End-to-End Mesh Optimization of a Hybrid Deep Learning Black-Box PDE Solver

Shaocong Ma
University of Utah
s.ma@utah.edu

James Diffenderfer
Lawrence Livermore National Laboratory
diffenderfer2@llnl.gov

Bhavya Kailkhura
Lawrence Livermore National Laboratory
kailkhura1@llnl.gov

Yi Zhou
University of Utah
yi.zhou@utah.edu

Abstract

Deep learning has been widely applied to solve partial differential equations (PDEs) in computational fluid dynamics. Recent research proposed a PDE correction framework that leverages deep learning to correct the solution obtained by a PDE solver on a coarse mesh. However, end-to-end training of such a PDE correction model over both solver-dependent parameters such as mesh parameters and neural network parameters requires the PDE solver to support automatic differentiation through the iterative numerical process. Such a feature is not readily available in many existing solvers. In this study, we explore the feasibility of end-to-end training of a hybrid model with a black-box PDE solver and a deep learning model for fluid flow prediction. Specifically, we investigate a hybrid model that integrates a black-box PDE solver into a differentiable deep graph neural network. To train this model, we use a zeroth-order gradient estimator to differentiate the PDE solver via forward propagation. Experiments show that the proposed approach based on zeroth-order gradient estimation underperforms the baseline that computes exact derivatives using automatic differentiation, but outperforms the baseline trained with a frozen input mesh to the solver.

1 Introduction

Studying fluid dynamics presents significant challenges in the physical sciences due to the complex nature of solving Navier-Stokes partial differential equations (PDEs) [4]. Traditional numerical solvers employ finite-difference methods to solve PDEs over highly granular discrete meshes, demanding substantial computational burden. To address these challenges, data-driven deep learning techniques have been developed to directly predict physical outcomes in computational fluid dynamics [3, 8, 16, 15, 6]. However, these approaches often require a substantial volume of data. Consequently, the learned models often exhibit limited generalizability when solving PDEs with out-of-distribution physical parameters (e.g., angle of attack).

Recently, a promising approach for PDE correction has emerged to enhance the generalization performance [6]. This approach jointly optimizes a deep model and the PDE solver's input mesh to correct coarse simulation outcomes. However, this approach requires that the PDE solver supports automatic differentiation through the numerical solving process – an attribute that often demands substantial programming effort, especially for solvers in advanced application domains. On the other hand, observe that mesh optimization is considerably less complex than neural network parameter optimization, it is possible to optimize the mesh using noisy algorithms that do not rely on exact gradient queries. Inspired by these insights, we are motivated to study the following question.

Machine Learning and the Physical Sciences Workshop, NeurIPS 2023.

- *Q: Can we perform end-to-end deep learning for fluid flow prediction with black-box solvers that do not support automatic differentiation?*

In this work, we provide an affirmative answer to the above question. Our contributions are in two-fold. First, we develop a zero-order type algorithm that can optimize the deep model's parameters and solver's mesh parameters end-to-end without back-propagation through the solver. Consequently, one can integrate any black-box solvers into the system and apply our algorithm to train this system. Second, experiments show that the proposed zeroth-order approach produces correction models that outperform the baseline model trained using first-order method with a frozen input mesh to the solver.

2 Hybrid Model for Fluid Flow Prediction

2.1 The SU2 PDE Solver

SU2 is a numerical solver developed for solving PDEs that arise in computation fluid dynamics [7]. It applies the finite volume method (FVM) to solve PDEs and calculate certain physical quantities over a given discrete mesh. To explain how SU2 works, consider the task of calculating the airflow fields around an airfoil. The input to SU2 includes the following three components: (1) *Mesh*: a discrete approximation of the continuous flow field; (2) *Angle of attack (AoA)*: the angle between the chord line of the airfoil and the oncoming airflow direction; (3) *Mach number*: the ratio of the airfoil's speed to the speed of sound in the same medium. Given the above input and initial conditions, SU2 can calculate the air pressure and velocity values at each node of the mesh.

2.2 The CFD-GCN Learning System

To accelerate SU2 simulations, [6] developed CFD-GCN – a hybrid machine learning model that integrates the physical SU2 solver with a graph convolution network (GCN). This hybrid model aims to predict the SU2 simulation outcome associated with fine mesh using that associated with coarse mesh, as illustrated in Figure 1.

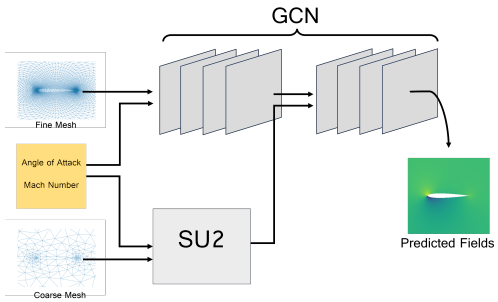

Figure 1: Illustration of CFD-GCN model [6]. Both the GCN model parameters and the coarse mesh's node positions are trainable.

Specifically, denote the fine mesh and coarse mesh as M_{fine} and M_{coarse}, respectively. Each mesh consists of a list of nodes specified by positional x, y-coordinates. Now consider a set of n settings of the AoA and Mach number parameters, and denote them as $\{P^1, P^2, ..., P^n\}$. For the i-th parameter setting P^i, denote the corresponding simulation outputs produced as

$$O^i_{\text{fine}} = \mathbf{Sol}(M_{\text{fine}}, P^i), \tag{1}$$

$$O^i_{\text{coarse}} = \mathbf{Sol}(M_{\text{coarse}}, P^i), \tag{2}$$

where $\mathbf{Sol}(\cdot, \cdot)$ stands for the SU2 solver. Further denote the graph convolutional network model as GCN_θ, where θ corresponds to the model parameters. Then, the overall training objective function can be written as

$$\min_{\theta, M_{\text{coarse}}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\Big(\text{GCN}_\theta(M_{\text{fine}}, O^i_{\text{coarse}}), O^i_{\text{fine}}\Big), \tag{3}$$

where $\mathcal{L}$ stands for the MSE loss. Here, the goal is to jointly learn and optimize the GCN model parameters θ and the coordinates of the nodes in the coarse mesh M_{coarse}. The coarse mesh usually contains very few number of nodes, and hence their positions critically affect the coarse simulation outcome O_{coarse}, which further affects the prediction accuracy of the GCN model.

2.3 Differentiability of Hybrid Model

To solve the above optimization problem, one standard approach is to use gradient-based methods such as SGD, which require computing the gradient $[\frac{\partial \mathcal{L}}{\partial \theta}; \frac{\partial \mathcal{L}}{\partial M_{\text{coarse}}}]$ using stochastic samples via back-propagation. Specifically, the partial derivative $\frac{\partial \mathcal{L}}{\partial \theta}$ can be calculated by standard machine learning

packages that support automatic differentiation (e.g., PyTorch [12] and TensorFlow [1]). For the other partial derivative $\frac{\partial \mathcal{L}}{\partial M_{\text{coarse}}}$, it can be decomposed as follows by the chain rule:

$$\frac{\partial \mathcal{L}}{\partial M_{\text{coarse}}} = \frac{\partial \mathcal{L}}{\partial O_{\text{coarse}}} \cdot \frac{\partial O_{\text{coarse}}}{\partial M_{\text{coarse}}}. \tag{4}$$

Note that the term $\frac{\partial \mathcal{L}}{\partial O_{\text{coarse}}}$ can also be calculated by standard machine learning packages. The challenge is to compute the other term $\frac{\partial O_{\text{coarse}}}{\partial M_{\text{coarse}}}$, which corresponds to the derivative of the output of the solver with regard to the input coarse mesh. In [6], the authors proposed to compute this term by adopting the SU2 solver to support automatic differentiation. However, this requirement can be restrictive for general black-box solvers. Notably, in physics and chemistry disciplines, ML models may be required to interface with experiments or complex simulation codes for which the underlying systems are non-differentiable [13, 14, 11, 2, 5].

We propose to estimate the partial derivative $\frac{\partial O_{\text{coarse}}}{\partial M_{\text{coarse}}}$ by leveraging the simulation outcomes, i.e., forward propagation through the solver [10]. The coordinate-wise zeroth-order gradient estimator estimates the gradient of the solver via the following steps. First, we sample a mini-batch $b \in \mathbb{N}$ of the coordinates of the input coarse mesh nodes uniformly at random. In our setting, each node has two coordinates to specify its position in the x-y plane. We denote the index of these sampled coordinates as $\{\xi_1, \xi_2, ..., \xi_b\}$, and denote the Euclidean basis vector associated with the coordinate ξ_j as e_{ξ_j}. Then, choosing a parameter $\mu > 0$, the coordinate-wise zeroth-order (Coordinate-ZO) gradient estimator is defined as follows:

$$\text{(Coordinate-ZO):} \quad \frac{\widehat{\partial} O}{\partial M} := \frac{1}{b} \sum_{j=1}^{b} \frac{O(M + \mu e_{\xi_j}) - O(M)}{\mu} e_{\xi_j}. \tag{5}$$

To elaborate, Coordinate-ZO estimates the gradient based on the finite-difference formula, which requires to perturb the input mesh M over the uniformly sampled coordinates $\{e_{\xi_j}\}_{j=1}^{b}$ and compute their corresponding simulation outcomes via running the solver. The main advantage of using the Coordinate-ZO estimator is its high asymptotic accuracy as μ tends to 0. When $\mu \downarrow 0$, each term in the above summation converges to the exact partial gradient of O with regard to the corresponding node coordinate. In practice, however, setting μ too small can lead to numerical instability and less accurate evaluations. The bias due the a non-vanishing μ often results in subpar performance compared to automatic differentiation (AD).

Furthermore, comparing to the finite-difference method, Coordinate-ZO requires only b function evaluations per mesh update step. For instance, a forward pass in SU2 takes approximately 2.0 seconds on the coarse mesh [6], leading to around 1000 seconds for a full finite-difference step, while the Coordinate-ZO estimation takes only $2.0 \times b$ seconds. Due to such high time cost, we did not provide the necessary rounds of SU2 simulations of obtaining the required accuracy; instead, we trade-off between the function call complexity and the model accuracy by tuning the parameter b.

While we considered other gradient estimators, like Gaussian and uniform smoothing, Coordinate-ZO was selected based on its empirical effectiveness with $b = 16$ and $\mu = 0.001$. We did not perform further experimentation or a comprehensive grid search, as comparing gradient estimators was not the primary aim of our research.

2.4 Training with Warm-Start

Asymmetry in Optimization. One key observation is that the optimization of the objective function in eq. (3) is highly asymmetric between the network model parameters θ and the coarse mesh M_{coarse}. To elaborate, optimizing a deep neural network's model parameters is challenging due to the nonconvex nature of the optimization problem and because the trained parameters are often substantially different from the randomly initialized parameters. Thus, in the initial training phase when the neural network parameters' gradients are large and noisy, they tend to amplify the noise of the mesh nodes' estimated gradients through the chain rule in eq. (4), which further slows down the overall convergence and degrades the generalization performance.

Warm-Start. Aiming to further improve the training of the hybrid model via zeroth-order approaches, we introduce a warm-start strategy to enhance convergence and generalization. This entails a two-stage training process:

- *Warm-up Stage*: Initially, we freeze the coarse mesh and focus solely on training the neural network parameters for 300 epochs. Given that the coarse mesh remains unchanged, no extra simulations are called upon during this phase.

- *Training Stage*: Leveraging the model developed in the prior stage, we unfreeze the coarse mesh and jointly train the network parameters and mesh using the zeroth-order training algorithm.

3 Experiment Results and Discussion

We apply the aforementioned zeroth-order gradient estimators with parameter $\mu = 1\text{e}{-3}$ to estimate the gradient of the PDE solver output with regard to the input coarse mesh coordinates, and then use the full gradient $[\frac{\partial \mathcal{L}}{\partial \theta}; \frac{\partial \mathcal{L}}{\partial M_{\text{coarse}}}]$ to train the model parameters θ and the coarse mesh coordinates M_{coarse} by optimizing the objective function in eq. (3).

The training dataset and test dataset consist of the range of AoA and mach numbers as shown in Table 1. The fixed fine mesh M_{fine} contains 6648 nodes, and the trainable coarse mesh M_{coarse} contains 354 nodes that are initialized by down-sampling the fine mesh. Moreover, for the training, we use the standard Adam optimizer [9] with learning rate 5×10^{-5} and batch size 16 (for sampling the AoA and mach number).

	AoA	Mach Number
Training data	[-10:1:10]	[0.2:0.05:0.45]
Test data	[-10:1:10]	[0.5:0.05:0.7]

Table 1: List of training data and test data.

We compare the Coordinate-ZO under warm-start with two baselines: (i) the gradient-based approach (referred to as *Grad*) proposed in [6], which requires a differentiable PDE solver; and (ii) the gradient-based approach but with a frozen coarse mesh (referred to as *Grad-FrozenMesh*), which does not optimize the coarse mesh at all. We observe that the Grad method with random initialization performs better than the warm-start initialization; this result will be used to compare with Coordinate-ZO.

We tested Coordinate-ZO with multiple batch sizes $b = 1, 2, 4$ for sampling the node coordinates (see eq. (5)). Using a larger batch size b leads to a more accurate estimation on $\frac{\partial O_{\text{coarse}}}{\partial M_{\text{coarse}}}$ but requires to more simulations. Therefore, we chose $b = 1$ to compare its test loss with that of the two gradient-based baselines, since it is most-sample-efficient for achieving the desired accuracy with the same number of simulations in our tested batch sizes $b = 1, 2, 4$. Figure 2-(a) shows the obtained test loss curves over the training epochs. The test loss of Coordinate-ZO are lower than that of the Grad-FrozenMesh approach and are higher than the Grad approach. This indicates that optimizing the coarse mesh using the Coordinate-ZO estimator leads to a lower test loss than using a frozen coarse mesh.

(a) Test loss comparison in the number of epochs

(b) Test loss comparison in the number of simulations

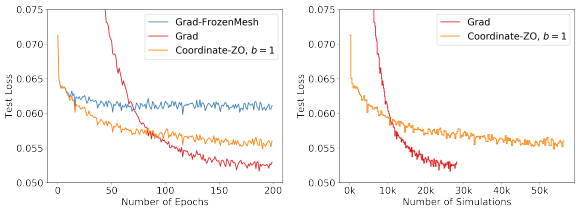

Figure 2: Test loss comparison among Coordinate-ZO, Grad and Grad-FrozenMesh.

We also see that gradient-based mesh optimization performs better than our zeroth-order optimization. This is expected as the gradient estimated by the Coordinate-ZO estimator is in general sparse and noisy, which slows down the convergence and degrades the test performance. Though the SU2 solver achieves better performance due to its automatic differentiation, our proposed hybrid model is compatible with other solvers which may not have implemented AD method. Figure 2-(b) illustrates the comparison between Grad and Coordinate-ZO in the number of simulations. The Grad-FrozenMesh approach does not update the coarse mesh at all and hence is not included in Figure 2-(b).

The following Figure 3 visualizes the pressure fields for input parameters AoA $= 9.0$ and Mach Number $= 0.8$. The ground truth is obtained by running the SU2 solver on the fine mesh to convergence. The field predicted by Coordinate-ZO is more accurate than that predicted by the

Grad-FrozenMesh baseline, due to the optimized coarse mesh. Also, the zoomed-in yellow region of the field predicted by our approach is closer to the Grad baseline.

(a) Ground Truth (b) Grad-FrozenMesh (c) Grad (d) Coordinate-ZO, $b = 1$

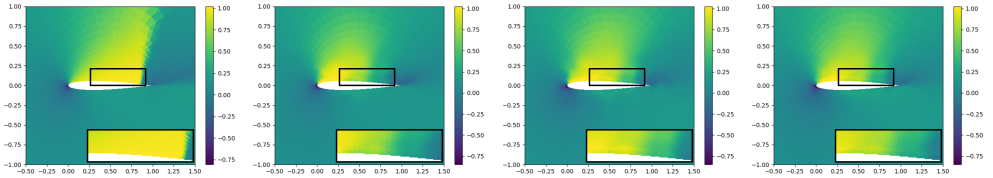

Figure 3: Visualization of the pressure fields predicted by Grad, Grad-FrozenMesh, and Coordinate-ZO with $b = 1$ for AoA $= 9.0$ and mach number $= 0.8$.

4 Conclusion

This study developed a learning system for fluid flow prediction that supports an end-to-end training of non-differentiable PDE solvers and deep learning models. We investigated the performance of optimizing this system using one zeroth-order estimator, which allow us to differentiate the solver without querying the exact gradients. Experiments showed that our approaches have competitive performance compare to gradient-based baselines, especially when adopting a warm-start strategy. We expect that our research will help integrate physical science modules into modern deep learning without the need for substantial adaptation.

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by the LLNL-LDRD Program under Project No. 23-ERD-030 (LLNL-PROC-856968).

S. Ma and Y. Zhou's work are supported by the National Science Foundation under Grants CCF-2106216, DMS-2134223, ECCS-2237830 (CAREER).

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: a system for Large-Scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[2] F. Abreu de Souza, M. Crispim Romão, N. F. Castro, M. Nikjoo, and W. Porod. Exploring parameter spaces with artificial intelligence and machine learning black-box optimization algorithms. *Phys. Rev. D*, 107:035004, Feb 2023.

[3] Y. Afshar, S. Bhatnagar, S. Pan, K. Duraisamy, and S. Kaushik. Prediction of Aerodynamic Flow Fields Using Convolutional Neural Networks. *Computational Mechanics*, 64(2):525–545, Aug. 2019.

[4] G. K. Batchelor. *An introduction to fluid dynamics*. Cambridge university press, 1967.

[5] A. G. Baydin, K. C. NYU, M. Feickert, L. Gray, L. Heinrich, A. H. NYU, A. M. V. M. Neubauer, J. Pearkes, N. Simpson, N. Smith, et al. Differentiable programming in high-energy physics. *Submitted as a Snowmass LOI*, 2020.

[6] F. D. A. Belbute-Peres, T. Economon, and Z. Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *international conference on machine learning*, pages 2402–2411. PMLR, 2020.

[7] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso. Su2: An open-source suite for multiphysics simulation and design. *Aiaa Journal*, 54(3):828–846, 2016.

[8] 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*, pages 481–490, 2016.

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] S. Liu, J. Chen, P.-Y. Chen, and A. Hero. Zeroth-order online alternating direction method of multipliers: Convergence analysis and applications. In *International Conference on Artificial Intelligence and Statistics*, pages 288–297. PMLR, 2018.

[11] G. Louppe, J. Hermans, and K. Cranmer. Adversarial variational optimization of non-differentiable simulators. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1438–1447. PMLR, 2019.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[13] A. Thelen, X. Zhang, O. Fink, Y. Lu, S. Ghosh, B. D. Youn, M. D. Todd, S. Mahadevan, C. Hu, and Z. Hu. A comprehensive review of digital twin—part 1: modeling and twinning enabling technologies. *Structural and Multidisciplinary Optimization*, 65(12):354, 2022.

[14] I. Tsaknakis, B. Kailkhura, S. Liu, D. Loveland, J. Diffenderfer, A. M. Hiszpanski, and M. Hong. Zeroth-order sciml: Non-intrusive integration of scientific software with deep learning. *arXiv preprint arXiv:2206.02785*, 2022.

[15] K. Um, X. Hu, and N. Thuerey. Liquid Splash Modeling with Neural Networks. *arXiv:1704.04456 [cs]*, Apr. 2017.

[16] S. Wiewel, M. Becher, and N. Thuerey. Latent-space Physics: Towards Learning the Temporal Evolution of Fluid Flow. *arXiv:1802.10123 [cs]*, Feb. 2018.