FluidGPT-1: a proof-of-concept model for attention-based flow pattern generation in CFD

Davide Dapelo*[1] and John Bridgeman[1]

[1] Department of Civil and Environmental Engineering, School of Engineering, University of Liverpool, Liverpool, United Kingdom
* Corresponding author

Keywords: CFD, Lattice-Boltzmann, GPT, UNet, CNN, Transformer, Autoregressive

Abstract.

The integration of generative artificial intelligence in computational fluid dynamics (CFD) remains limited, as traditional methods often fail to capture critical flow parameters like viscosity, Reynolds number, or Smagorinsky constant whenever not explicitly passed to the model. To address this, we present FluidGPT-1, a proof-of-concept architecture that merges Generative Pre-trained Transformers (GPTs) with U-Nets to predict time-dependent fluid flow patterns. Unlike U-Net-only models, which rely solely on the immediate previous timestep, our model harnesses transformers' attention mechanisms to capture long-term dependencies, indirectly accounting for essential but missing parameters. We trained the network on 4,081 short synthetic 2D fluid flow simulations generated with the OpenLB Lattice-Boltzmann library, employing random domain geometries, inlet/outlet configurations, and laminar flow conditions. Numerical experiments show that FluidGPT-1 generalizes effectively and accurately predicts future flow patterns without direct parameter inputs. This transformer-based approach reduces computational complexity by eliminating the need for explicit parameter encoding while retaining accuracy across diverse CFD configurations. Our findings suggest that combining GPTs and U-Net can address longstanding challenges in data-driven fluid flow prediction, paving the way for more efficient simulation pipelines and opening new frontiers in AI-driven CFD research.

1. Introduction

The synergy between Artificial Intelligence (AI)—particularly Neural Networks (NNs)—and Computational Fluid Dynamics (CFD) has rapidly expanded in multiple directions, thereby advancing the field of numerical modelling. AI-based surrogates have been developed, in which neural networks predict the relevant outputs for a given problem, in place of a full CFD simulation. For instance, [1] employ Convolutional Neural Networks (CNNs) to estimate wall shear stress in arteries, and [2] integrate CNNs with a transformer-based autoregressive model to predict time time-series particle trajectories and erosion patterns in industrial settings.

AI has also been applied to support CFD modelling itself. For instance, data-driven techniques have been proposed to enhance turbulence modelling in CFD simulations, such as by identifying optimal model parameters [3, 4, 5] or improving closures [6]. Additionally, AI-based tools for mesh generation streamline the process of mesh generation [7], effectively removing bottlenecks in the numerical modelling workflow. Furthermore, neural networks can predict the required time and computational resources in advance [8], or optimise GPU kernel scheduling during a run [9]. Moreover, AI has also been integrated with CFD models to enhance their performance, particularly in local flow pattern prediction for solution refinement [10], as well as in simulation speedup and residual monitoring [11].

Finally, AI has been employed to generate flow patterns as an alternative to CFD solvers. For instance, [12, 13, 14] use CNNs to predict flow around airfoils, whereas [15] address the same problem using a transformer-based attention mechanism applied to linearized patches of the computational domain's geometry.

So far, these flow-pattern generation approaches have only been applied to produce steady-state solutions. In this work, we propose FluidGPT-1, a proof-of-concept model designed to reproduce time-dependent solutions—that is, time series of snapshots of flow patterns. The underlying modelling approach is to employ a CNN-based method to encode flow-pattern snapshots into semantic representations and decode them back to flow patterns, alongside a transformer-based autoregressive

model that reproduces time evolution via the self-attention mechanism. This approach differs from [2] in that it produces general flow patterns rather than outputs tailored to a specific problem, and from [15] it applies self-attention to snapshots at different times, addressing a broader scenario than the airfoil. The Navier–Stokes equations are fully deterministic, so in principle, knowledge of a single timestep would suffice to determine the next. However, this is only feasible when all details of rheology, the equation of state, and turbulent dissipation are fully known; even then, there remains a risk of errors being amplified in a positive-feedback loop as the number of simulated timesteps grows. By contrast, our self-attention approach across multiple timesteps enables the model to learn the effects of rheology, the equation of state, and turbulent dissipation implicitly, while also recovering from any errors at an individual timestep.
The training dataset is generated artificially through repeated CFD runs with random parameters. The Lattice-Boltzmann Method (LBM) [16] is used—specifically, OpenLB (www.openlb.net) [17]—because its simple and explicit updating mechanism allows fast generation of a large number of time series whilst retaining numerical accuracy.

2. Architecture of FluidGPT-1

A high-level scheme of the model is shown in Figure 1. FluidGPT-1 processes sequences of 2D snapshots, each representing the x and y components of the velocity field and pressure, over a 101×101 grid. The model integrates a Convolutional Neural Network (CNN)-based UNet architecture ("UNet_tri_decoupled") for spatial feature extraction with a Transformer-based autoregressive model ("Transformer") to capture temporal dependencies. Specifically, the model comprises:

- *A UNet-based encoder-decoder.* The UNet_tri_decoupled's encoder converts each snapshot into a tuple consisting of a high-level semantic embedding vector, and a set of three intermediate feature maps. The decoder performs the reverse operation.
- *A Transformer.* The Transformer applies self-attention to a sequence of embeddings produced by the UNet_tri_decoupled's encoder, and generates the next embedding of the sequence.

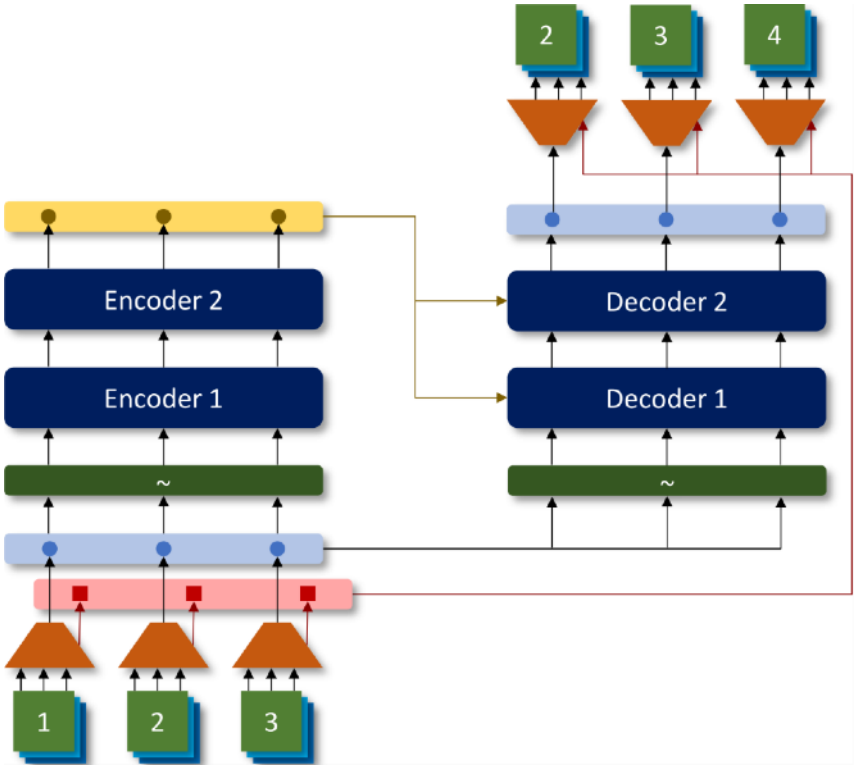

Figure 1: High-level representation of FluidGPT-1's architecture.

The model uses ELU activation functions. The snapshot sequence is processed as follows:

1. Each snapshot is converted into a sequence of semantic embeddings by the UNet_tri_decoupled's encoder.

2. The sequence of embeddings is processed by the Transformer's encoder into a context-aware latent representation of the sequence.
3. The latent representation is then passed to the Transformer's decoder, which predicts the next embedding of the sequence.
4. The updated sequence of embeddings is then converted into a sequence of snapshots by the UNet_tri_decoupled's encoder. The intermediate feature maps are passed directly to the decoder at the corresponding position in the sequence, shifted forward by one timestep.

2.1. Architecture of the UNet_tri_decoupled

A scheme of the UNet_tri_decoupled model is shown in Figure 2. The model employs a UNet-based architecture to encode a snapshot of pressure and velocity patterns (x and y component) into a semantic embedding vector, and to decode an embedding back into a flow and pressure pattern snapshot. A set of intermediate feature maps are generated as well, in order to facilitate the reconstruction of low-level local details. Unlike the original U-Net architecture, pressure and each component of the velocity field are processed separately through dedicated pathways. This design choice arises because these fields differ significantly—unlike the RGB channels in the original U-Net application—so combining them in a single pathway would lead to destructive interference. Following the encoder's downsampling, the pathway embeddings are concatenated and passed through a fully connected layer, enabling cross-pathway feature exchange. Symmetrically, in the decoder, the combined embedding is again processed by a fully connected layer before being split and upsampled across the dedicated pathways.

The architecture of the UNet_tri_decoupled's encoder is as follows:

1. The encoder accepts an input tensor of shape (batch_size, 101, 101, 3), where the last dimension corresponds to the three channels: p, ux, and uy.

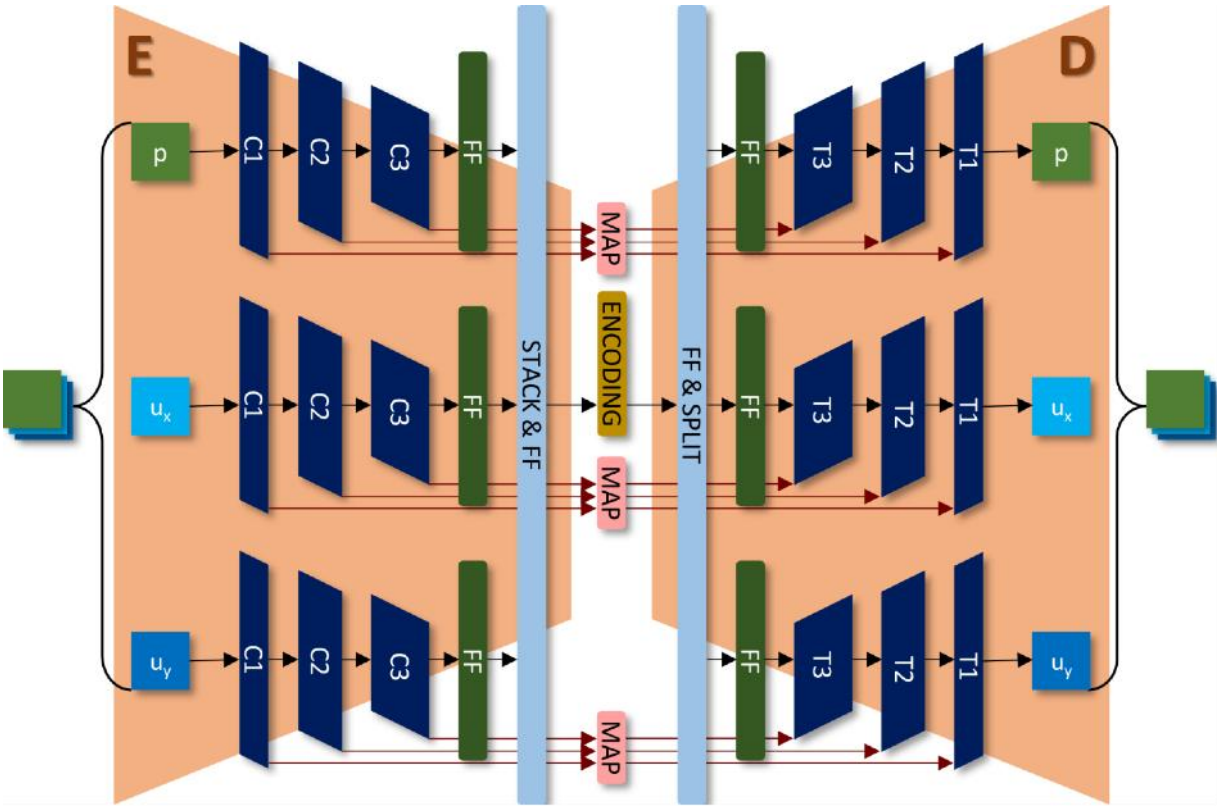

Figure 2: Scheme of the Unet_tri_decoupled model. E stands for encoder, D for decoder, C for convolutional layer, T for transposed convolutional layer, FF for feed-forward layer.

2. The input is split across the last dimension into a tensor of shape (batch_size, 101, 101, 1), thereby separating, p, ux and uy form one another, and each of them is fed into a dedicated downsampling pathway.

3. Each downsampling pathway consist of three convolutional layers with 3×3 kernels, strides of 2 and respectively 8, 16, and 32 filters. This arrangement creates intermediate feature maps of shapes (batch_size,51, 51, 8), (batch_size, 26, 26, 16) and (batch_size, 13, 13, 32).
4. The deepest feature map is then flattened in a vector of shape (batch_size, 5408) and passed to a fully-connected layer. The final pathway embedding has shape (batch_size, 256).
5. The three pathway embeddings are concatenated and passed to a fully-connected layer that maintains the same shape. The final embedding has shape (batch_size, 768).

The architecture of the UNet_tri_decoupled's decoder mirrors the encoder and is as follows:

1. The embedding is passed to a fully-connected layer that maintains the same shape, and then split into three pathway embeddings, each one of shape (batch_size, 256).
2. Each pathway embedding is passed to a fully-connected layer which outputs a tensor of shape (batch_size, 5408), and then reshaped into a tensor of shape (batch_size, 13, 13, 32).
3. Each tensor shaped in this way is processed through a dedicated upsampling pathway consisting of three transposed convolutional layers with 3×3 kernels, strides of 2, respectively 8, 16, and 32 filters and (0, 0) output paddings to enforce the correct dimension where appropriate. This arrangement creates intermediate tensors of shapes (batch_size, 26, 26, 16), (batch_size, 51, 51, 8) and (batch_size, 101, 101, 1). Each intermediate tensor is concatenated with the corresponding feature map before being passed to the transposed convolutional layer.
4. The tensors produced by the upsampling pathways are finally concatenated into the final output of shape (batch_size, 101, 101, 3).

2.2. Architecture of the transformer

The Transformer is a standard autoregressive transformer that applies self-attention to a sequence of embeddings produced by Unet_tri_decoupled's encoder, in order to predict the next embedding of the sequence.

The Transformer comprises an encoder and a decoder. The encoder is structured as follows:

1. The encoder accepts a sequence of n_seq embeddings (with n_seq limited to 19), arranged in a tensor of shape (batch_size, n_seq, 768).
2. A standard positional encoding is added to the input.
3. The input is passed through two identical standard encoder blocks, each featuring four attention heads. A dropout rate of 0.1 is applied to their dropout layers. The first fully connected layer contains 1,536 perceptrons, while the second reduces the embedding dimension back to 768 and does not use an activation function.
4. The final output of the encoder is a context-aware latent representation of the sequence.

The decoder is structured as follows:

1. The decoder accepts an input sequence with the same characteristics as the encoder's input.
2. A standard positional encoding is added to the input.
3. The input is passed through two identical standard decoder blocks, again each featuring four attention heads. A dropout rate of 0.1 is applied to their dropout layers. The first fully connected layer contains 1,536 perceptrons, while the second reduces the embedding dimension back to 768 and does not use an activation function. In each block, the context-aware latent produced by the encoder is passed to the second multi-head attention layer as key/value.
4. The final output of the decoder is a sequence of embeddings of the same type and size as the input, but shifted by one timestep forward for autoregressive prediction. The first n_seq – 1 embeddings align with the last n_seq – 1 embeddings of the input in order to allow a teacher-forcing training strategy, whereas the last embedding of the last sequence is newly-generated.

3. Training strategy

Training follows a teacher-forcing approach. Each example is a sequence of n_seq + 1 = 20 snapshots: the first n_seq are used as inputs, and the remaining n_seq provide the ground truth. The dataset is split into training and validation sets in a 0.9 ratio, then further divided into batches of 128 examples. An Adam optimizer is used. A learning rate scheduler starts from a value of 0.001, and then reduces it by a factor of 0.5 after five consecutive epochs of no improvement (on plateau) of the validation loss, down to a minimum of 1e-6. Early-stopping monitors the validation loss with patience of five. The

maximum number of epochs is set to 2,000. All model weights are initialized using the HeNormal scheme.

3.1. Loss function

The loss function is a linear combination of the mean squared error over all the output components and the cosine distance (defined as 1 - cosine similarity) over the ux and uy components, each weighted by 0.5. The reason for introducing the cosine distance is that the mean squared error alone would allow reconstructing the magnitude of the velocity field, but would lose information about its direction.

3.2. Metrics

Three metric functions are used: mean squared error over pressure alone, mean square error over the ux and uy components, and cosine distance over the ux and uy components.

4. Generation of the dataset

The dataset consists of 4,081 sequences of 20 2D snapshots each, artificially generated through a standard Smagorinsky-LES BGK Lattice-Boltzmann model implemented through OpenLB, an open-source library for Lattice-Boltzmann modelling. Under the assumption of small Mach and Knudsen numbers, the Lattice-Boltzmann Method reproduces a fluid obeying the Navier-Stokes equations with a compressibility error proportional to Ma^2, where Ma is the Mach number, and the equation of state $p = \rho c_s^2$ where $c_s \equiv \frac{1}{\sqrt{3}} \frac{\delta x}{\delta t}$ is the tuneable speed of sound, which depends on the lattice size, x and the timestep, t. It has been shown that, under the assumption of low Mach number, the speed of sound (and therefore, the Mach number itself) can be altered and used as a tuneable parameter in order to improve the balance between stability, accuracy and numerical expense, without altering the system significantly [16, 18].

The computational domain is a 2D square grid of 101 × 101 lattice sites, containing randomly generated inlet, outlet, and multiple impenetrable circles within its interior. The inlet is placed on one of the four sides of the domain sampled uniformly, while the outlet is placed on one of the remaining three sides, also sampled uniformly. For both the inlet and outlet, the size of the opening is sampled uniformly between 0.3 and 1.0 times the domain side length. The number of impenetrable internal circles is sampled uniformly between 1 and 6. For each of them, position and radius are randomly generated in such a way that the minimum radius and the minimum distance from the borders (including the other circles) are 0.05 and times the domain side length, respectively. The inlet is defined as a fixed-velocity boundary following a Poiseuille profile; the outlet as a fixed-pressure; the external boundaries and the boundaries of the circle are respectively defined as bounce-back and Bouzidi boundaries (both equivalent to second-ordered no-slip).

The Smagorinsky constant and the lattice velocity are set to 0.14 and 0.1 respectively. The Reynolds number is evaluated using domain side length and average inlet velocity, and is generated according to a lognormal distribution with mean value 90 and standard deviation 118. The physical quantities of domain side length and reference average inlet velocity are conventionally set to 1 m and 0.5 m/s respectively. All the other parameters are tuned accordingly, following standard definitions and laws of similarity [16].

Pressure and flow patterns are stored every 80 timesteps and a total of 20 snapshots are stored. Each run is uniformly randomly chosen to follow either an “increasing”, or a “constant” simulation. In the “increasing” simulation, the inlet average velocity is ramped linearly from zero to the reference value throughout the simulation. In the “constant” simulation, the average inlet velocity is preliminarily ramped up to the reference value and then kept constant; the timesteps are stored after the initial plateau.

5. Results

Training occurs for 571 epochs before the early stopping triggers its termination. The numerical job is performed on an NVIDIA Volta V100-SXM2-16GB GPU card for around five hours. Figures 3 and 4 show respectively loss and metrics as functions of the epoch. The model converges to training and a validation losses of respectively 0.0916 and 0.0905. The validation trends closely follow the training

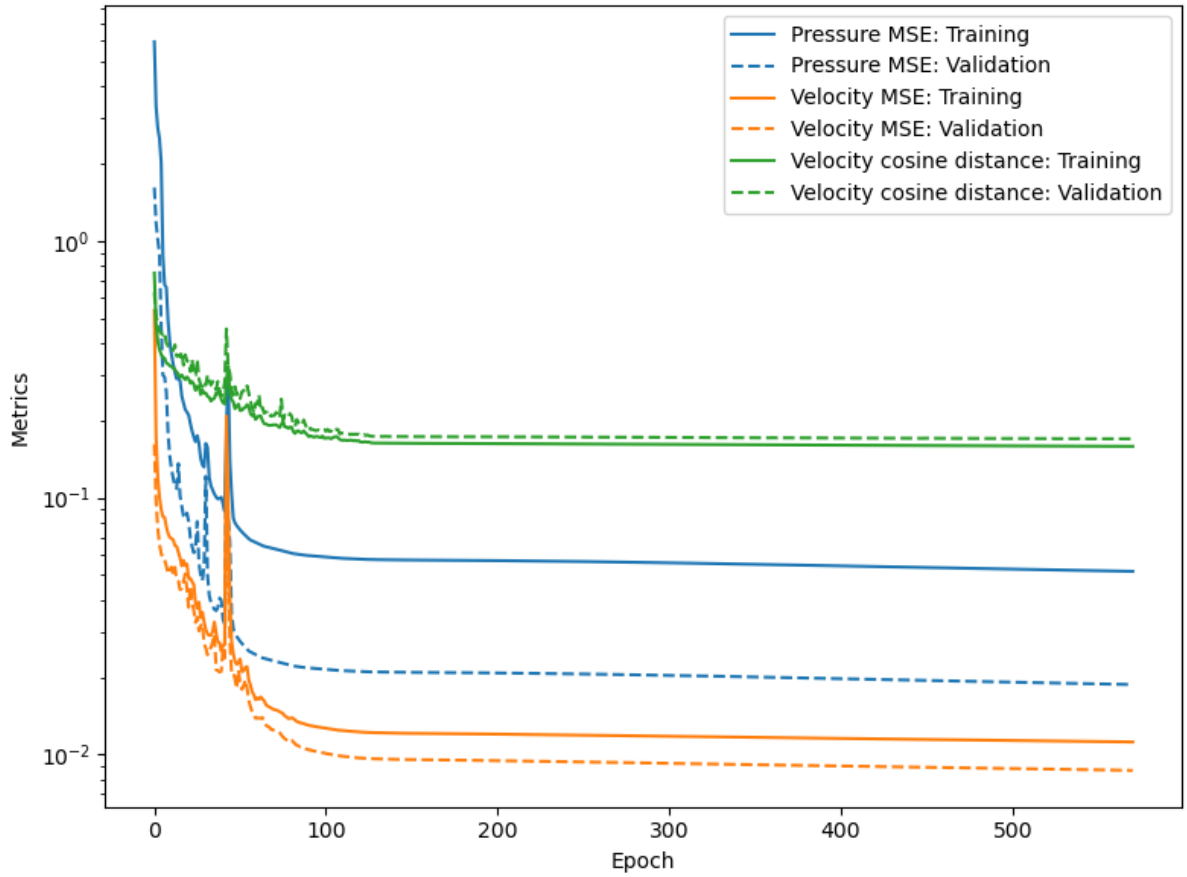

Figure 4: Training and validation metrics.

with no sign of underfitting or overfitting, indicating that the model generalizes well and that both model and dataset sizes are adequate. The training process presents spikes in the loss at around the fiftieth epoch; the process presented here quickly recovers and converges successfully, but in a small number of cases, the training process does not recover and the early-stopping callback triggers an untimely termination. The losses decrease at negligible rates after around 120 epochs, indicating that the model has reached its maximum precision achievable with its current architecture.

Figure 5 shows the flow patterns of a random "constant" sequence. The model, in general, predicts the ground truth well. Discrepancies consist mainly of diffusion noise around the borders of the internal circles, and repeated convolutional patterns. This evidences possible limits in the UNet architecture employed in this model.

Figures 6 to 11 show the flow patterns of two random "increasing" sequences, taken at timesteps 6, 12 and 18 (wherein the first timestep of the sequence is labelled with a 0). The same observations for the "constant" case apply, with the difference that the relative intensity of the discrepancies is more pronounced in absolute value. This evidences possible limits in the current approach of passing the feature maps from the UNet encoder directly to the next timestep's UNet decoder.

6. Conclusions

FluidGPT-1, a proof-of-concept generative model to predict 2D flow patterns, is presented. The model successfully predicts variable pressure and velocity fields over time. The limitations of the current approach are discussed. Future improvements will involve the implementation of encoders-decoders for encodings with architectures more advanced than the UNet, and the deployment of strategies to elaborate the feature maps before passing them to the next timestep's decoders. In our opinion, FluidGPT-1 demonstrates that it is possible to develop generative models for flow pattern prediction.

7. Acknowledgments

The authors would like to thank the University of Liverpool's High-Performance Computing (HPC) services, where the computational work was carried out. The authors used Microsoft Word to assemble the manuscript, JabRef for referencing management and GPT language models (ChatGPT and Perplexity) for text refinement, grammar check, debugging and reference/literature mining.

8. References

[1] J. Suk, P. d. Haan, P. Lippe, C. Brune and J. M. Wolterink, "Mesh Convolutional Neural Networks for Wall Shear Stress Estimation in 3D Artery Models," in *Statistical Atlases and Computational Models of the Heart. Multi-Disease, Multi-View, and Multi-Center Right Ventricular Segmentation in Cardiac MRI Challenge*, Cham, 2022.

[2] S. D. Yang, Z. A. Ali and B. M. Wong, "FLUID-GPT (Fast Learning to Understand and Investigate Dynamics with a Generative Pre-Trained Transformer): Efficient Predictions of Particle Trajectories and Erosion," *Industrial & Engineering Chemistry Research,* vol. 62, pp. 15278-15289, 2023.

[3] X. Sun, W. Cao, Y. Liu, L. Zhu and W. Zhang, "High Reynolds number airfoil turbulence modeling method based on machine learning technique," *Computers & Fluids,* vol. 236, p. 105298, 2022.

[4] M. R. Alhafiz, P. S. Palar and Z. Lavi, "An Efficient Data-Driven Neural Network Turbulence Model for Flow over Periodic Hills and Parametric Bump," in *AIAA SCITECH 2024 Forum*.

[5] G. Calzolari and W. Liu, "Deep learning to develop zero-equation based turbulence model for CFD simulations of the built environment," *Building Simulation,* vol. 17, pp. 399-414, 01 March 2024.

[6] X. Fan, D. Akhare and J.-X. Wang, "Neural differentiable modeling with diffusion-based super-resolution for two-dimensional spatiotemporal turbulence," *Computer Methods in Applied Mechanics and Engineering,* vol. 433, p. 117478, 2025.

[7] C. Lorsung and A. Barati Farimani, "Mesh deep Q network: A deep reinforcement learning framework for improving meshes in computational fluid dynamics," *AIP Advances,* vol. 13, p. 015026, January 2023.

[8] G. Rubio, E. Valero and S. Lanzan, "Computational Fluid Dynamics Expert System using Artificial Neural Networks," *International Journal of Engineering and Applied Sciences,* vol. 6, p. 413–417, 26 March 2012.

[9] W. Xue and C. J. Roy, *Machine Learning based Autotuning of a GPU-accelerated Computational Fluid Dynamics Code,* 2023.

[10] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner and S. Hoyer, "Machine learning–accelerated computational fluid dynamics," *Proceedings of the National Academy of Sciences,* vol. 118, p. e2101784118, 2021.

[11] J. Jeon, J. Lee, R. Vinuesa and S. J. 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,* vol. 220, p. 124900, 2024.

[12] X. Guo, W. Li and F. Iorio, "Convolutional Neural Networks for Steady Flow Approximation," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2016.

[13] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy and S. Kaushik, "Prediction of aerodynamic flow fields using convolutional neural networks," *Computational Mechanics,* vol. 64, pp. 525-545, 01 August 2019.

[14] M.-Y. Wu, Y. Wu, X.-Y. Yuan, Z.-H. Chen, W.-T. Wu and N. Aubry, "Fast Prediction of Flow Field around Airfoils Based on Deep Convolutional Neural Network," *Applied Sciences,* vol. 12, 2022.

[15] K. Zuo, Z. Ye, W. Zhang, X. Yuan and L. Zhu, "Fast aerodynamics prediction of laminar airfoils based on deep attention network," *Physics of Fluids,* vol. 35, p. 037127, March 2023.

[16] T. Krueger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva and E. M. Viggen, The Lattice Boltzmann Method: Principles and Practice, Springer, 2016.

[17] M. J. Krause, A. Kummerländer, S. J. Avis, H. Kusumaatmaja, D. Dapelo, F. Klemens, M. Gaedtke, N. Hafen, A. Mink, R. Trunk, J. E. Marquardt, M.-L. Maier, M. Haussmann and S. Simonis, "OpenLB—Open source lattice Boltzmann code," *Computers & Mathematics with Applications,* vol. 81, pp. 258-288, 2021.

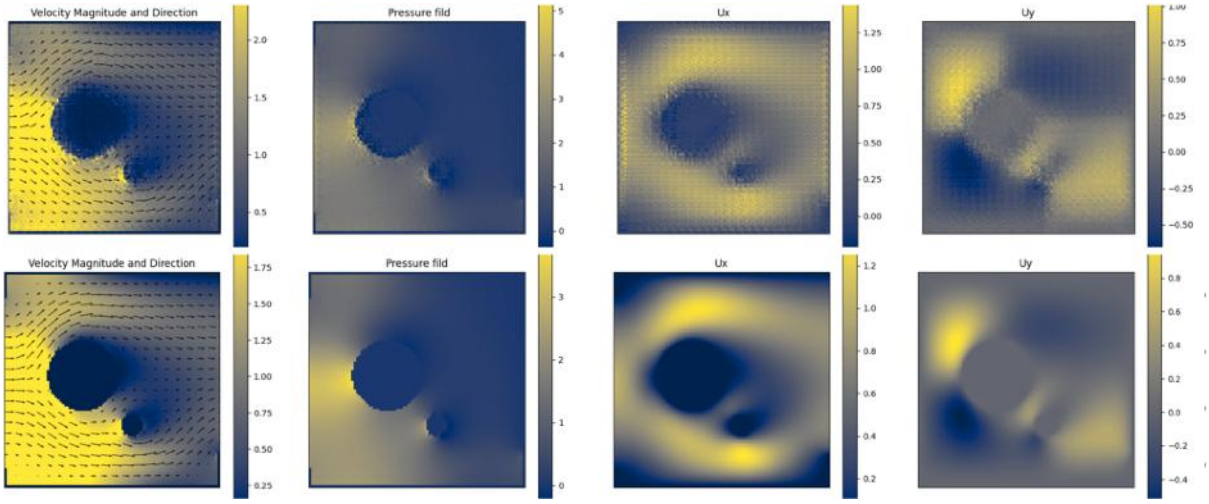

Figure 5:"Constant" sequence. Top: predicted patterns. Bottom: ground truth. From the left to the right column: velocity magnitude, pressure, velocity's x and y components.

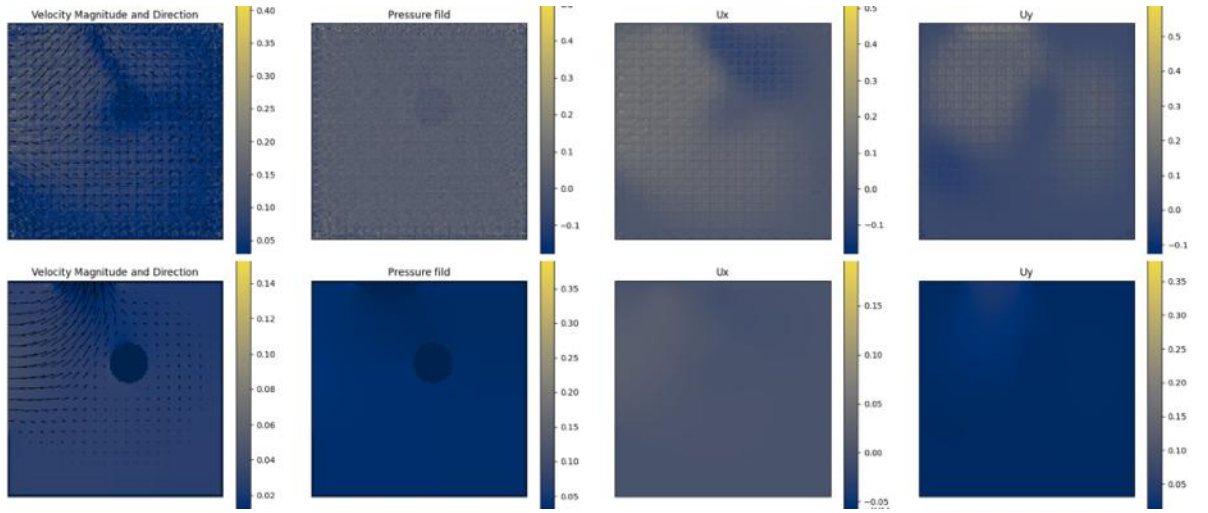

Figure 6: "Increasing" sequence, example 1, snapshot 6. Top: predicted patterns. Bottom: ground truth. From the left to the right column: velocity magnitude, pressure, velocity's x and y components.

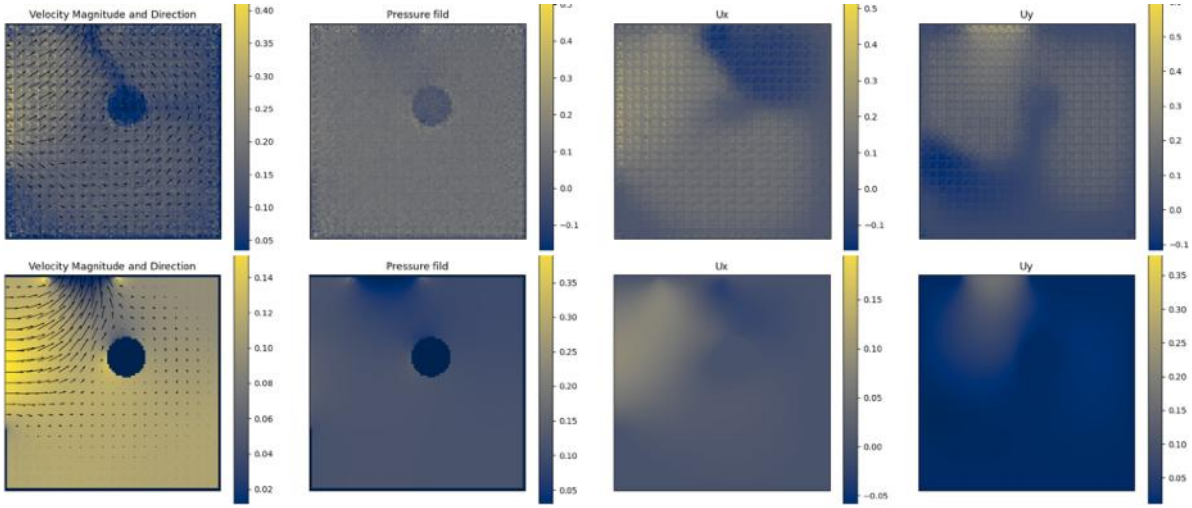

Figure 7: "Increasing" sequence, example 1, snapshot 12. Top: predicted patterns. Bottom: ground truth. From the left to the right column: velocity magnitude, pressure, velocity's x and y components.

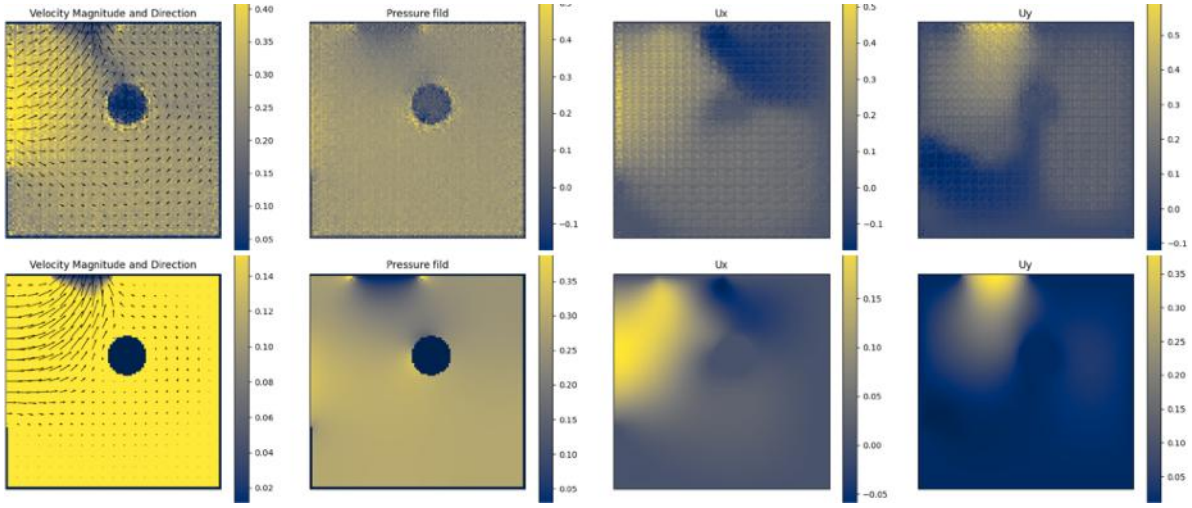

Figure 8 "Increasing" sequence, example 1, snapshot 18. Top: predicted patterns. Bottom: ground truth. From the left to the right column: velocity magnitude, pressure, velocity's x and y components.

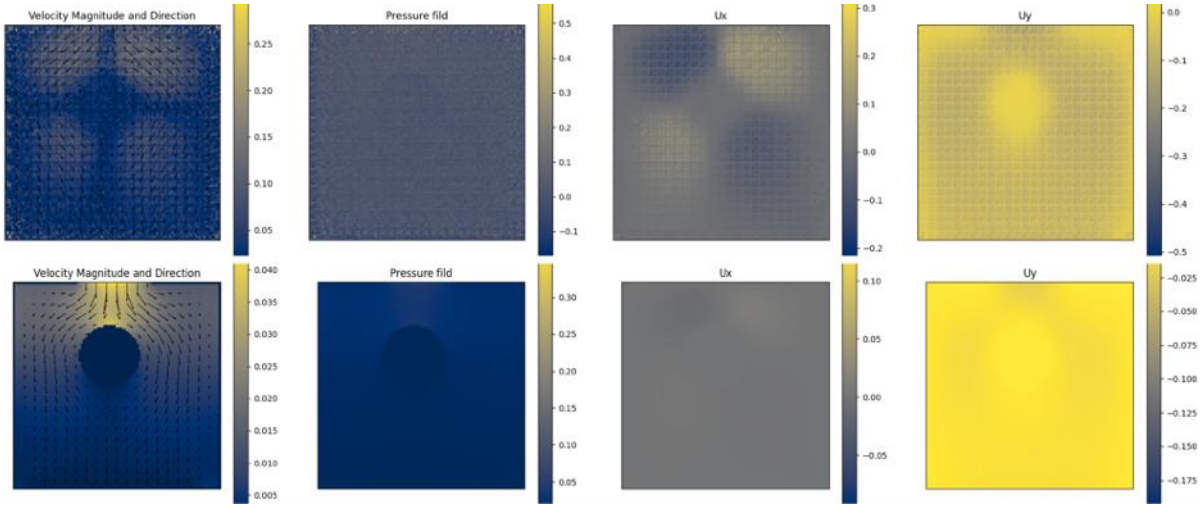

Figure 9: "Increasing" sequence, example 2, snapshot 6. Top: predicted patterns. Bottom: ground truth. From the left to the right column: velocity magnitude, pressure, velocity's x and y components.

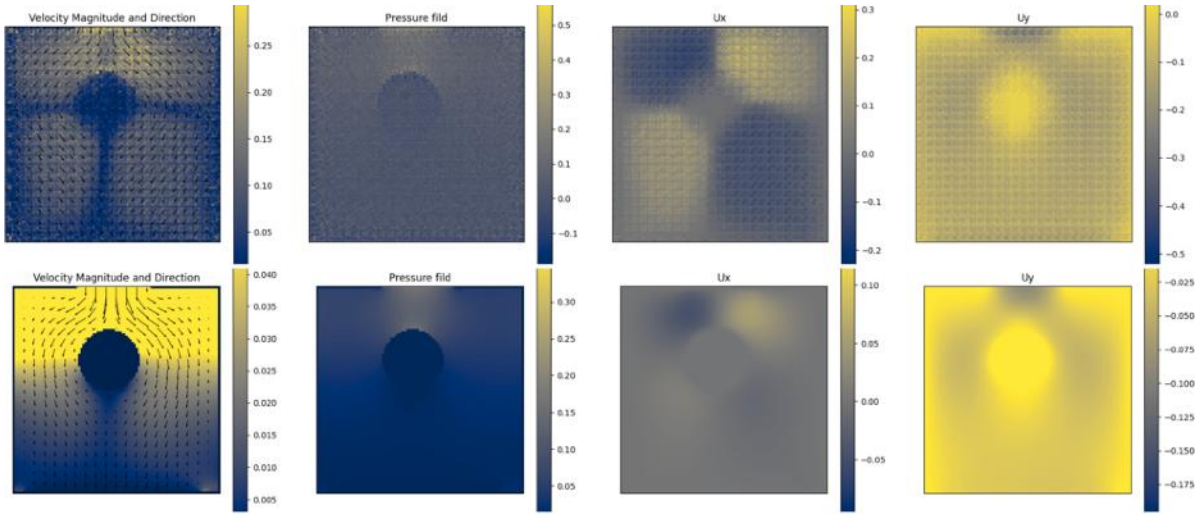

Figure 10: "Increasing" sequence, example 2, snapshot 12. Top: predicted patterns. Bottom: ground truth. From the left to the right column: velocity magnitude, pressure, velocity's x and y components.

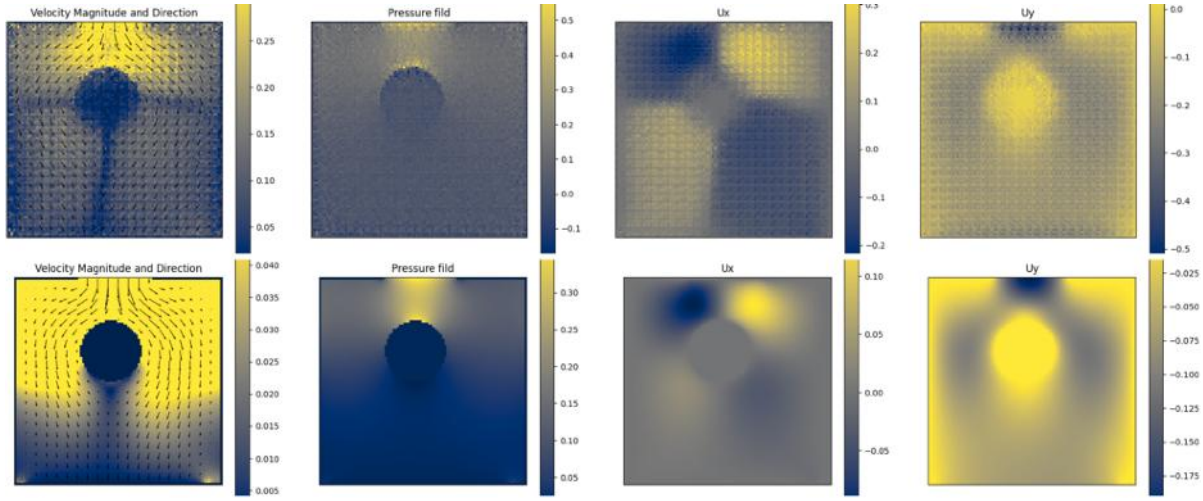

Figure 11: "Increasing" sequence, example 2, snapshot 18. Top: predicted patterns. Bottom: ground truth. From the left to the right column: velocity magnitude, pressure, velocity's x and y components.