, provided that the acceleration factors are maintained and the iterative method employed does not scale linearly with the degrees of freedom [27]. However, this statement remains to be demonstrated for large-scale problems.

In this study, no comparison was made with respect to the tolerance set to 10^{-6} (see Section 4.6). The acceleration factors are anticipated to increase when the tolerance is reduced [24], rendering the method even more attractive. Nevertheless, it would be beneficial to corroborate this assertion within the context of our investigation.

Finally, only a single model was assessed in this experiment. It is imperative to ascertain the influence of more accurate models on the observed acceleration. The precise relationship between a reduction in quadratic error and the acceleration factors may not be immediately apparent.

Taking a step back, while we concentrated on the buoyant plume case, this coupling methodology is valuable for all incompressible simulations. Local domain-specific models could emerge for a robust pressure-velocity coupling acceleration without any compromises in accuracy.

6 Conclusion and future work

This study introduced a hybrid solver methodology that combines a traditional CFD solver with a machine learning model. This approach enables the acceleration of unsteady incompressible simulations while maintaining accuracy. A generic workflow for data-driven cell-level predictions and a coupling strategy with the PISO algorithm were described to speed up the time-consuming pressure correction step. The methodology entails the offline supervised training of a model, which can then be applied to various interpolated cases with disparate domain geometries and initial conditions.

The experimental results demonstrated the concept's applicability to non-isothermal 2D buoyant plume cases. In particular, we proved that cell-level pressure predictions can be employed as initial guesses to accelerate Poisson equation iterative solvers. The study is noteworthy for the simplicity of its neural network architecture, which relied on a limited number of handcrafted local physics-based features. Finally, we conducted a comparative analysis of the impact of data-driven predictions on computing time across three distinct categories of state-of-the-art linear iterative solvers.

In particular, a multi-layer perceptron enhanced the pressure initial guess by 94% regarding the mean squared error. It was demonstrated that a 90% enhancement could be achieved with only five predictors. Notably, the divergence fields associated with the intermediate quantities in the PISO algorithm were crucial features. Finally, the hybrid strategy facilitated a reliable acceleration of the first pressure corrector in diverse test scenarios. Observed acceleration factors ranged from 1.3 with the multigrid solver to 3.0 with the Gauss-Seidel algorithm.

Although the methodology is versatile and produced encouraging results, the scope of this study was limited to 2D buoyant plume cases with a uniform Cartesian mesh. Furthermore, the time comparisons did not include the model inference or feature computations.

Future research should concentrate on a comprehensive time comparison involving larger cases, including 3D ones, to confirm the potential and scalability of the hybrid solver. In addition, the extrapolation capabilities must be evaluated. Assessing the methodology for unstructured meshes and turbulent flows would be beneficial in addressing increasingly complex applications. Concurrently, the precision of the models can be improved to achieve greater acceleration. Several avenues can be pursued, including model optimization (e.g., algorithm, architecture, hyperparameters) or feature engineering (e.g., temporal features, increased receptive field, representation learning).

This study constitutes a step toward integrating machine learning models into CFD solvers. The hybrid methodology is versatile and could be applied to any incompressible flow. We hope this study will inspire further investigations into coupling CFD solvers with local data-driven predictions, leading to the development of reliable and scalable hybrid solutions for real-world challenges.

References

[1] Joel H. Ferziger, Milovan Perić, and Robert L. Street. *Computational Methods for Fluid Dynamics*. Springer International Publishing, Cham, 2020.

[2] Henk K. Versteeg and Weeratunge Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson/Prentice Hall, Harlow, 2. ed., [nachdr.] edition, 2007.

[3] Akshai Kumar Runchal and Madhukar M Rao. CFD of the Future: Year 2025 and Beyond. *50 Years of CFD in Engineering Sciences: A Commemorative Volume in Memory of D. Brian Spalding*, pages 779–795, 2020.

[4] N. Morozova, F. X. Trias, R. Capdevila, C. D. Pérez-Segarra, and A. Oliva. On the feasibility of affordable high-fidelity CFD simulations for indoor environment design and control. *Building and Environment*, 184:107144, October 2020.

[5] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine Learning for Fluid Mechanics. *Annual Review of Fluid Mechanics*, 52(1):477–508, January 2020.

[6] Ricardo Vinuesa and Steven L. Brunton. Enhancing computational fluid dynamics with machine learning. *Nature Computational Science*, 2(6):358–366, June 2022.

[7] Giovanni Calzolari and Wei Liu. Deep learning for CFD analysis in built environment applications: A review. *CLIMA 2022 conference*, May 2022.

[8] Navid Zehtabiyan-Rezaie, Alexandros Iosifidis, and Mahdi Abkar. Data-driven fluid mechanics of wind farms: A review. *Journal of Renewable and Sustainable Energy*, 14(3):032703, June 2022.

[9] Trevor Hastie, Jerome H Friedman, and Robert Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2 edition, 2009.

[10] Ricardo Vinuesa and Steven L. Brunton. Emerging trends in machine learning for computational fluid dynamics. *Computing in Science & Engineering*, 24(5):33–41, 2022.

[11] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep Learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[13] Salah A. Faroughi, Nikhil M. Pawar, Célio Fernandes, Maziar Raissi, Subasish Das, Nima K. Kalantari, and Seyed Kourosh Mahjour. Physics-guided, physics-informed, and physics-encoded neural networks and operators in scientific computing: Fluid and solid mechanics. *Journal of Computing and Information Science in Engineering*, 24(4):040802, January 2024.

[14] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019.

[15] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Graph Kernel Network for Partial Differential Equations. https://arxiv.org/abs/2003.03485v1, March 2020.

[16] Bjoern List, Li-Wei Chen, Kartik Bali, and Nils Thuerey. How Temporal Unrolling Supports Neural Physics Simulators, February 2024.

[17] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, May 2021.

[18] Joongoo Jeon, Juhyeong Lee, and Sung Joong Kim. Finite volume method network for the acceleration of unsteady computational fluid dynamics: Non-reacting and reacting flows. *International Journal of Energy Research*, 46(8):10770–10795, 2022.

[19] Joongoo Jeon, Juhyeong Lee, Ricardo Vinuesa, and Sung Joong Kim. Residual-based physics-informed transfer learning: A hybrid method for accelerating long-term CFD simulations via deep learning. *International Journal of Heat and Mass Transfer*, 220:124900, March 2024.

[20] João Pedro Souza de Oliveira, Joao Victor Barbosa Alves, João Neuenschwander Escosteguy Carneiro, Ricardo de Andrade Medronho, and Luiz Fernando Lopes Rodrigues Silva. Coupling a neural network technique with CFD simulations for predicting 2-D atmospheric dispersion analyzing wind and composition effects. *Journal of Loss Prevention in the Process Industries*, 80:104930, December 2022.

[21] Wenhui Peng, Shaoxiang Qin, Senwen Yang, Jianchun Wang, Xue Liu, and Liangzhu (Leon) Wang. Fourier neural operator for real-time simulation of 3D dynamic urban microclimate. *Building and Environment*, 248:111063, January 2024.

[22] Ekhi Ajuria Illarramendi, Michaël Bauerheim, and Bénédicte Cuenot. Performance and accuracy assessments of an incompressible fluid solver coupled with a deep convolutional neural network. *Data-Centric Engineering*, 3:e2, 2022.

[23] Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowliswharan. CFD-Net: A deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th ACM International Conference on Supercomputing*, pages 1–12, Barcelona Spain, June 2020. ACM.

[24] Ruilin Chen, Xiaowei Jin, and Hui Li. A machine learning based solver for pressure Poisson equations. *Theoretical and Applied Mechanics Letters*, page 100362, August 2022.

[25] Enrui Zhang, Adar Kahana, Eli Turkel, Rishikesh Ranade, Jay Pathak, and George Em Karniadakis. A Hybrid Iterative Numerical Transferable Solver (HINTS) for PDEs Based on Deep Operator Network and Relaxation Methods, August 2022.

[26] Paulo Sousa, Alexandre Afonso, and Carlos Veiga Rodrigues. Application of machine learning to model the pressure poisson equation for fluid flow on generic geometries. *Neural Computing and Applications*, May 2024.

[27] Paulo Sousa, Carlos Veiga Rodrigues, and Alexandre Afonso. Enhancing CFD solver with Machine Learning techniques. *Computer Methods in Applied Mechanics and Engineering*, 429:117133, September 2024.

[28] G. Maragkos, P. Rauwoens, Y. Wang, and B. Merci. Large Eddy Simulations of the Flow in the Near-Field Region of a Turbulent Buoyant Helium Plume. *Flow, Turbulence and Combustion*, 90(3):511–543, April 2013.

[29] A. Kondrashov, I. Sboev, and P. Dunaev. Heater shape effects on thermal plume formation. *International Journal of Thermal Sciences*, 122:85–91, December 2017.

[30] G. G. Rooney and P. F. Linden. Strongly buoyant plume similarity and 'small-fire' ventilation. *Fire Safety Journal*, 29(4):235–258, November 1997.

[31] R.I Issa. Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of Computational Physics*, 62(1):40–65, 1986.

[32] Raad I. Issa Paulo J. Oliveira. An improved piso algorithm for the computation of buoyancy-driven flows. *Numerical Heat Transfer, Part B: Fundamentals*, 40(6):473–493, 2001.

[33] Ekhi Ajuria Illarramendi, Antonio Alguacil, Michaël Bauerheim, Antony Misdariis, Benedicte Cuenot, and Emmanuel Benazera. Towards an hybrid computational strategy based on Deep Learning for incompressible flows. In *AIAA AVIATION 2020 FORUM*, VIRTUAL EVENT, June 2020. American Institute of Aeronautics and Astronautics.

[34] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating Eulerian Fluid Simulation With Convolutional Networks, June 2017.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[36] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.