Embedding temporal error propagation on CNN for unsteady flow simulations

Ekhi Ajuria Illarramendi
ISAE-SUPAERO / CERFACS
Université de Toulouse, France
ekhi.ajuria@cerfacs.fr

Michaël Bauerheim
ISAE-SUPAERO
Université de Toulouse, France
michael.bauerheim@isae-supaero.fr

Bénédicte Cuenot
CERFACS
23 av Coriolis, Toulouse
cuenot@cerfacs.fr

Antonio Gimenez Nadal
ISAE SUPAERO
Université de Toulouse, France
antonio.gimenez-nadal@student.isae-supaero.fr

Abstract

This work investigates the interaction of a CNN-based Poisson solver with an incompressible fluid solver for unsteady flow simulations. During training, the network prediction is used to continue in time the computation, embedding the influence of the network prediction on the simulation using a long-term loss. This study investigates three implementations of such a loss, as well as the number of look-ahead iterations. On all test cases, results show that long-term losses are always beneficial. Interestingly, a partial implementation without differentiable solver is found accurate, robust and less costly than full implementation.

1 Introduction

Simulating incompressible flows is vital for engineering purposes, although performing accurate simulations usually comes at a high computational cost [1]. The recent rise of machine learning (ML) has introduced a powerful tool to the CFD community, which can be used as surrogate model to model fluid flows [2, 3].

However, neural networks do not guarantee respecting the underlying physics of the simulated problem, thus different strategies have been developed to embed *physical knowledge* to the networks. The residual of the governing physical equations has been introduced as loss functions [4, 5] outperforming their image-to-image counterparts. The differentiable nature of the network has also been used to accurately compute the residual of the studied PDE [6] with the introduction of physics-informed neural networks (PINN). Moreover, the development of differentiable fluid solvers [7] has enabled to backpropagate errors through entire fluid simulations, making it possible to directly encode the nonlinear fluid solver behavior into the network.

This work follows the study of Tompson et al. [4], where a CNN solves the Poisson equation for incompressible flows, and the propagation of the network error through the simulation is embedded during training using a *long-term loss* (LTL). The differentiable *phiflow* [7] (`https://github.com/tum-pbs/PhiFlow`) fluid solver is used, to assess the benefit of computing the intermediate gradients, *full long-term loss* (F-LTL), as well as the influence of the number of *look-ahead iterations* (LAI). Since F-LTL is computationally expensive and requires a large amount of memory, F-LTL is compared with a *partially frozen long-term loss* (PF-LTL) for which no differentiable solver is required. The benefits of the introduction of intermediate gradients is still an open question

Fourth Workshop on Machine Learning and the Physical Sciences (NeurIPS 2021).

with crucial fundamental and practical implications, which is studied here by comparing accuracy, robustness, and computational cost of F-LTL and PF-LTL[1].

2 Poisson equation on incompressible fluid solvers

This work focuses on the resolution of the 2D incompressible *Navier-Stokes* equations. They are solved with a standard operation splitting method, which divides the solving procedure into two main steps: (i) the advection and (ii) the pressure projection [8]. The advection step generates a non divergence-free velocity $\boldsymbol{u}^*$ which needs to be updated in the pressure projection step. To update the velocity, a simple forward Euler is used, which after taking the divergence of both sides results in the Poisson equation

$$\frac{\Delta t}{\rho_0} \nabla^2 p = \nabla \cdot \boldsymbol{u}^* \tag{1}$$

where p is the pressure field, ρ_0 the background density and Δt the simulation timestep. The resolution of this equation can take up to 80% of the computational time [3], and it is usually solved with iterative solvers, such as the Conjugate Gradient (CG). A thorough comparison of the network inference ran on GPU cards, compared to traditional CPU linear solvers can be found in the work of Cheng et al. [9]. Following the work of Tompson et al. [4], this work substitutes the costly resolution of this equation with a CNN.

3 Training procedure

A Unet neural network [10] with 5 down-sampled scales and 400 000 parameters [3] is chosen as architecture. The network is trained on a set of 320 simulations with 64 time steps (i.e. 20480 snapshots) computed with a CG solver. These simulation domains have 128x128 cells, including random geometries and divergence sources, with no buoyancy-driven forces and no viscosity.

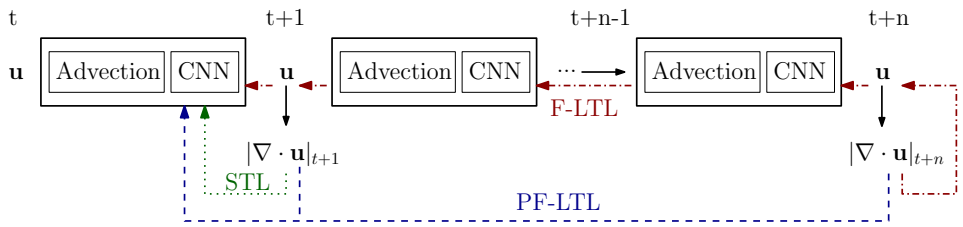

Figure 1: Training configurations: STL (green dotted line), PF-LTL (blue dashed line) and F-LTL (red dash-dotted line).

The training strategy is depicted in Fig. 1. As first introduced by Tompson et al. [4], the corrected velocity field needs to respect the incompressibility condition ($\nabla \cdot \boldsymbol{u} = 0$), which can be used as a loss function to train the network. If only the divergence of the corrected velocity field is taken, without further modifications, the training loss will be known as *Short-term loss* (STL), corresponding to the green dotted line in Fig. 1. However, the corrected velocity field $\boldsymbol{u}_{t+1}$ can then be advected n times, propagating the error made by the network through the fluid simulation. The contribution of the divergence of the velocity $\boldsymbol{u}_{t+n}$ to the global loss $\mathcal{L}$ is denoted *long-term loss* (LTL).The choice of the number of LAI (n) remains an open question that this work tries to answer. Longer simulations can incorporate more information, although it can de-stabilize trainings and increase the computational cost of the training. Thus, the training loss can be written as

$$\mathcal{L} = \alpha \|\nabla \cdot \mathbf{u_{t+1}}\|_2 + \beta \|\nabla \cdot \mathbf{u_{t+n}}\|_2 \tag{2}$$

where α and β correspond to user-defined weighting parameters. The particular case where $\beta = 0$ corresponds to the STL training. When computing the error made in the timestep n, the error can be backpropagated over all the solver steps (F-LTL), including all advection and Poisson solvers (red dash-dotted line in Fig. 1), thus requiring a differentiable solver (here *phiflow* [7]) and large

[1]The code used to reproduce the results of this paper is publically available at `https://gitlab.isae-supaero.fr/daep/neurasim`

computational and memory requirements. To avoid these constraints, the PF-LTL strategy [4, 3] proposes to *freeze* the network parameters through the simulations to skip the advection blocks during backpropagation, so that only $\boldsymbol{u}_{t+1}$ and $\boldsymbol{u}_{t+n}$ are used (blue dashed line in Fig. 1).

Thus, the Unet network is trained with the three mentioned strategies (STL, PF-LTL and F-LTL), where several different numbers of LAI are tested to check the influence of longer simulations on the network training. This number comes in a tuple, since during the training process the first number is performed 90% of the time, and the second the remaining 10%. For the loss function hyperparameters, α is set to 1 and β to 5 for the PF-LTL and F-LTL cases, as increasing the relative weight of the LTL helped to balance the network training. All the networks are trained with the Adam optimizer (with an initial learning rate set to $5 \cdot 10^{-5}$). All trainings and inference tests were performed in 16 Gb Nvidia Tesla V-100 GPU cards. The objective of this work is to determine, on 3 test cases, the actual accuracy and requirements of these 3 strategies, and to establish guidelines for future practical implementations of such CNN coupled with unsteady solvers.

4 Test Cases

The neural networks are tested in three flow configurations, depicted in Fig. 2 that significantly differ from the training cases (same as in [3, 4]). The objective is to evaluate the accuracy but also the robustness of the 3 strategies, as encountered in a real application of such a solver. First, as the training dataset is inviscid, the flow around a cylinder (Fig. 2-a) at Reynolds 100 is tested. Evaluation is also performed on rotating flows. To do so, a rotating velocity is imposed in the previous case, parametrized by the dimensionless rotating velocity $\alpha = \omega D/(2U_\infty)$, where ω is the counter-clockwise rotating velocity of the cylinder, D the obstacle diameter and U_∞ the input velocity. The flow is tested at α = 0 (no rotation), 1 and 1.5 and the accuracy is evaluated comparing the relative difference of the amplitude of the dimensionless lift coefficients $A_l = max(|C_l|) - min(|C_l|)$, between the neural network prediction and the result computed with the reference CG solver: $\mathcal{E}_{VK} = |A_l^{CG} - A_l^{NN}|/A_l^{CG}$.

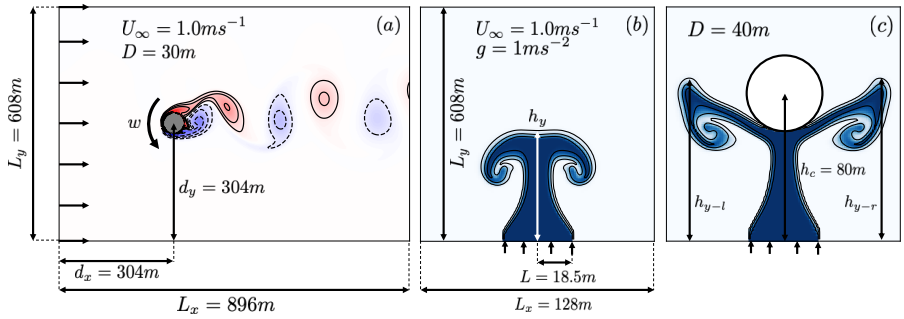

Figure 2: Test configurations: a rotating cylinder (a) and a plume without (b) and with (c) obstacle.

Then, as the training dataset does not have buoyancy-driven forces, the networks are tested on a buoyant plume, flows where a lighter fluid is injected into a quiescent environment. Plumes without obstacle (Fig. 2-b) and with a cylinder (Fig. 2-c) are tested. Buoyant plumes are parametrized with the dimensionless Richardson number, $Ri = (\Delta\rho/\rho_0)Lg/U_\infty^2$, where $\Delta\rho/\rho_0$ is the density difference between the injected fluid and the quiescent environment, L the inflow radius, g gravity and U_∞ the injection velocity. This parameter characterizes the ratio between buoyancy-driven forces and momentum-driven forces. For both the non-cylinder and cylinder configurations, Ri = 0.1, 1 and 10 are tested. For the non cylinder configuration (referred to as plume) the accuracy is evaluated comparing the relative difference of the plume head position h_y between the studied network and the simulation computed with a reference CG solver: $\mathcal{E}_{Plume} = |h_y^{CG} - h_y^{NN}|/L_y$ at the moment where the reference plume reaches the 70% of the domain. The metric used for the cylinder configuration (referred as Cyl. Plume) penalizes the loss of symmetry between the left and right branches. Thus, the average of the left and right plume heads (h_{y-l} and h_{y-r}) are compared to the simulation performed with the CG solver, when the plume head reaches 80% of the domain height: $\mathcal{E}_{Cyl-plume} = (1 + \Delta h_y^{NN})|h_{y-avg}^{CG} - h_{y-avg}^{NN}|/L_y$, where $\Delta h_y^{NN} = |h_{y-r}^{NN} - h_{y-l}^{NN}|$ and $h_{avg-y}^{NN} = 1/2(h_{y-l}^{NN} + h_{y-r}^{NN}|)$. Note that trainings are performed at $\alpha = 0$, $Re = \infty$ and $Ri = 0$.

Table 1: $\mathcal{E}_{VK-l}$, $\mathcal{E}_{Plume}$ and $\mathcal{E}_{Cyl-plume}$ in % for the STL, F-LTL and PF-LTL trainings.

		Rotating cyl. (α)			Plume (Ri)			Cyl. plume (Ri)		
	LAI	**0**	**1**	**1.5**	**0.1**	**1**	**10**	**0.1**	**1**	**10**
STL	0	14.9	44.3	35.8	11.7	11.7	8.6	11.7	13.7	289.5
F-LTL	1-2	2.7	3.6	2.3	**7.8**	**7.0**	**10.9**	3.9	**7.4**	**64.4**
	2-4	**2.0**	**1.0**	**0.7**	8.6	**7.0**	**10.9**	**3.1**	15.6	107.4
	2-6	3.6	3.3	4.7	**7.8**	**7.0**	**10.9**	5.5	17.2	84.4
PF-LTL	2-4	**2.6**	6.0	**1.9**	8.6	7.0	**10.1**	5.5	15.6	84.4
	4-8	5.2	3.4	2.4	10.2	**5.5**	10.9	5.5	9.0	**64.4**
	4-16	3.7	**2.2**	3.2	**7.8**	6.2	10.9	**3.9**	**8.2**	84.4

5 Results

Results for the 3 strategies are found in Table 1 and Fig. 3. First, the benefit of adding the LTL to the training procedure (either F-LTL or PF-LTL) is highlighted, as results are almost one order of magnitude better in the rotating cylinder case, and almost two times better in the plume. Comparing the F-LTL and PF-LTL trainings, the results are rather similar, with small differences between the studied networks. The F-LTL networks slightly outperform the PF-LTL networks in the flow around the cylinder and at plumes with low Richardson numbers. However, the PF-LTL networks better handle higher Richardson numbers, hinting a possible slight *over-fitting* for the F-LTL networks for which backpropagation through advection (without buoyancy and viscosity) has been performed. This effect is especially highlighted on the Cylinder plume test case, as small losses in symmetry considerably penalize the F-LTL trainings. The Cylinder plume test case highlights the difficulty of encoding complex flows, which could be tackled introducing a hybrid strategy combining classical linear solvers with the network prediction, to ensure an user-defined accuracy level [11].

Regarding the number of LAI, its increase is beneficial for the PF-LTL networks, as on average the 4-16 iteration network shows the best overall behavior. Using PF-LTL networks, as no intermediate gradients were needed, the batch size could be increased up to 110 for all 3 studied networks. During training, the 4-8 and 4-16 networks took respectively an average of 280 s and 295 s per epoch (in a single GPU training), whereas the 2-4 network took an average of 170 s per epoch. However, for the F-LTL trainings, the 2-4 network is an optimal intermediate compared to the 1-2 and 2-6 networks. Too many LAI can cause vanishing gradients, making the network convergence during training more difficult. Moreover, the batch size needed to fill the GPU is reduced, resulting respectively in batch sizes of 46, 24 and 16, which lead to an average of 290, 490 and 1750 s per epoch (in a single GPU training). For practical applications (large datasets, etc.), F-LTL is difficult to train, as the intermediate gradients introduce longer backpropagation chains, more prone to vanishing/exploding gradient issues, with small benefits compared with PF-LTL in terms of accuracy. However, PF-LTL requires longer LAI, which only affects the training time, since memory usage does not depend on LAI for this strategy.

6 Conclusions

Training strategies that embed information of the unsteady flow simulation (LTL) have been studied. Results show that both F-LTL and PF-LTL outperform STL trainings. Moreover, the use of intermediate gradients during the training might not always be beneficial, as similar results can be obtained with PF-LTL trainings at smaller computational cost. F-LTL networks can be trickier to train due to the greater backpropagating gradient chains. Increasing the number of LAI iterations improves the accuracy of PF-LTL networks, while lower LAI are needed for F-LTL networks. The method could be extrapolated to 3D cases, where CNNs are expected to outperform classical linear solvers and perform *accurate* simulations. However, more computational power would be needed, especially to perform a F-LTL strategy for 3D cases. While limiting the computational resources during training, PF-LTL is expected to encode sufficient information to make *accurate* predictions while not considerably increasing the computational cost, even in such realistic 3D large simulations.

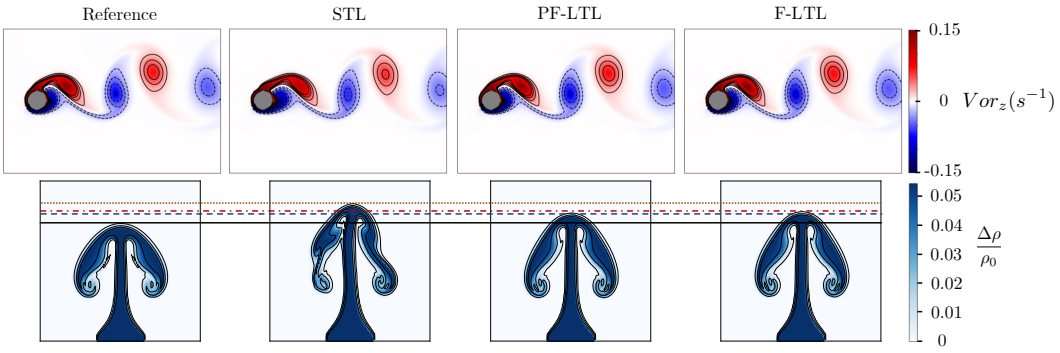

Figure 3: Snapshots of the reference (black line), STL (orange dotted line), PF-LTL 4-16 (blue dashed line) and F-LTL 2-4 (red dash-dotted line) for the flow around a rotating cylinder with $\alpha = 1.5$ (top) and plume simulation with Ri=1 (bottom).

References

[1] LE Jones, RD Sandberg, and ND Sandham. Direct numerical simulations of forced and unforced separation bubbles on an airfoil at incidence. *Journal of Fluid Mechanics*, 602:175–207, 2008.

[2] Steffen Wiewel, Moritz Becher, and Nils Thuerey. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer graphics forum*, volume 38, pages 71–82. Wiley Online Library, 2019.

[3] Ekhi Ajuria Illarramendi, Michaël Bauerheim, and Bénédicte Cuenot. Performance and accuracy assessments of an incompressible fluid solver coupled with a deep convolutional neural network. *arXiv e-prints*, pages arXiv–2109, 2021.

[4] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 3424–3433. JMLR.org, 2017.

[5] Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, pages 59–70. Wiley Online Library, 2019.

[6] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[7] Philipp Holl, Vladlen Koltun, Kiwon Um, Telecom Paris LTCI, IP Paris, and Nils Thuerey. phiflow: A differentiable pde solving framework for deep learning via physical simulations. In *NeurIPS Workshop*, 2020.

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

[9] Lionel Cheng, Ekhi Ajuria Illarramendi, Guillaume Bogopolsky, Michael Bauerheim, and Benedicte Cuenot. Using neural networks to solve the 2d poisson equation for electric field computation in plasma fluid simulations. *arXiv preprint arXiv:2109.13076*, 2021.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[11] 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*, page 3058, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No] This work does is not susceptible of having negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [No] This work does not include theoretical results
   (b) Did you include complete proofs of all theoretical results? [No] This work does not include theoretical results
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 1, `https://anonymous.4open.science/r/neurasim-D84B/`
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] The license agreement is specified in the repository of the code of this work
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See Section 3, `https://github.com/tum-pbs/PhiFlow`
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] The consent was not necessary under the asset's license.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] The used data does not contain any personal or offensive content.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [No] No crowdsourcing or human subjects were used for this study.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [No] No crowdsourcing or human subjects were used for this study.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [No] No crowdsourcing or human subjects were used for this study.