Journal of Computational Science 62 (2022) 101741

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Accelerating urban scale simulations leveraging local spatial 3D structure

Sergio Iserte [a,*], Aina Macías [a], Raúl Martínez-Cuenca [a], Sergio Chiva [a], Roberto Paredes [b], Enrique S. Quintana-Ortí [c]

[a] Universitat Jaume I (UJI), Dpt. of Mechanical and Engineering Construction (DEMC), Spain
[b] Universitat Politècnica de València (UPV), Dpto. de Sistemas Informáticos y Computación (DSIC), Spain
[c] Universitat Politècnica de València (UPV), Dpto. de Informática de Sistemas y Computadores (DISCA), Spain

ARTICLE INFO

Keywords:
Computational fluid dynamics
Structured mesh
Convolutional neural network
High performance computing
Data-driven simulations

ABSTRACT

This paper presents a hybrid methodology for accelerating Computational Fluid Dynamics (CFD) simulations intertwining inferences from deep neural networks (DNN). The strategy leverages the local spatial data of the velocity field to leverage three-dimensional convolutional kernels within DNN. The hybrid workflow is composed of two-step cycles where CFD solvers calculations are utilized to feed predictive models, whose inferences, in turn, accelerate the simulation of the fluid evolution compared with traditional CFD. This approach has proved to reduce 30% time-to-solution in an urban scale study case, which leads to generating massive datasets at a fraction of the cost.

1. Introduction

Traditional Computational Fluid Dynamics (CFD) simulations are able to compute and analyze the fluid flow behavior from a set of Partial Differential Equations (PDE) [1]. This permits the optimization of the performance of a wide variety of industrial devices and facilities in fields such as aerodynamics [2,3], industrial mixing [4,5], nuclear reactor safety [6] and wastewater treatment [7,8]. The ability of these codes to forecast future states of the fluid flow makes them also a valuable tool for environmental studies [9,10]. Here, the focus lies on the dispersion of contaminants depending on emissions and local climatology to prevent risks to the population.

The huge computational cost of the CFD modeling (both in terms of memory and time) strongly limits their practical implementation. Recent studies try to speed up CFD simulations by using Artificial Intelligence (AI) techniques following two main approaches: (1) learning from simulations and predicting results using neural networks [11]; and (2) including predictive models in the CFD solvers to work over a specific problem [12]. The combination of both approaches has provided such promising results that it has given birth to the new area of knowledge *Data-driven CFD* [13]. It is interesting to note that AI methods have been also leveraged in other tasks of CFD such as, for instance, automating the spatial discretization of the fluid flow domain by estimating optimal meshes based on large sets of examples [14].

This paper contributes with a novel methodology that combines stages of CFD resolution and predictions to accelerate the calculation of transient flows in a specific 3D domain leveraging its local spatial structure. First, the domain for the simulations is established. Then, a conventional transient CFD solver is used to train a Deep Neural Network (DNN) consisting of convolutional layers that rely on 3D kernels to detect patterns and extract the physical characteristics of the simulated flow. Finally, the resulting DNN provides spatial–temporal predictions of the flow evolution within the chosen domain in a small fraction of the time needed by the transient CFD solver.

The proposed methodology calculates the flow evolution by alternating stages of transient CFD simulations and predictions of the DNN. In this way, the hybrid approach profits from both the reliability of CFD resolutions to mitigate error accumulation and propagation and the inference speed of predictive models.

The reduction of the time-to-result while achieving a low error rate can be crucial when simulating computationally expensive CFD models, particularly, in the case of generating datasets for training neural networks with the aim of making fluid dynamic predictions. For this purpose, datasets are expected to be composed of many similar simulations, where boundary conditions can be slightly altered [15,16]. This operation provides a dataset with a large variety of scenarios to train the DNN, which can then learn and generalize solutions.

The rest of the paper is structured as follows: Section 2 summarizes the related work. Section 3 presents the study case and how it has been modeled. Section 4 introduces the hybrid solver based on CFD and DNN and the interfaces developed to integrate both subsystems. Section 5 describes the training and evaluation of the devised deep learning model and details the results provided by the hybrid model. Finally, Section 6 concludes this work and states open research lines.

* Corresponding author.
E-mail address: siserte@uji.es (S. Iserte).

https://doi.org/10.1016/j.jocs.2022.101741
Received 10 February 2022; Received in revised form 20 May 2022; Accepted 1 June 2022
Available online 15 June 2022

2. Related work

The incursion of AI methods in CFD simulations to reduce the time-to-solution while providing an acceptable accuracy is a current hot research topic [17]. Several strategies are being followed to provide predictive models capable of accelerating simulations, as well as improving turbulence closure modeling or enhancing Reduced-Order Models (ROM).

Deep learning is used in CFD to infer the physic behavior that allows predicting the evolution of a fluid flow. In principle, it may seem that the ultimate aim here is to deploy a neural network that completely substitutes the numerical solvers of CFD, in other words, use a predictive model to simulate flows virtually in real-time. However, the current state of the art is still far from accomplishing that mission [18]. In the following paragraphs, we refer to some of the most relevant efforts, related to this work.

In [19], authors complement their CFD with deep learning surrogate models of specific subdomains of the case. Especially those that are highly computational demanding. For this purpose, the authors leverage recurrent neural networks (RNN) to predict the unsteady aerodynamic loads of free pitching and plunging airfoil in a transonic flow field.

The work in [20] presents a real-time iterative model that learns from previous data and predicts subsequent temperatures in a voxel. This project trains the model with the available data (calculated + predicted) at each step to make new predictions, which means that the error may be propagated in case of a bad prediction.

Tompson et al. [21] used a predictive model to compute just one of the CFD solver tasks. Specifically, CFD solvers for incompressible flows need to ensure that the computed velocity fields are divergence-free. In practice, this involves a high time-consuming resolution of a Poisson equation that was accelerated thanks to the training of a ConvNet model.

In [22], the authors include the actual equations of the physics models to data-driven approaches in which neural networks are trained to fit the physical laws (i.e.: mass or momentum conservation). They designed data-driven algorithms for inferring solutions to nonlinear PDEs and construct physics-informed surrogate models. Similarly, in [23] physics laws were embedded within the convolutional layers of their network to improve temporal predictions of three-dimensional unsteady vortex dynamics, which are expected to accumulate and propagate the errors when the model is fed with only predicted data.

More related to wind urban environments, there have been several AI applications reviewed in [24] including field regressors, Bayesian deep learning, dynamic node decomposition, cyber–physical systems, autonomous morphing, or digital twins. Particularly, in [25] authors devised a Non-Intrusive Reduced Order Model (NIROM) combining Proper Orthogonal Decomposition (POD) and machine learning techniques. Their results show that the model is capable of predicting the evolution of a turbulent flow in a quasi-steady state. The methodology based on reducing the temporal characteristics before learning is also used in [15], where the authors propose a method based on a convolutional autoencoder that computes a latent space and a Long Short Term Memory (LSTM) network for temporal evolution learning. A more general approach can be found in [16], where authors present CFDNet, a deep learning framework coupled with a physical simulator for RANS models. These solutions are based on reducing the data dimensionality, which in turn removes the local spatial information of the data.

There are also data-driven solutions for Lagrangian mechanics. For instance, the authors in [26] present a general-purpose framework for learning from data of particle-based representation of physics. Nevertheless, it is important to note that the approach taken in this manuscript leverages Eulerian mechanics for calculating fluid motion.

At the end of the day, scientists aim to reduce the extremely large computation time of some simulations using AI (machine or deep learning) techniques. In this paper, a novel hybrid methodology for inferring flow evolution that combines traditional CFD simulation and DNN predictions is presented. The strategy leverages the accuracy of CFD solvers to feed a deep 3D convolutional neural network producing predictions which are then used to accelerate the simulation of the fluid evolution. These two-stage cycles will accelerate simulations thanks to the interleaved predictions, which can be translated in non-negligible time, energy, and cost savings when generating large datasets. To the best of our knowledge, no CFD-DNN hybrid solver like the one proposed in this work has previously incorporated a fully convolutional approach to accelerate CFD simulations.

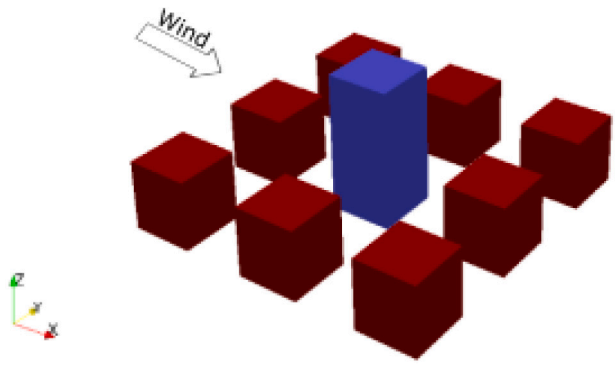


Fig. 1. Representation of the case *simple city blocks* of AIJ. X-axis is aligned with the wind direction and the Z-axis with the vertical plane.

3. Case description

The present study considers a flow scenario comprising several city blocks in an urban area. This configuration has been extensively characterized in wind tunnel tests by the Architectural Institute of Japan (AIJ) and serves for the validation of CFD simulations aiming at the prediction of pedestrian wind environment around buildings [27, 28]. Particularly, the experimental test case C (simple city blocks) of AIJ [29] has been targeted. Fig. 1 represents the case and the wind direction. Notice that the figure illustrates that the X-axis is aligned with the wind direction and the Z-axis with the vertical plane.

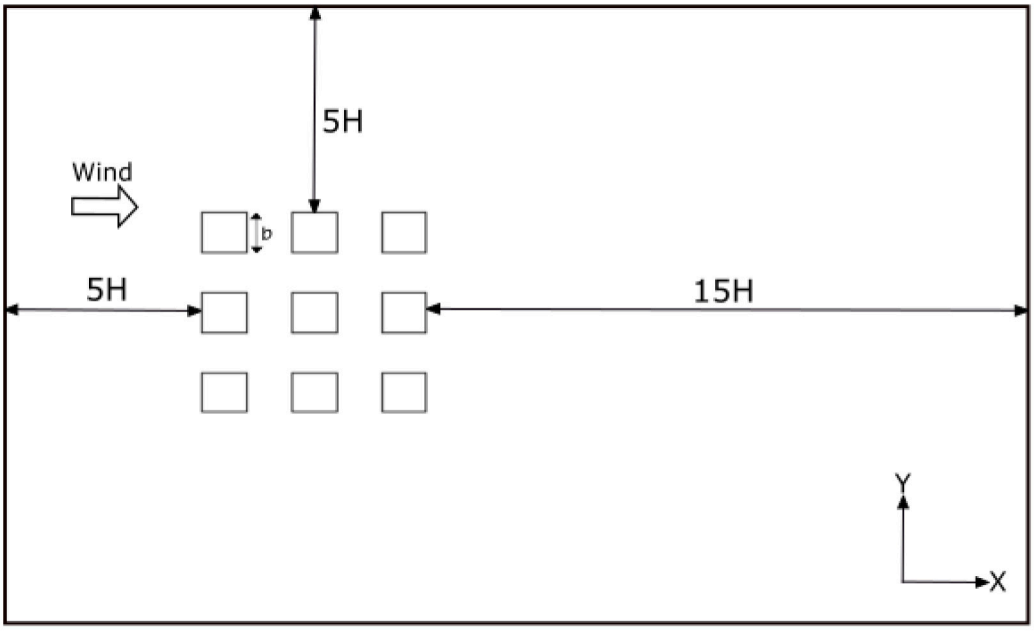
The computational domain limits for the case are established following the COST guideline [30]. Fig. 2 depicts a vertical 2(a) and an horizontal 2(b) section of the domain under study. The resultant domain is configured being $H = 0.4$ m the height of the highest-rise building, and $b = 0.2$ m the height of the low-rise buildings.

The case is discretized into a mesh harnessing the *blockMesh*[1] utility provided by OpenFOAM [31] software, to create a three-dimensional structured grid composed of hexahedral cells, with physical values, such as velocity, pressure, etc., centered in each cell. The methodology presented in this work is spatial dependent because it is based on convolutional kernels, and therefore unstructured meshes cannot be used directly. However, if the unstructured mesh values are interpolated into a structured grid, data could be used in the hybrid methodology. A standard grid is established to ensure computational accuracy for CFD simulations. Fig. 3 shows the vertical cross-section of the computational grid distribution around the buildings in the middle of the Y domain (Fig. 3(a)) and at the bottom of Z (Fig. 3(b)). As a result, the generated mesh is conformed to 374,400 grid points.

Flow equations are modeled using the Unsteady Reynolds-averaged Navier–Stokes (URANS) equations. Particularly, the pimpleFOAM solver was used for their numerical resolution, providing an accurate (though highly time-consuming) description of the flow evolution. To grant

[1] https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.3-mesh-generation-with-the-blockmesh-utility

(a) Vertical section.



(b) Horizontal section.

Fig. 2. Computational domain for H for AIJ case C following the COST guidelines.

a smooth initialization of the transient solver, the initial conditions are computed from a steady-state resolution using the simpleFOAM solver. It is beyond the scope of this paper to conduct a more detailed evaluation of these specific CFD solvers, but a detailed description can be found in the OpenFOAM documentation.[2]

4. CFD-DNN hybrid solver

This section details the methodology implemented by the hybrid solver. Furthermore, the dataset generated to train and evaluate the predictive model is described in detail.

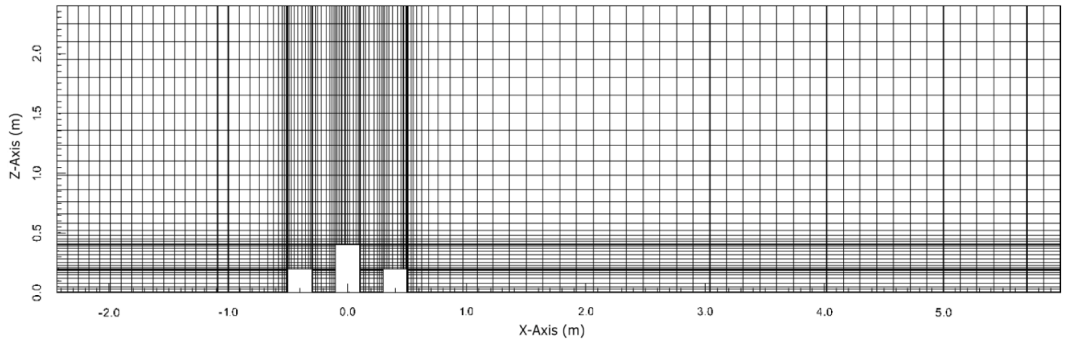
The solution proposed in this work provides an accelerated alternative to the traditional CFD simulations. As introduced in Section 2, the high accuracy of CFD is achieved by solving complex PDEs which usually require large computational times. The DNN offers a technique that provides highly-reduced time-to-solution at the expense of small errors. Alternatively, the predictive model could be applied to subdomains of the case under study [19], or certain periods of the temporal evolution [25].

The hybrid solver combines both approaches, CFD and DNN, to rapidly return a solution while preventing error accumulation and propagation. Fig. 4 depicts how the different subsystems interact. Every cycle (the ith cycle is represented) starts with the CFD computation of N_{CFD} solved timesteps. These results are kept in memory, and just a small subset of $N_{w,CFD}$ timesteps (green squares) are stored on disk at a given write interval. Then, the last three written timesteps (also known as snapshots) serve as input for the DNN after a proper transformation (T) of the CFD results into a format that the DNN solver can process. Next, the DNN computes N_{DNN} predicted timesteps as output (orange squares). The cycle ends with the parsing (P) from the three-dimensional DNN format to the CFD flat-array domain so that the next cycle starts. These cycles can be repeated as many times as needed to provide the flow evolution. In this work, the cycle is composed of $N_{w,CFD} = 10$ written timesteps followed by $N_{DNN} = 10$ predicted timesteps.

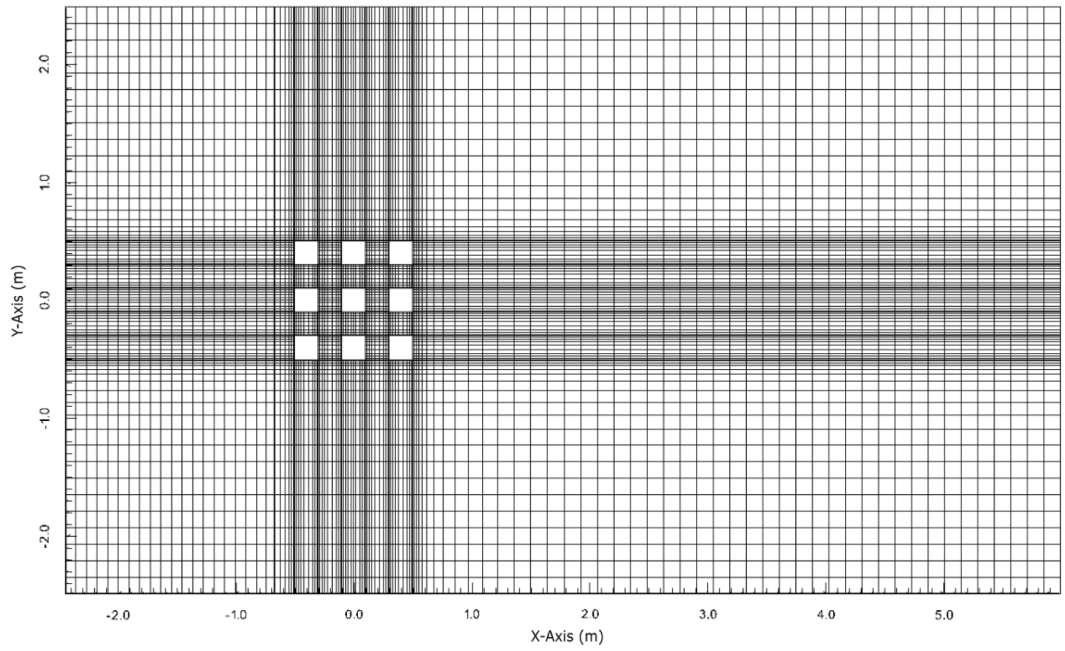
The election of three time steps for DNN input intents to capture the acceleration that theoretically corresponds to the second derivative of the displacement. The 10-timestep output is chosen as a trade-off between performance and accuracy, which for this specific study returned appropriate results.

The presented hybrid solver employs OpenFOAM and Keras/Tensorflow. In this regard, OpenFOAM velocity files for every written timestep are transformed into NumPy [32] arrays (T). The NumPy

[2] https://www.openfoam.com/documentation/user-guide/a-reference/a.1-standard-solvers



(a) Vertical plane $Y = 0$.



(b) Horizontal plane $Z = 0$.

Fig. 3. Computational mesh of the discretized domain depicted in Fig. 2. Range of X-axis: [-2.44:5.98]. Range of Y-axis: [-2.48:2.48]. Range of Z-axis: [0:2.4].

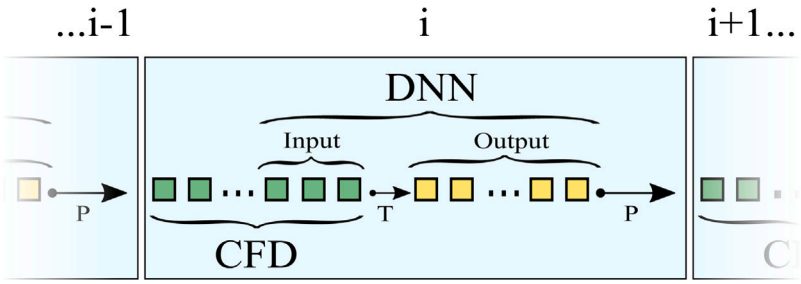


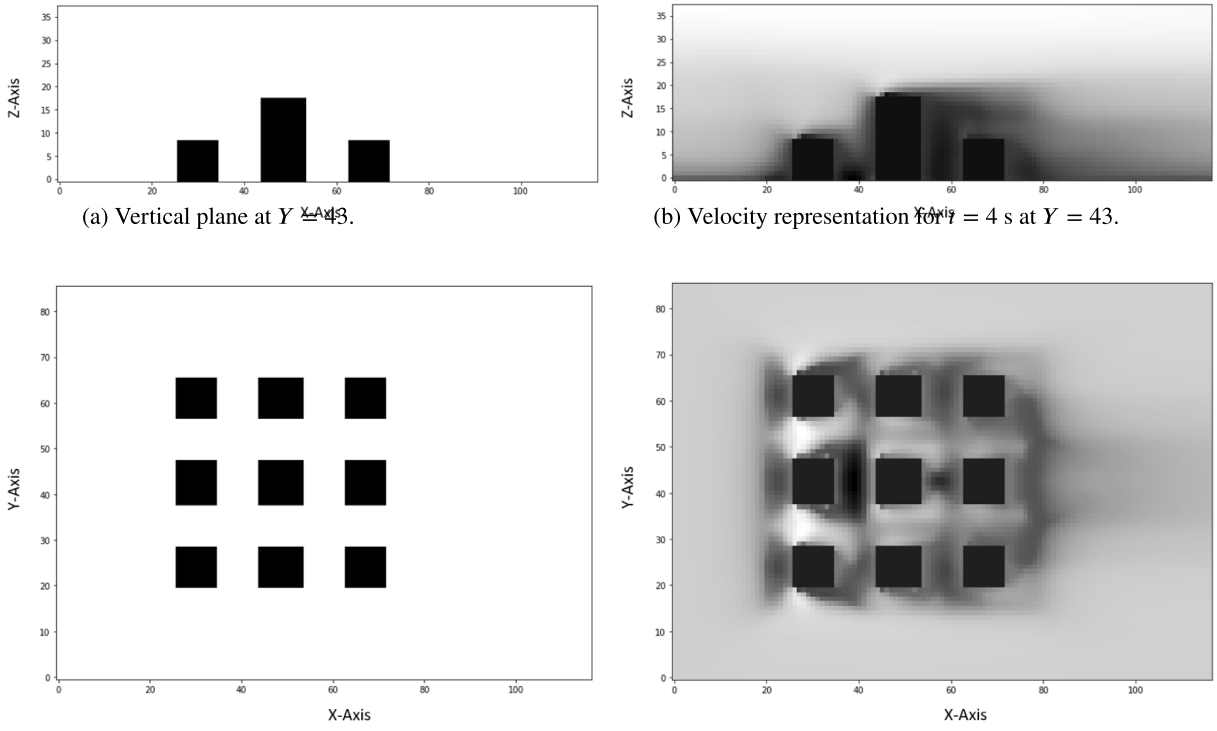
Fig. 4. Hybrid CFD-DNN model scheme.

arrays returned by the DNN are parsed to the OpenFOAM expected file format (P). Through these format conversions, the hybrid solver intertwines both subsystems.

The OpenFOAM format represented in a cartesian system in Fig. 3 has to be transformed to an equidistant space handled by NumPy. Each point in the OpenFOAM domain matches a cell in the NumPy tensor of shape $117 \times 86 \times 38$, corresponding respectively to axes X, Y, and Z of the domain (see Fig. 5). For this purpose, the primarily structured mesh of 382,356 cells is shrunk to 374,400 by setting to zero the cells that correspond to the interior of the buildings (these nodes are removed from the CFD mesh). This conversion can be easily done thanks to the multiblock-structured nature of the mesh with matching cell faces. Figs. 5(a) and 5(c) represent the same planes, once the domain is converted, of Figs. 3(a) and 3(b), respectively. As an example of the transformed domain representation, Figs. 5(b) and 5(d) show how the velocity field within the domain would be after 4 s of time evolution (400 written timesteps).

4.1. Dataset generation

The CFD simulations have been performed with OpenFOAM v2006 and executed in Tirant III supercomputer at Universitat de València



(a) Vertical plane at $Y = 43$.

(b) Velocity representation for $t = 4$ s at $Y = 43$.

(c) Horizontal plane at $Z = 0$.

(d) Velocity representation for $t = 4$ s at $Z = 0$.

Fig. 5. NumPy representation of the domain with the shape (117, 86, 38) corresponding to dimensions X, Y, and Z, respectively. Range of X-axis: [0:116]. Range of Y-axis: [0:85] in (c) and (d). Range of Z-axis[0:37] in (a) and (b).

(UV). The servers in this platform are equipped of two Intel Xeon SandyBridge E5-2670 sockets, and 32 GB of main memory.

The dataset used for training and evaluating the neural network is composed of 31 transient CFD simulations. Details about these simulations and how they have been preprocessed to conform the dataset are described next.

4.1.1. CFD simulations

The inflow conditions proposed by [33,34] were used to ensure a homogeneous Atmospheric Boundary Layer (ABL). They are implemented in OpenFOAM as the following utilities:

- `atmBoundaryLayerInletVelocity` defines the average wind velocity profile represented by the logarithmic law:

$$U(z) = \frac{u^*}{\kappa} \ln\left(\frac{z + z_0}{z_0}\right), \tag{1}$$

where κ is the von Karman constant ($\kappa = 0.4$), z_0 [m] is the aerodynamic roughness length, and u^* [m/s] is the friction velocity, given by

$$u^* = \frac{\kappa u_{ref}}{\ln\left(\frac{z_{ref}+z_0}{z_0}\right)}, \tag{2}$$

where u_{ref} [m/s] is the user-defined reference velocity at a reference height z_{ref} [m].
- `atmBoundaryLayerInletK` defines the default generalized log-law for the turbulent kinetic energy.
- `atmBoundaryLayerInletOmega` defines the default inlet profile for the specific dissipation of the eddies.

Notice that *slip (free-slip)* conditions are used for the lateral and top boundaries of the domain; while *zero gradient* condition is applied to the outflow boundary.

The transient CFD simulations were initialized from a common steady-state CFD simulation to ensure an adequate smooth initialization of the transient solver. The steady-state simulation used a constant inlet profile with $u_{ref} = 1$ m/s, $z_{ref} = 0.15$ m and $z_o = 0.001$ m and was run up to convergence, defined at a root mean square error (RMSE) lower than 10^{-4}.

In these experiments, the transient simulations are run to simulate flow evolution to $t = 6.51$ s with a solving timestep of 0.001 s and a writing interval of 10. At second 6.51 the residual thresholds of the solver convergence criteria are lower than an RMSE of 10^{-5} for all variables.

In this regard, solutions are written to disk every 0.01 s of flow time. Consequently, DNN predicted timesteps comprise 0.01 s. During the first 0.5 s of evolution, u_{ref} is linearly increased from its initial value (1 m/s) up to a value $u_{ref,end}$, keeping this value for the rest of the execution. A total of 31 simulations were launched with $u_{ref,end}$ ranging between 3 m/s (minimum velocity profile) and 6 m/s (maximum velocity profile) in steps of 0.1 m/s. Fig. 6 represents the minimum and maximum wind profiles. These profiles show, for each height (Y-axis), the expected velocity (X-axis). Notice that all the profiles configurations of $u_{ref,end}$ lie within these bounds.

4.1.2. Data preprocessing

The dataset generation stage involves the methods of filtering, remapping, reshaping, standardizing, and splitting the data in order to produce a dataset that feeds the deep convolutional neural network.

In this work, only the velocity field (in its three dimensions) within the physical domain is taken into account. For this reason, pressure or turbulence-related metrics are filtered out from raw data. Furthermore, the DNN training was focused on just the first two seconds of time evolution, which involves 200 written timesteps of the transient simulations. Simulations up to $t = 2$ s contain enough information for fitting the predictive model. After completion of this process, the initial dataset is composed of 31 cases of 200 timesteps each, with 374,400 three-component vectors corresponding to the velocity in the mesh cell locations.

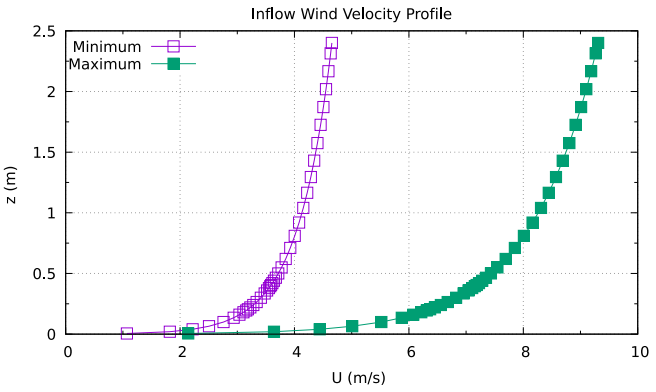


Fig. 6. Inflow profiles for the cases with maximum and minimum values for $u_{ref,end}$.

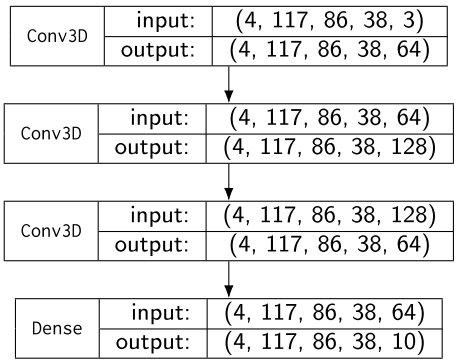


Fig. 7. CNN architecture per velocity dimension. For each layer it is indicated its type (leftmost) and the data shape in the input and output (rightmost). Dimensions at layers' input/output correspond to mini-batch size, width, height, depth, and channels, respectively.

As described in Section 3, the domain studied in this work relies on a structured mesh. To leverage the spatial arrangement of data, the next step rearranges the velocity flat array of 374,400 elements into a 3D space adapted to the convolutional kernels. The resultant domain, with dimension 117 × 86 × 38 (X, Y, and Z, respectively), counts with a total of 382,356 cells. Furthermore, the domain contains 7956 additional cells than the OpenFOAM flat array. These extra cells are associated with the buildings, and they are necessary for computing 3D convolutions. At this point, the dataset shape is 31 × 200 × 117 × 86 × 38 × 3, respectively describing the number of cases, number of written timesteps per case, cells in domain dimension X, cells in domain dimension Y, cells in domain dimension Z, and velocity dimensions.

Since the proposed predictive model handles the velocity in three different neural networks, one per velocity dimension, the shape of each dataset is 31 × 200 × 117 × 86 × 38, where the components respectively correspond to the number of cases, number of written timesteps per case, cells in domain dimension X, cells in domain dimension Y, cells in domain dimension Z. Velocity values are then transformed to have a distribution of mean zero and standard deviation of one. The resultant dataset has a size in memory of 26.49 GB. Values are stored in a 32-bit floating point datatype.

Finally, the dataset is split into train and test subsets. In particular, the cases are shuffled and 80% of them (for a total of 24 cases) are assigned to the training dataset, while the remaining seven cases to the testing dataset. In addition 20% of the training cases are used for cross-validation.

4.2. CNN training and evaluation

The DNNs have been trained in CTE-Power at Barcelona Supercomputing Center (BSC) with the following software: Python 3.7.4, NumPy 1.18.4, Scikit-learn 0.23.1, and Keras 2.4 over Tensorflow [35] 2.3. Each CTE-Power node is equipped with two IBM Power9 8335-GTH processors, 512 GB of main memory, and four GPU NVIDIA V100 with 16 GB HBM2.

The predictive model devised in this work is designed to provide multi-step predictions of a 3D variable. For this purpose, three independent identical convolutional neural networks have been implemented, one for each velocity dimension.

These networks are composed of three 3D convolutional layers plus a dense layer, as Fig. 7 depicts. In the figure, for each layer, the input and output data shape is showcased. These layers expect data to be formatted as (*mini-batch size*, *height*, *width*, *depth*, *timesteps*). The mini-batch size is determined by the GPU memory which, for the proposed model, cannot fit larger mini-batches, and the last dimension, corresponding to the channels of the layers, is used to convey the consecutive timesteps.

Table 1
Training epochs and time per neural network.

Dimension	Epochs	Time
X	34	9,644 s
Y	29	8,169 s
Z	47	13,378 s

Table 2
Aggregated errors of evaluation cases per velocity dimension.

Dim.	MSE			
	Min.	Max.	Mean	Std. dev.
X	8.31×10^{-4}	1.03×10^{-1}	6.75×10^{-3}	5.20×10^{-4}
Y	1.50×10^{-3}	2.37×10^{-1}	1.28×10^{-2}	5.33×10^{-3}
Z	1.78×10^{-3}	1.74×10^{-1}	1.69×10^{-2}	2.16×10^{-3}

The convolutional layers are configured with a kernel size of 3 × 3 × 3 with a padding set to zero. Furthermore, the padding type is defined to "same" which configures a same convolution where the output matrix is of the same dimension as the input matrix. The number of filters defined for the three layers is 64, 128, and 64, respectively. Non-linearity in the convolutional layers is realized with the *LeakyReLU* activation function, a variation of the rectified linear unit (ReLU) which allows small positive gradients when the unit is not active [36].

The dense layer counts with the number of neurons corresponding to the number of steps that will be predicted. Particularly, this layer is configured with 10 neurons meaning that the prediction will return 10 written timesteps. This dense last layer is activated linearly since it is a regression problem

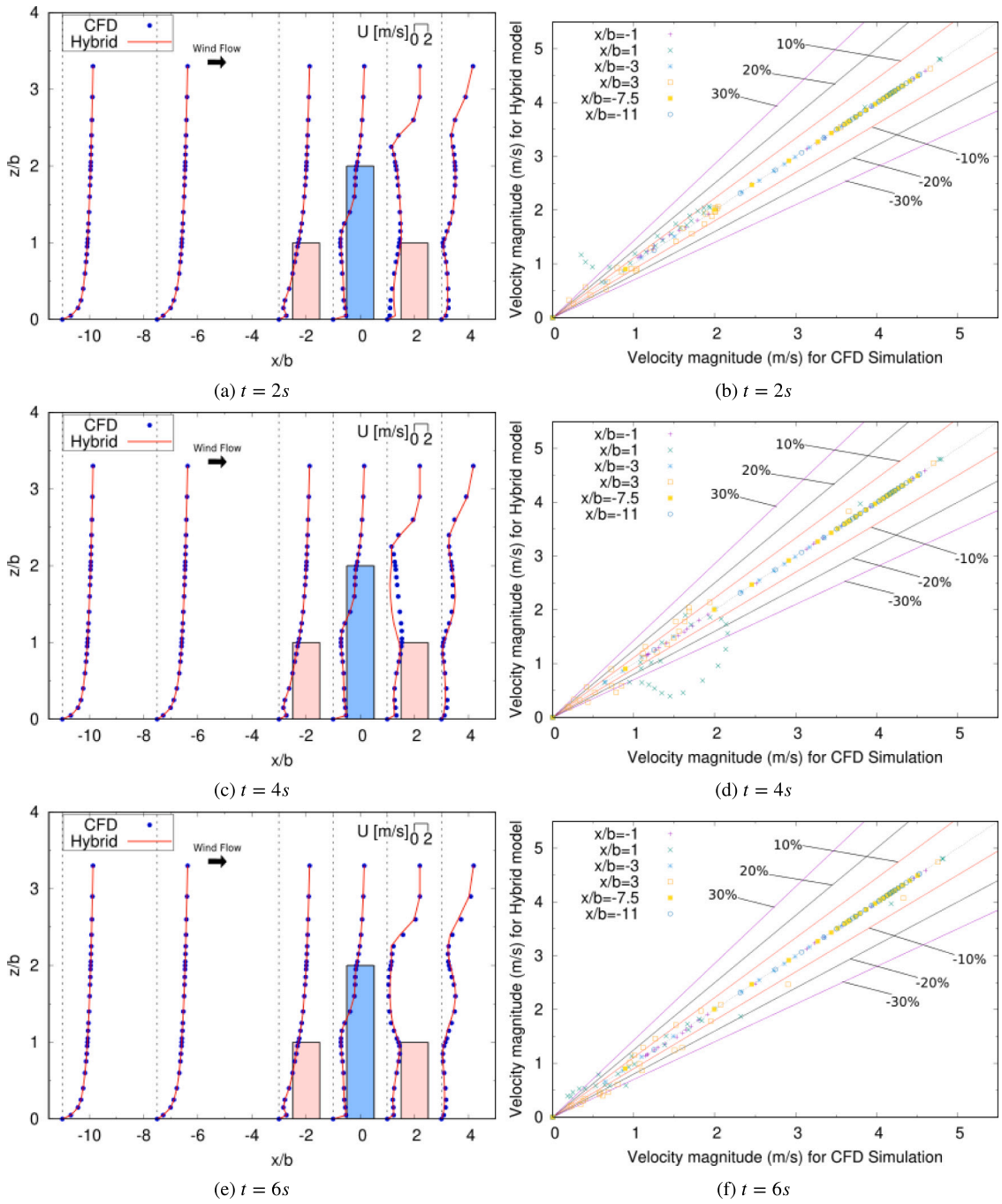
It is important to note that the network expects three observation samples as input. For this purpose, instead of using recurrent neural networks, we have used the channels of the input layer (last parameter of the input tensor) to convey the values of three consecutive written timesteps of a dimension of the velocity.

Mini-batches are composed of four elements, and the model is compiled with the *Adam* optimizer [37], with a learning rate of 0.0001, to update weights and biases within the network, which relies on the mean square error (MSE) loss function:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2. \tag{3}$$

Each independent neural network, one per velocity dimension, has been trained using a GPU of CTE-Power cluster. Table 1 compiles the number of epochs and time employed in each training.

The model is evaluated with the test subset composed of seven cases.



(a) $t = 2s$ (b) $t = 2s$

(c) $t = 4s$ (d) $t = 4s$

(e) $t = 6s$ (f) $t = 6s$

Fig. 8. CFD and Hybrid results comparison for six velocity magnitude profiles in a vertical plane $Y = 0$ for different three times. (a, c, e) Comparisons of the vertical profiles; (b, d, f) Comparisons of the percentage difference.

Table 2 shows the absolute errors of the evaluation for this subdataset. In this regard, testing cases have been independently evaluated and their errors aggregated with *minimum*, *maximum*, *mean* and *standard deviation* per dimension.

When predicting the dimension X of velocity, the model is able to provide lower errors than for the other dimensions. Since the wind blows from the X-axis, this dimension contains more data diversity and the model is likely to learn more patterns. In contrast, axes Y and Z present more homogeneity in their data, which undermines pattern detection. As a result, the model encounters more difficulties to learn and predictions are not so accurate.

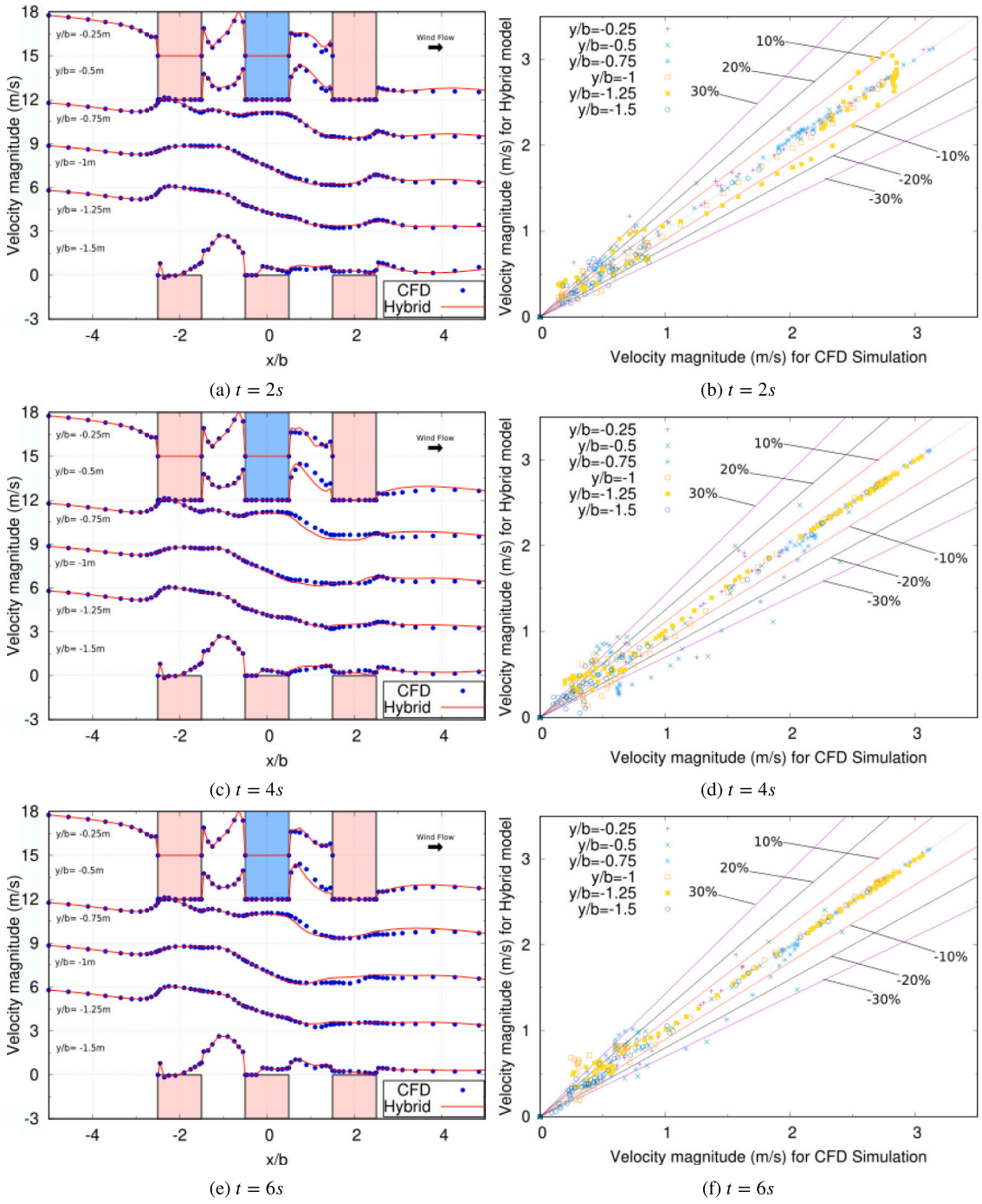
A priori, the evaluation errors are acceptable since all the values have been standardized. Although maximum errors are not ideal, the mean and the standard deviation show small errors with low variance.

5. Results

This section compares the hybrid model execution with the CFD simulation, as well as their performance.

5.1. Hybrid model evaluation

Next, we illustrate the accuracy of the hybrid solver by performing an in-depth analysis of one of the cases. Since the standard deviation of the errors calculated during the evaluation of the predictive model (see Table 2) shows little variations when predicting the different cases of the test subdataset, an arbitrary case from the test dataset was selected ($u_{ref,end}$ = 3.5 m/s). In this subsection, we present the results



(a) $t = 2s$ (b) $t = 2s$

(c) $t = 4s$ (d) $t = 4s$

(e) $t = 6s$ (f) $t = 6s$

Fig. 9. CFD and Hybrid results comparison for six velocity magnitude profiles in a horizontal plane $z = 0.1$ m for different three times. (a, c, e) Comparisons of the horizontal profiles; (b, d, f) Comparisons of the percentage difference.

obtained with the proposed hybrid solver in different temporal and spatial points.

Fig. 8 compares the resulting velocities of the CFD and hybrid solvers in three different instants: $t = 2$ s, $t = 4$ s, and $t = 6$ s. Figs. 8(a), 8(c), and 8(e) compare the velocity profiles at six vertical lines in the ZX plane, specifically at locations: $x/b = -11, -7.54, -3, -1, 1$, and 3. Notice that x/b in the X-axis and z/b in the Y-axis represent the non-dimensional horizontal and vertical distance, respectively. From these results, it is patent that the hybrid solver can properly capture the spatial–temporal evolution of the flow for most locations at any time. Only a small deviation from the CFD is observed at line $x/b = 1$ for $t = 4$ s, where the velocity magnitude is slightly under-predicted by the hybrid model. Figs. 8(b), 8(d), and 8(f) directly compare the velocities at the same locations of their peer Figs. 8(a), 8(c), and 8(e), respectively. The plots showcase the strong correlation in the studied points between CFD and hybrid solvers. Some discrepancies appear for the lowest velocities due to the recirculation regions behind the blocks, particularly in the high-rise building. Overall, the deviations remain below 20%, especially for high-velocity values.

Fig. 9 shows the results corresponding to a horizontal plane $Z = 0.1$ m, close to the ground. Again, some discrepancies are observed in the space between blocks, particularly behind the high-rise building due to the complex eddies generated behind it. Hybrid results at the majority points are adequate, with an average velocity difference lower

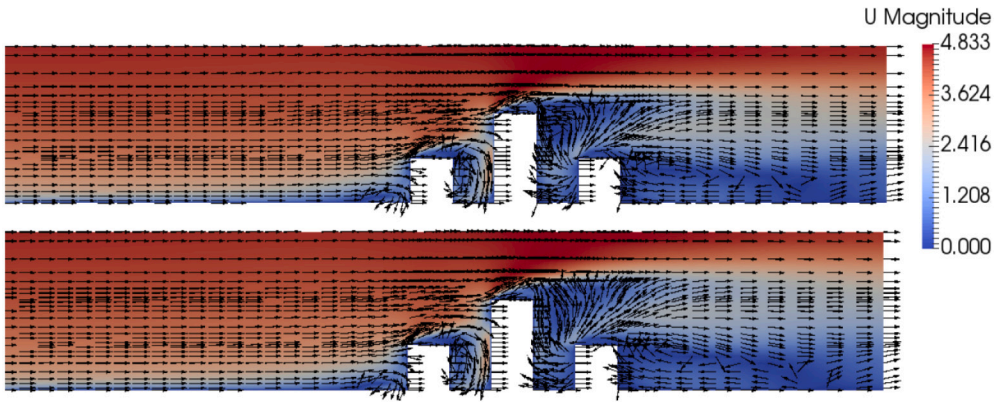


Fig. 10. The velocity vectors and contours in a vertical plane at $Y = 0$ for CFD simulation (top) and Hybrid (bottom) at $t = 6.51$ s.

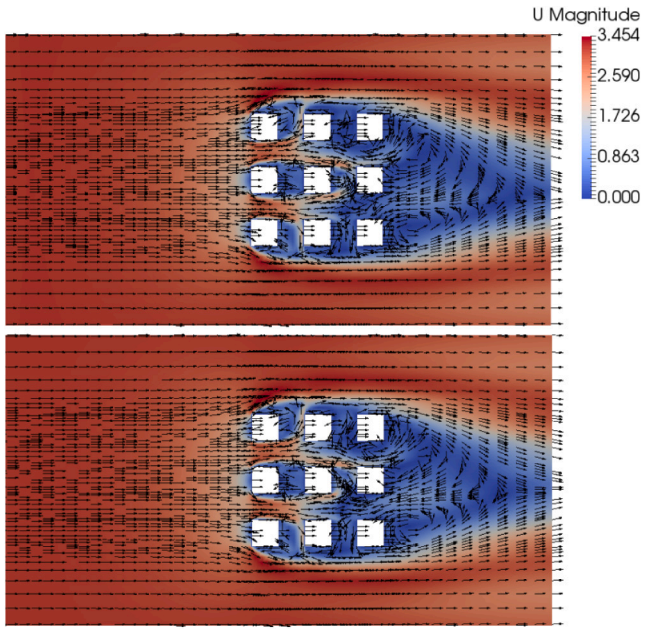


Fig. 11. The velocity vectors and contours in a horizontal plane at $Z = 0.05$ m for CFD simulation (top) and Hybrid (bottom) at $t = 6.51$ s.

Table 3
Execution time comparison of case $u_{ref,end} = 3.5$ m/s.

Solvers	#Simulated TS	#Predicted TS	Execution time
CFD	652	0	15,891.16 s
Hybrid	322	330	11,018.57 s

Table 4
Stages of a hybrid simulation. Notice that times in the *time evolution* stage are the average of the 32 cycles values.

Stage	Operation	#Written timesteps	Time
1. Boot	Initialization	–	2.08 s
	Simulation	2	75.10 s
2. Inference	Generate DS	–	5.57 s
	Prediction	10	38.20 s
	Regenerate DS	–	11.59 s
3. Time evolution (32 cycles)	Simulation	10	284.81 s
	Inference stage	10	55.36 s

than 20%. The differences in a few locations are higher than 30% characterized by under-predicting the CFD results as mentioned above.

Finally, Figs. 10 and 11 present the mean velocity contours and vectors in vertical (plane $Y = 0$) and horizontal (plane $Z = 0.05$ m) cross-sections. These results demonstrate that the CFD-DNN model captures the main trends of the flow, especially for the separation region around the blocks and recirculation region behind blocks, despite the small deviations in velocity values.

5.2. Performance analysis

Once the hybrid solver is validated, in this section, we study its performance in terms of time compared with the traditional CFD solver.

The interleaved predictions provided by the DNN in the hybrid solver are expected to reduce the time-to-solution by accelerating the flow simulation. For this purpose, the execution of the simulation case described in the previous section, ($u_{ref,end} = 3.5$ m/s), is analyzed. All the executions described in this section have been run on a single core of a Tirant III server in an effort to provide a fair comparison.

For evaluation purposes, the flow evolution has been extended to second 6.51 of the simulation, calculating a total of 652 written timesteps. Table 3 compares the experiment execution times for both approaches. The CFD solver computes all timesteps in nearly 15,900 s. The hybrid solver combines CFD and DNN leverages OpenFOAM to calculate 322 written timesteps and the DNN to infer 330 predicted timesteps. The results reveal that the hybrid solver reduces the execution time by more than 30%.

In more detail, Table 4 analyzes the stages of the hybrid solver workflow digging deeper into its operations.

The workflow begins with an initialization of the necessary data structures and file systems. During this operation, the predictive model is also loaded in memory.

The *boot* stage also involves a two-written timestep CFD simulation, starting from the steady-state initialization ($t = 0$ s) to end at $t = 0.02$ s. The three resulting written timesteps are used for the first inference of 10 predicted timesteps.

The *inference* stage comprehends the operations of generating the dataset (T) for feeding the predictive model, the prediction, and the conversion (P) of the predicted timesteps into the OpenFOAM format, to continue with the simulation (recall Fig. 4).

Finally, in the *time evolution* stage, the hybrid solver performs 32 cycles that combine CFD simulation and DNN inference. Particularly, every cycle is composed of 10 written timesteps and 10 predicted timesteps.

6. Conclusions

This paper presents a novel hybrid methodology that combines CFD calculations and DNN. Compared with the variety of approaches that are being taken to integrate AI with physics simulations, the hybrid

model proposes an intermediate solution between the traditional CFD simulations and the continuous spatial–temporal flow inferences. The results show a significant reduction of the execution time (over 30%), while the errors in the velocity values remain typically below 10%, with some conflictive locations in the vortex shredding regions behind the buildings (30% error). The execution time savings could be exacerbated by reducing the ratio between the number of CFD resolved written timesteps and the number of DNN predicted timesteps, at the expense of increasing errors in the velocity field.

The hybrid solver has been proved as an interesting tool for large sets of massive simulations. The training took 8.6 GPU-hours, and the dataset generation around 31.2 CPU-hours (1.3 CPU-hours per case for a total of 24 two-second long cases). The simulation of a case, up to $t = 6.51$ s, using the hybrid solver needs 1.3 CPU-hours (as Table 3 showcased) less than the CFD solver. For this reason, the hybrid model demonstrates that its training cost is amortized in the long run, especially when resorting to paying-per-use facilities.

In summary, the hybrid CFD-DNN solver can reduce the time-to-solution while delivering sufficient accuracy.

The authors have limited the study to a proof of concept that, once validated, can be scaled to other types of flows. Hence, there is still room for improvement in the accuracy of the predictions, which is planned as future work: for instance, expanding the training dataset with other wind directions, or cases or tuning the network model hyperparameters. Also, designing specific models for the different velocity dimensions, or even other types of neural networks such as geometric graph-based, could pose important benefits and extend the flexibility of the presented solution to be adopted in other types of meshes.

Particularly, the authors aim to apply hybrid simulations for generating datasets of transient flows in a fraction of the time that common CFD simulations would take. In this regard, the methodology presented in this paper is enormously useful to create a dataset of converged simulations of a given case. The generated dataset provides an invaluable tool that can be used in other predictive models or analyses.

All the code developed in this work can be found in https://github.com/siserte/HybridCFDML.

CRediT authorship contribution statement

Sergio Iserte: Software, Investigation, Writing – original draft. **Aina Macías:** Validation, Visualization. **Raúl Martínez-Cuenca:** Formal analysis, Project administration. **Sergio Chiva:** Conceptualization, Supervision, Funding acquisition. **Roberto Paredes:** Methodology, Supervision. **Enrique S. Quintana-Ortí:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Researcher S. Iserte was supported by postdoctoral fellowship APOSTD/2020/026 from GVA-ESF. While researcher A. Macias was supported by predoctoral fellowship FDGENT from GVA. CTE-Power cluster of the Barcelona Supercomputing Center, and Tirant III cluster of the *Servei d'Informàtica* of the University of Valencia were leveraged in this research. Authors want to thank the anonymous reviewers whose suggestions significantly improved the quality of this manuscript.

References

[1] H.K. Versteeg, W. Malalasekera, An Introduction to Computational Fluid Dynamics: the Finite Volume Method, second ed., Pearson Education Limited, Essex, 2007.

[2] B. Sanderse, S.P.V.D. Pijl, B. Koren, Review of computational fluid dynamics for wind turbine wake aerodynamics, Wind Energy 14 (2011) 799–819, http://dx.doi.org/10.1002/we.458.

[3] Y. Li, K.J. Paik, T. Xing, P.M. Carrica, Dynamic overset CFD simulations of wind turbine aerodynamics, Renew. Energy 37 (2012) 285–298, http://dx.doi.org/10.1016/j.renene.2011.06.029.

[4] T. Kumaresan, J.B. Joshi, Effect of impeller design on the flow pattern and mixing in stirred tanks, Chem. Eng. J. 115 (2006) 173–193, http://dx.doi.org/10.1016/j.cej.2005.10.002.

[5] T. Norton, D.W. Sun, Computational fluid dynamics (CFD) - an effective and efficient design and analysis tool for the food industry: A review, Trends Food Sci. Technol. 17 (2006) 600–620, http://dx.doi.org/10.1016/j.tifs.2006.05.004.

[6] A. Guelfi, D. Bestion, M. Boucker, P. Boudier, P. Fillion, M. Grandotto, J.M. Hérard, E. Hervieu, P. Péturaud, NEPTUNE: A New software platform for advanced nuclear thermal hydraulics, Nucl. Sci. Eng. 156 (2007) 281–324, http://dx.doi.org/10.13182/NSE05-98.

[7] Y. Fayolle, A. Cockx, S. Gillot, M. Roustan, A. Héduit, Oxygen transfer prediction in aeration tanks using CFD, Chem. Eng. Sci. 62 (2007) 7163–7171, http://dx.doi.org/10.1016/j.ces.2007.08.082.

[8] S. Iserte, P. Carratalà, R. Arnau, R. Martínez-Cuenca, P. Barreda, L. Basiero, J. Climent, S. Chiva, Modeling of wastewater treatment processes with HydroSludge, Water Environ. Res. (2021).

[9] N.S. Holmes, L. Morawska, A review of dispersion modelling and its application to the dispersion of particles: An overview of different dispersion models available, Atmos. Environ. 40 (2006) 5902–5928, http://dx.doi.org/10.1016/j.atmosenv.2006.06.003.

[10] N. Wright, D. Hargreaves, Environmental applications of computational fluid dynamics, in: Environmental Modelling: Finding Simplicity in Complexity, second ed., 2013, http://dx.doi.org/10.1002/9781118351475.ch6.

[11] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 13-17-August-2016, 2016, pp. 481–490, http://dx.doi.org/10.1145/2939672.2939738.

[12] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. 807 (2016) 155–166, http://dx.doi.org/10.1017/jfm.2016.615.

[13] J.N. Kutz, Deep learning in fluid dynamics, J. Fluid Mech. 814 (2017) 1–4, http://dx.doi.org/10.1017/jfm.2016.803.

[14] K. Huang, M. Krügener, A. Brown, F. Menhorn, H.-J. Bungartz, D. Hartmann, Machine Learning-Based Optimal Mesh Generation in Computational Fluid Dynamics, 2021, pp. 1–22, arXiv:2102.12923.

[15] S. Wiewel, M. Becher, N. Thuerey, Latent space physics: Towards learning the temporal evolution of fluid flow, Comput. Graph. Forum 38 (2) (2019) 71–82, arXiv:1802.10123.

[16] O. Obiols-Sales, A. Vishnu, N. Malaya, A. Chandramowliswharan, CFDNet: A deep learning-based accelerator for fluid simulations, in: Proceedings of the International Conference on Supercomputing, 2020, http://dx.doi.org/10.1145/3392717.3392772, arXiv:2005.04485.

[17] R. Vinuesa, S.L. Brunton, The potential of machine learning to enhance computational fluid dynamics, 2021, arXiv preprint arXiv:2110.02085.

[18] S.L. Brunton, B.R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annu. Rev. Fluid Mech. 52 (1) (2020) arXiv:1905.11075.

[19] S. Srivastava, M. Damodaran, B.C. Khoo, Machine learning surrogates for predicting response of an aero-structural-sloshing system, ArXiv (2019).

[20] A. Paul, M. Mozaffar, Z. Yang, W.K. Liao, A. Choudhary, J. Cao, A. Agrawal, A real-time iterative machine learning approach for temperature profile prediction in additive manufacturing processes, in: Proceedings - 2019 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2019, 2019, pp. 541–550, arXiv:1907.12953.

[21] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin, Accelerating eulerian fluid simulation with convolutional networks, in: 34th International Conference on Machine Learning, ICML 2017, vol. 7, 2017, pp. 5258–5267, arXiv:1607.03597.

[22] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.

[23] S. Lee, D. You, Analysis of a convolutional neural network for predicting unsteady volume wake flow fields, Phys. Fluids 33 (3) (2021) 35152.

[24] A. Kareem, Emerging frontiers in wind engineering: Computing, stochastics, machine learning and beyond, J. Wind Eng. Ind. Aerodyn. 206 (October) (2020) 104320.

[25] D. Xiao, C.E. Heaney, L. Mottet, F. Fang, W. Lin, I.M. Navon, Y. Guo, O.K. Matar, A.G. Robins, C.C. Pain, A reduced order model for turbulent flows in the urban environment using machine learning, Build. Environ. 148 (2019) 323–337.

[26] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P.W. Battaglia, Learning to simulate complex physics with graph networks, in: 37th International Conference on Machine Learning, ICML 2020, PartF16814, 2020, pp. 8428–8437, arXiv:2002.09405.

[27] Y. Tominaga, et al., AIJ Guidelines for practical applications of CFD to pedestrian wind environment around buildings, J. Wind Eng. Ind. Aerodyn. 96 (10–11) (2008) 1749–1761.

[28] R. Yoshie, A. Mochida, Y. Tominaga, H. Kataoka, K. Harimoto, T. Nozu, T. Shirasawa, Cooperative project for CFD prediction of pedestrian wind environment in the architectural institute of Japan, J. Wind Eng. Ind. Aerodyn. 95 (9) (2007) 1551–1578.

[29] AIJ, Guidebook for CFD predictions of urban wind environment, 2021, https://www.aij.or.jp/jpn/publish/cfdguide/index_e.htm. [Online; accessed 11-May-2021].

[30] J. Franke, A. Baklanov, Best Practice Guideline for the CFD Simulation of Flows in the Urban Environment: COST Action 732 Quality Assurance and Improvement of Microscale Meteorological Models, 2007.

[31] H.G. Weller, G. Tabor, H. Jasak, C. Fureby, A tensorial approach to computational continuum mechanics using object-oriented techniques, Comput. Phys. 12 (6) (1998) 620–631.

[32] C.R. Harris, K.J. Millman, S.J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, R. Kern, M. Picus, S. Hoyer, M.H. van Kerkwijk, M. Brett, A. Haldane, J.F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T.E. Oliphant, Array programming with numpy, Nature 585 (7825) (2020) 357–362, http://dx.doi.org/10.1038/s41586-020-2649-2.

[33] P. Richards, R. Hoxey, Appropriate boundary conditions for computational wind engineering models using the k-e turbulence model, J. Wind Eng. Ind. Aerodyn. 46–47 (1993) 145–153, Proceedings of the 1st International on Computational Wind Engineering.

[34] Y. Yang, M. Gu, S. Chen, X. Jin, New inflow boundary conditions for modelling the neutral equilibrium atmospheric boundary layer in computational wind engineering, J. Wind Eng. Ind. Aerodyn. 97 (2) (2009) 88–95.

[35] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: A System for large-scale machine learning, in: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16, USENIX Association, USA, 2016, pp. 265–283.

[36] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in: ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 28, 2013.

[37] D.P. Kingma, J.L. Ba, Adam: A method for stochastic optimization, in: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 2015, pp. 1–15, arXiv:1412.6980.

Sergio Iserte holds the degrees of BS in Computer Engineering (2011), MS in Intelligent Systems (2014), and Ph.D. in Computer Science (2018) from Universitat Jaume I (UJI), Spain. He is currently postdoc researcher (APOSTD20) in the Mechanical and Engineering Construction Dept. at the same University, and course instructor of high-performance computing at Universitat Oberta de Catalunya (UOC). He is currently involved in HPC projects related to parallel distributed computing, resource management, workload modeling, and deep learning for industrial applications.

Aina Macias holds the degrees of BS inIndustrial Technology Engineering (2015), and MS inIndustrial Engineering (2017) from Universitat Jaume I (UJI), Spain. She is currently a Generalitat Valenciana FDGENT Fellow and Research and Teaching Staff (PDI) at Jaume I University since 2018 in the Group of Multiphase Flow in the Department of Mechanical Engineering and Construction. Her thesis is focussed on the development of an odour management protocol integrating the use of sensor networks, the perception of citizens and dispersion models using computational fluid dynamics techniques (CFD).

Raúl Martínez-Cuenca received the Bachelor and Ph.D. in Physics from the Universitat de València, Spain, in 2003 and 2008, respectively. He is Assistant Professor at the University Jaume I in Castelló. His current research interest includes computational fluid dynamics simulations, multiphase flows, and waste water treatment technologies.

Dr. Sergio Chiva Vicent is Full professor at the Mechanical Engineering and Construction Dept at the Universitat Jaume I of Castellón (Spain). His expertise fields relate to the use of CFD techniques for complex simulations, usually applied to multiphase flows and environmental scenarios, mainly in wastewater, petrochemical, and nuclear engineering. Another highlight topic of research has been the development of experimental works and techniques to characterize and measure multiphase flow and odor components. Dr. Chiva has published more than 40 JCR papers and he has supervised 12 Ph.D. thesis.

Dr. Roberto Paredes is Full Professor of Computer Science at Universitat Politècnica de València in Spain (UPV). The research interests of Roberto Paredes include deep learning, machine learning, pattern recognition, biometrics and their application to computer vision and big data analysis. Professor Paredes is also co-founder and CTO of the spin-off Solver Intelligent Analytics. Roberto Paredes is the coauthor of more than 30 articles in international journals, 10 of them in the top of journal rankings and more than 100 articles in proceedings of international and national conferences.

Enrique S. Quintana-Orti received the bachelor and Ph.D. degrees in computer sciences from the Universitat Politecnica de Valencia, Spain, in 1992 and 1996, respectively. He is a Professor in Computer Architecture in the Universitat Poitecnica de Valencia. Recently, he has participated in EU projects such as OPRECOMP, RED-SEA and eFLOWS4HPC. His current research interests include parallel programming, linear algebra, energy consumption, transprecision computing and deep learning as well as advanced architectures and hardware accelerators.