When Non-Differentiable PDE Solver Meets Deep Learning: Partially-Differentiable Learning for Efficient Fluid Flow Prediction

Anonymous submission

Abstract

There has been a growing interest in utilizing deep learning techniques to predict solutions to partial differential equations (PDEs) that arise in computational fluid dynamics. Recent research proposed to integrate a differentiable PDE solver into a deep learning model to improve the generalization performance. However, the approach therein requires the PDE solver to support automatic differentiation through the iterative numerical process, a feature not readily available in many existing solvers. In this study, we explore the feasibility of combining a non-differentiable solver with deep learning for fluid flow prediction. Specifically, we investigate a partially-differentiable hybrid model that integrates a non-differentiable PDE solver into a differentiable deep graph neural network. To train such a learning system using gradient-based algorithms, we employ various zeroth-order gradient estimators to differentiate the PDE solver using forward propagation. Experiments demonstrate that these zeroth-order gradient estimators lead to convergent models that outperform a first-order baseline trained using a frozen input mesh to the solver. Moreover, with a simple warm-start on the neural network parameters, we show that models trained by these zeroth-order algorithms achieve an accelerated convergence and improved generalization performance.

Introduction

Studying the fluid dynamics behavior is a fundamental and challenging pursuit in the physical sciences due to the complex nature of solving highly nonlinear and large-scale Navier-Stokes partial differential equations (PDEs) (Batchelor 1967). Conventionally, numerical solvers employ finite-difference methods to approximate PDE solutions on discrete meshes. However, this approach requires running a full simulation for each mesh, which can impose a substantial computational burden, particularly when dealing with highly granular meshes.

To address these challenges, there has been a growing interest in the application of data-driven deep learning techniques for inferring solutions to PDEs, particularly within the field of computational fluid dynamics (CFD) (Afshar et al. 2019; Guo, Li, and Iorio 2016; Wiewel, Becher, and Thuerey 2018; Um, Hu, and Thuerey 2017; Belbute-Peres, Economon, and Kolter 2020). Notably, one line of research aims to predict the outcomes of physical processes directly using data-driven deep learning methods (Afshar et al. 2019; Guo, Li, and Iorio 2016). However, these approaches do not consider the underlying physical mechanisms and often require a substantial volume of data, particularly high-granularity simulation data. Consequently, the learned models often exhibit limited generalizability when solving PDEs with unseen parameters.

Recently, one promising approach has emerged to further enhance the generalization performance (Belbute-Peres, Economon, and Kolter 2020). This approach proposes to integrate a physical PDE solver into a graph neural network, wherein the solver generates simulation data at a coarse granularity to aid the network in predicting simulation outcomes at a fine granularity. Notably, the solver is required to be differentiable, which enables the simultaneous optimization of both the coarse mesh and network parameters using stochastic gradient optimizers. Despite demonstrating significant enhancement in generalization performance, this approach requires that the solver supports automatic differentiation – an attribute lacking in many existing solvers. This limitation arises from the fact that some solvers are tailored to specific application domains and are complied by outdated languages, demanding substantial effort and interdisciplinary expertise to rewrite them with automatic differentiation capability. On the other hand, since the optimization of the coarse mesh is considerably less complex than optimizing the neural network parameters, it is possible to conduct mesh optimization using noisy algorithms that do not rely on exact gradient queries. Inspired by these insights, we are motivated to study the following key question.

- *Q: Can we develop a hybrid learning system for fluid flow prediction that integrates non-differentiable solvers with differentiable deep neural networks?*

In this work, we provide an affirmative answer to the above question by developing a hybrid learning system that supports non-differentiable solvers for efficient fluid flow prediction. We summarize our contributions as follows.

Our Contributions

We consider the CFD-GCN hybrid machine learning model proposed in Belbute-Peres, Economon, and Kolter (2020) for fluid flow prediction. In particular, this hybrid model involves a differentiable PDE solver named SU2 (Economon et al. 2015). However, in many practical situations, exposing new variables for differentiation in an external module

requires either highly complicated modification on the underlying codes or is simply not supported for closed-sourced commercial software. To overcome this obstacle, our goal is to train this hybrid model without querying differentiation through the PDE solver.

1. Based on the differentiable CFD-GCN hybrid model, we develop various algorithms that can train this model without directly querying the exact gradients of the solver. Consequently, one can integrate any non-differentiable solvers into the hybrid system and apply our algorithms to train the system parameters.
2. Specifically, we apply two classical zeroth-order gradient estimators, i.e., Coordinate-ZO and Gaussian-ZO (see eqs. (5) and (6)) to estimate the gradients of the solver solely via forward-propagation, and use them together with the neural network's gradients to train the hybrid model. Our experiments on fluid flow prediction verify the improved generalization performance of these zeroth-order algorithms compared with a first-order gradient baseline trained using a frozen input mesh to the solver. Furthermore, we propose a mixed gradient estimator named Gaussian-Coordinate-ZO (see eq. (7)) that combines the advantages of the previous two zeroth-order estimators, and show that it leads to an improved generalization performance.
3. Lastly, we observe an asymmetry between the optimization of mesh parameters and that of neural network model parameters, based on which we propose a warm-start strategy to enhance the training process. Experiments show that such a simple strategy leads to an improved generalization performance.

Related Work

Deep learning for CFD. The incorporation of machine learning into Computational Fluid Dynamics (CFD) has been a topic of growing interest in recent years. In many cases, the deep learning techniques are directly used to predict physical processes (Afshar et al. 2019; Guo, Li, and Iorio 2016). Some recent studies combine the physical information and deep learning model to obtain a hybrid model (Belbute-Peres, Economon, and Kolter 2020; Um et al. 2020). This paper goes one step further to consider the scenario that the external physical features extractor contains non-differentiable parameters.

Differentiability of Solvers. Fluid dynamics simulators, like SU2 (Economon et al. 2015), have incorporated adjoint-based differentiation and are extensively used in fields such as shape optimization (Jameson 1988) and graphics (McNamara et al. 2004). However, in physics and chemistry disciplines, ML models may be required to interface with numerical solvers or complex simulation codes for which the underlying systems are non-differentiable (Thelen et al. 2022; Tsaknakis et al. 2022; Louppe, Hermans, and Cranmer 2019; Abreu de Souza et al. 2023; Baydin et al. 2020).

Zeroth-order optimization. The adoption of zeroth-order optimization techniques has garnered considerable attention within the domain of optimization research. Instead of directly evaluating the function derivative, the zeroth-order method applies the random perturbation to the function input to obtain the gradient estimation (Liu et al. 2018; Duchi et al. 2015; Liu et al. 2020). Based on the types of random direction, the zeroth-order estimator can be roughly classified into Gaussian estimator (Berahas et al. 2022; Nesterov and Spokoiny 2017) and coordinate estimator (Berahas et al. 2022; Kiefer and Wolfowitz 1952). This paper mainly focuses on the performance of these classical zeroth-order optimization techniques in the hybrid model.

Background: Hybrid Machine Learning for Fluid Flow Prediction

In this section, we recap the hybrid machine learning system named CFD-GCN proposed by Belbute-Peres, Economon, and Kolter (2020) for fluid flow prediction. We first introduce the SU2 PDE solver developed for computation fluid dynamics. Then, we introduce the CFD-GCN model that integrates this PDE solver into a graph convolutional neural network. Lastly, we discuss the differentiability issue of hybrid machine learning systems.

The SU2 PDE Solver

SU2 is a numerical solver developed for solving PDEs that arise in computation fluid dynamics (Economon et al. 2016). SU2 applies the finite volume method (FVM) to solve PDEs and calculate certain physical quantities over a given discrete mesh. In particular, we consider SU2 because it supports automatic differentiation, and hence provides a first-order gradient baseline to compare with our zeroth-order algorithms.

To explain how SU2 works, consider the task of calculating the air flow fields around an airfoil. The input to SU2 includes the following three components.

1. Mesh: a discrete approximation of the continuous flow field. The mesh consists of multiple nodes whose positions depend on the shape of the airfoil;
2. Angle of attack (AoA): a parameter that specifies the angle between the chord line of the airfoil and the oncoming airflow direction;
3. Mach number: a parameter that specifies 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. However, one issue of using a numerical solver is that the runtime scales polynomially with regard to the number of nodes in the mesh. For example, for a standard airfoil problem, the run time of SU2 is about two seconds for a coarse input mesh with 354 nodes, and it increases to more than three minutes for a fine input mesh with 6648 nodes (Belbute-Peres, Economon, and Kolter 2020). Therefore, it is generally computationally costly to obtain a simulation result at a high resolution, which is much desired due to its better approximation of the original PDEs over the continuous field. Another issue is the lack of generalizability, i.e., one needs to run SU2 to obtain the simulation result for each instance of the AoA and mach number parameters.

The CFD-GCN Learning System

To accelerate SU2 simulations, Belbute-Peres, Economon, and Kolter (2020) developed CFD-GCN – a hybrid machine learning model that integrates the physical SU2 solver into 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. In particular, since the coarse mesh usually contains very few nodes, it is critical to jointly optimize the coarse mesh's node positions with the GCN model parameters. For more details, please refer to the Figure 1 in (Belbute-Peres, Economon, and Kolter 2020).

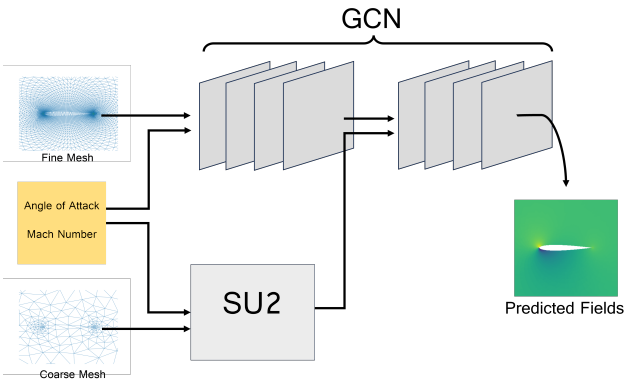

Figure 1: Illustration of CFD-GCN (Belbute-Peres, Economon, and Kolter 2020). Both the GCN model parameters and the coarse mesh's node positions are trainable.

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 by the SU2 solver with fine and coarse meshes respectively 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 reason is that the coarse mesh usually contains very few number of nodes, and hence the positions of the nodes critically affect the coarse simulation outcome O_{coarse}, which further affects the prediction accuracy of the GCN model.

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 (Paszke et al. 2019) and TensorFlow (Abadi et al. 2016)). 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 (Belbute-Peres, Economon, and Kolter 2020), the authors proposed to compute this term by adopting the SU2 solver that supports automatic differentiation. However, this requirement can be restrictive for general hybrid models as many existing solvers are not differentiable. 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 (Thelen et al. 2022; Tsaknakis et al. 2022; Louppe, Hermans, and Cranmer 2019; Abreu de Souza et al. 2023; Baydin et al. 2020). On the other hand, as the optimization of the coarse mesh is considerably less complex than tuning the neural network parameters, it is possible to conduct mesh optimization through noisy algorithms that do not rely on queries of exact gradient. Motivated by these insights, we aim to develop an effective approach that allows to directly integrate non-differentiable solvers into differentiable deep learning models.

Differentiating Numerical Solver via Forward Propagation

In the scenario that the PDE solver does not support automatic differentiation, one can only 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. Fortunately, the classic zeroth-order gradient estimators provide solutions to estimate high-dimensional gradients using only forward propagation (Liu et al. 2018). Next, we apply the classic coordinate-wise and Gaussian formulas of the zeroth-order gradient estimators to estimate the derivative of the solver. To simplify the presentation, we focus on scalar-valued simulation outcome O as a function of the input mesh M. In practice, O is vector-valued and we simply apply the estimators to each dimension of it.

Coordinate-wise Gradient Estimator

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, choose parameter $\mu > 0$, the coordinate-wise zeroth-order (Coordinate-ZO) gradient estimator is constructed as follows.

(Coordinate-ZO):

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

To elaborate, Coordinate-ZO estimates the gradient based on the finite-difference formula, which requires to perturb the input mesh M over the sampled coordinates $\{e_{\xi_j}\}_{j=1}^{b}$ and compute their corresponding simulation outcomes via running the solver.

Remark 1: The main advantage of using the Coordinate-ZO estimator is its high accuracy and exactness. This is because 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. However, to estimate the partial derivative over b coordinates, Coordinate-ZO needs to query the function value oracle O in total $b+1$ times, and hence is not efficient when estimating gradients in a high dimension space. In particular, in the CFD-GCN system, this requires querying the simulation outcome associated with $b+1$ coarse meshes, which can be time-consuming.

Gaussian Gradient Estimator

Another popular zeroth-order gradient estimator estimates the gradient via Gaussian perturbations. Specifically, we first generate a mini-batch $b \in \mathbb{N}$ of standard Gaussian random vectors, each with dimension d that equals 2x of the number of nodes in the mesh (because each node has two coordinates). Denote these generated Gaussian random vectors as $\{g_1, g_2, ..., g_b\} \sim \mathcal{N}(0, I)$. Then, choose parameter $\mu > 0$, the Gaussian zeroth-order (Gaussian-ZO) gradient is constructed as follows.

(Gaussian-ZO):

$$\frac{\widehat{\partial} O}{\partial M} := \frac{1}{b} \sum_{j=1}^{b} \frac{O(M + \mu g_j) - O(M)}{\mu} g_j. \quad (6)$$

Note that the Gaussian-ZO estimator perturbs the mesh using a dense Gaussian vector g_j. This is very different from the Coordinate-ZO that perturbs a single coordinate of the mesh nodes using e_{ξ_j}. In fact, Gaussian-ZO is a stochastic approximation of the gradient of a Gaussian-smoothed objective function $O_\mu(M) := \mathbb{E}_g[O(M + \mu g)]$.

Remark 2: The main advantage of using the Gaussian-ZO estimator is that it estimates the gradient over the full dimensions. Therefore, even with batch size $b = 1$, the formula still generates a gradient estimate over all the coordinates. This is an appealing property to the CFD-GCN learning system, as one can update the entire coarse mesh with just one Gaussian noise (one additional query of simulation). However, the gradient estimate generated by Gaussian-ZO usually suffers from a high variance, and hence a large batch size is often needed to control the variance. Moreover, the Gaussian-ZO formula estimates the gradient of the smoothed objective function $O_\mu(M)$, which introduces extra approximation error.

Gaussian-Coordinate Gradient Estimator

To avoid the high variance issue of the Gaussian-ZO estimator and the low sample efficiency issue of the Coordinate-ZO estimator, we propose Gaussian-Coordinate-ZO estimator – a mixed zeroth-order gradient estimator that leverages the advantages of both estimators. Specifically, we first randomly sample a mini-batch of d coordinates of the mesh nodes, denoted as $D := \{\xi_1, \xi_2, ..., \xi_d\}$. Then, we generate a mini-batch of b standard Gaussian random vectors, denoted as $\{g_1, g_2, ..., g_b\}$. Lastly, we construct the Gaussian-Coordinate-ZO estimator as follows with parameter $\mu > 0$, where $[g]_D$ denotes a vector whose coordinates excluding D are set to be zero.

(Gaussian-Coordinate-ZO):

$$\frac{\widehat{\partial} O}{\partial M} := \frac{1}{b} \sum_{j=1}^{b} \frac{O(M + \mu [g_j]_D) - O(M)}{\mu} [g_j]_D. \quad (7)$$

Intuitively, the Gaussian-Coordinate-ZO estimator can be viewed as a Gaussian-ZO estimator applied only to the coordinates in D.

Remark 3: Compared to the Gaussian-ZO estimator that estimates the gradient over all the coordinates, Gaussian-Coordinate-ZO estimates the gradient only over the coordinates in D, and hence is less noisy. On the other hand, unlike the Coordinate-ZO estimator that estimates the gradient over a single coordinate per sample, the Gaussian-Coordinate-ZO estimator estimates the gradient over a mini-batch of coordinates in D even with a single Gaussian sample. This leads to an improved sample/learning efficiency as validated by our experiments later.

Experiments

Experiment Setup

We apply the three aforementioned zero-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 following range of AoA and mach numbers.

	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.

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 (Kingma and Ba 2014) with learning rate 5×10^{-5} and batch size 16 (for sampling the AoA and mach number).

We compare our ZO methods with two baselines: (i) the gradient-based approach (referred to as *Grad*) proposed in (Belbute-Peres, Economon, and Kolter 2020), 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.

Results and Discussion

1. Coordinate-ZO. We first implement the Coordinate-ZO approach with different batch sizes $b = 1, 2, 4$ for sampling the node coordinates (see eq. (5)), and compare its test loss with that of the two gradient-based baselines. Figure 2 (left) shows the obtained test loss curves over the training epochs. It can be seen that the test loss curves of Coordinate-ZO are lower than that of the Grad-FrozenMesh approach and are higher than that of 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, and leads to a higher test loss than gradient-based mesh 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. In particular, as the batch size b increases, Coordinate-ZO achieves a lower test loss at the cost of running more coarse mesh simulations, as illustrated by Figure 2 (right). We note that the Grad-FrozenMesh approach does not update the coarse mesh at all and hence is not included in the right figure.

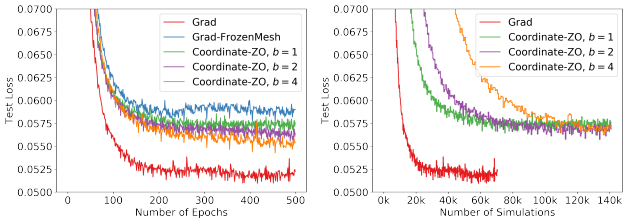

Figure 2: Test loss comparison among Coordinate-ZO, Grad and Grad-FrozenMesh.

The following Figure 3 visualizes the pressure fields predicted by the Grad, Grad-FrozenMesh baselines and our Coordinate-ZO approach with batch size $b = 4$, for input parameters AoA $= 9.0$ and mach number $= 0.8$. It can be seen that the field predicted by our Coordinate-ZO approach is more accurate than that predicted by the Grad-FrozenMesh baseline, due to the optimized coarse mesh. Also, the highlighted yellow region of the field predicted by our approach looks very close to that predicted by the Grad baseline (which uses a differentiable solver).

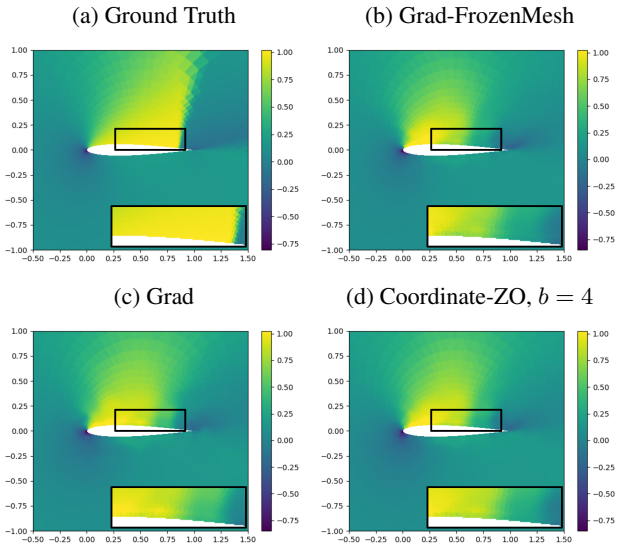

Figure 3: Visualization of the pressure fields predicted by Grad, Grad-FrozenMesh and Coordinate-ZO with $b = 4$ for AoA $= 9.0$ and mach number $= 0.8$.

Figure 4 compares the optimized coarse mesh (red) obtained by our Coordinate-ZO approach with the original frozen coarse mesh (blue). It can be seen that many nodes' positions have been updated by Coordinate-ZO to improve the overall prediction performance. In particular, the nodes in the dense area tend to have a more significant shift than those in the sparse area. This is reasonable as the dense nodes typically play a more important role in simulations.

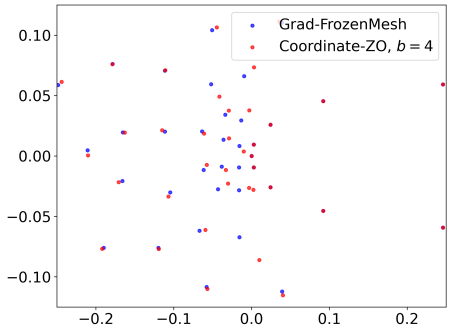

Figure 4: Comparison between the mesh optimized by Coordinate-ZO with the original frozen mesh.

2. Gaussian-ZO. We further implement the Gaussian-ZO approach with different batch sizes $b = 1, 2, 4$ and compare its test loss with that of the two baselines. Figure 5 shows the obtained comparison results, which are very similar to those obtained by Coordinate-ZO in Figure 2. This indicates that Gaussian-ZO can optimize the coarse mesh with a comparable performance to that of the Coordinate-ZO approach, and the convergence speed is slower than the gradient-based Grad approach due to the intrinsic high variance of the Gaussian-ZO estimator.

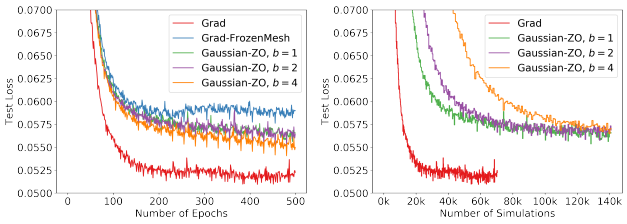

Figure 5: Test loss comparison among Gaussian-ZO, Grad and Grad-FrozenMesh.

The following Figure 6 visualizes the pressure fields predicted by the Grad, Grad-FrozenMesh baselines and our Gaussian-ZO approach with batch size $b = 4$, for input parameters AoA $= -10.0$ and Mach $= 0.8$. One can see that the highlighted yellow area predicted by Gaussian-ZO is more accurate than that predicted by Grad-FrozenMesh, and is close to the predictions of the Grad approach.

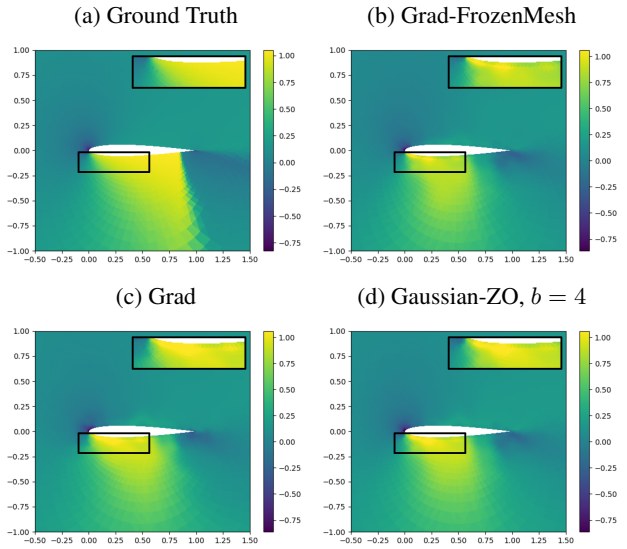

Figure 6: Compare simulation results with Grad, Grad-FrozenMesh and Gaussian-ZO.

3. Gaussian-Coordinate-ZO. Lastly, we test the Gaussian-Coordinate-ZO approach (referred to as *Gauss-Coord-ZO*) with different choices of parameters d and b (see eq. (7)). Figure 7 (left) shows the comparison result under a fixed batch size $b = 1$ and different choices of d. Due to the fixed batch size, these Gauss-Coord-ZO algorithms query the same total number of simulations, and one can see that increasing d leads to a slightly lower test loss. Moreover, Figure 7 (right) shows the comparison result under a fixed $d = 4$ and different choices of b. One can see that increasing b leads to a lower test loss due to the reduced variance of the Gaussian-Coordinate-ZO estimator. Overall, we found that the choice of parameters $b = 1, d = 16$ achieves the best balance between performance and sample/time efficiency.

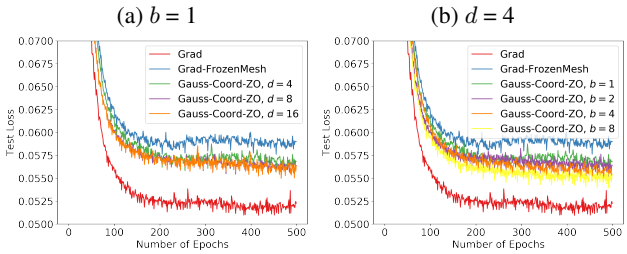

Figure 7: Test loss comparison of Gauss-Coord-ZO with different choices of b and d.

We further compare the performance of Gauss-Coord-ZO ($d = 16$) with that of the two baselines and the previous two zeroth-order approaches. Figure 8 shows the comparison results obtained with different choices of batch size. With batch size $b = 1$, it can be seen from the left figure that Gauss-Coord-ZO achieves a slightly lower test loss than Coordinate-ZO and Gaussian-ZO, demonstrating the advantage of this mixed zeroth-order estimator. On the other hand, as the batch size increases to $b = 4$ in the right figure, all the three zeroth-order approaches achieve a similar test loss, and is slightly better than that achieved by Gauss-Coord-ZO with $b = 1$. Overall, we observe that the Gauss-Coord-ZO estimator with $b = 1, d = 16$ already leads to a good practical performance. This choice of parameter is also favorable as the batch size $b = 1$ requires running the minimum number of simulations.

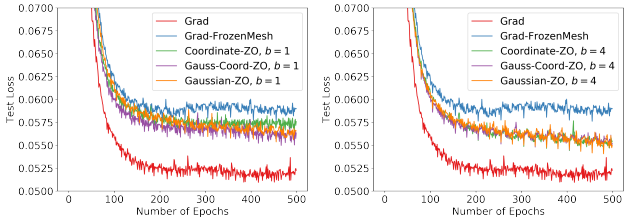

Figure 8: Test loss comparison among Coordinate-ZO, Gaussian-ZO, and Gauss-Coord-ZO.

Figure 9 shows the comparison among the simulation results predicted by Grad, Grad-FrozenMesh and Gaussian-Coordinate-ZO, all with batch size $b = 4$ and input parameters AoA $= 5.0$ and Mach $= 0.4$. It can be seen that all the three zeroth-order approaches obtain very similar prediction results that are close to the ground truth.

Improve Generalizability via Warm-Start

In this subsection, we propose a warm-start strategy to further improve the generalization performance of the proposed zeroth-order approaches.

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 deep neural network's model parameters is known to be a challenging

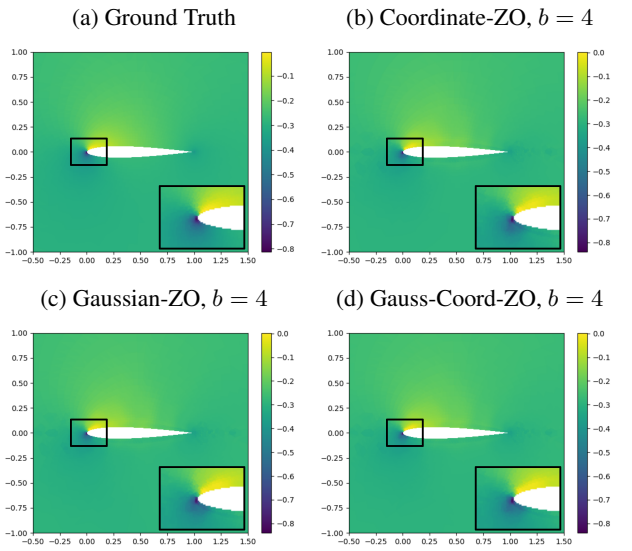

Figure 9: Compare simulation results with Grad, Grad-FrozenMesh and Gauss-Coord-ZO.

problem due to nonconvex optimization, and the trained parameters are often substantially different from the initialized parameters. As a comparison, optimizing coarse mesh is far less complex, since the optimized mesh is usually close to the initialized mesh, as shown in Figure 4. 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. With the above insight, we propose a simple warm-start strategy to accelerate the training of the hybrid model using zeroth-order approaches and improve the generalization performance. Specifically, we propose the following two-stage training process.

- Warm-up Stage: we first freeze the coarse mesh and train only the neural network parameters for 300 epochs. We note that this stage does not need to query any additional simulations due to the frozen coarse mesh.
- Stage two: with the model obtained in the previous stage, we unfreeze the coarse mesh and jointly train it with the network parameters using any of the previously discussed three zeroth-order algorithms.

	Gauss-Coord-ZO	Coord-ZO	Gauss-ZO
Random	0.05650	0.05721	0.05745
Warm-start	0.05540	0.05463	0.05455

Table 2: Comparison of the best test loss achieved by the zeroth-order approaches within 200 epochs using random initialization and warm-start. All approaches query the same number of simulations, and the batch size b is set to 1.

Table 2 compares the best test loss achieved by all the three zeroth-order approaches with $b = 1$ within 200 training epochs using random initialization and warm-start. Note that under both random initialization and warm-start, all the approaches query the same total number of simulations. It can be seen that with the warm-start strategy, all the zeroth-order approaches achieve improved generalization performance without querying additional simulations.

Figure 10 further compares the predicted pressure fields generated by these zeroth-order approaches under random initialization and warm-start for input parameters AoA $= 5.0$ and Mach $= 0.8$. It can be seen that all the simulation results produced by the model trained with warm-start are more accurate than those trained with random initialization.

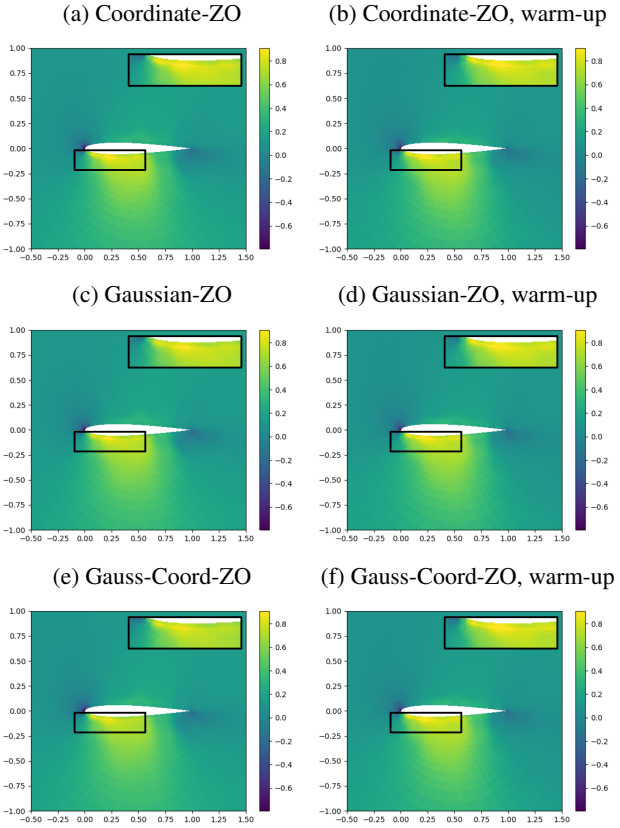

Figure 10: Comparison of simulation results predicted by training with random initialization and warm-start.

Conclusion

This study developed a learning system for fluid flow prediction that supports non-differentiable PDE solvers and deep learning models. We investigated the performance of optimizing this system using various zeroth-order estimators, 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.

References

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

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

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

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

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

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

Berahas, A. S.; Cao, L.; Choromanski, K.; and Scheinberg, K. 2022. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *Foundations of Computational Mathematics*, 22(2): 507–560.

Duchi, J. C.; Jordan, M. I.; Wainwright, M. J.; and Wibisono, A. 2015. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5): 2788–2806.

Economon, T. D.; Palacios, F.; Copeland, S. R.; Lukaczyk, T. W.; and Alonso, J. J. 2015. SU2: An Open-Source Suite for Multiphysics Simulation and Design. *AIAA Journal*, 54(3): 828–846.

Economon, T. D.; Palacios, F.; Copeland, S. R.; Lukaczyk, T. W.; and Alonso, J. J. 2016. SU2: An open-source suite for multiphysics simulation and design. *Aiaa Journal*, 54(3): 828–846.

Guo, X.; Li, W.; and Iorio, F. 2016. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 481–490.

Jameson, A. 1988. Aerodynamic Design via Control Theory. *Journal of Scientific Computing*, 3(3): 233–260.

Kiefer, J.; and Wolfowitz, J. 1952. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 462–466.

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

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

Liu, S.; Chen, P.-Y.; Kailkhura, B.; Zhang, G.; Hero III, A. O.; and Varshney, P. K. 2020. A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications. *IEEE Signal Processing Magazine*, 37(5): 43–54.

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

McNamara, A.; Treuille, A.; Popovic, Z.; and Stam, J. 2004. Fluid control using the adjoint method. *ACM Transactions On Graphics (TOG)*, 23(3): 449–456.

Nesterov, Y.; and Spokoiny, V. 2017. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17: 527–566.

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

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

Tsaknakis, I.; Kailkhura, B.; Liu, S.; Loveland, D.; Diffenderfer, J.; Hiszpanski, A. M.; and Hong, M. 2022. Zeroth-Order SciML: Non-intrusive Integration of Scientific Software with Deep Learning. *arXiv preprint arXiv:2206.02785*.

Um, K.; Brand, R.; Fei, Y. R.; Holl, P.; and Thuerey, N. 2020. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. *Advances in Neural Information Processing Systems*, 33: 6111–6122.

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

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