Manuscript accepted for publication in *Physics of Fluids*, 2024

Field Inversion Machine Learning Augmented Turbulence Modeling for Time-Accurate Unsteady Flow

Lean Fang[1] and Ping He[1, a)]

[1]*Department of Aerospace Engineering, Iowa State University, Ames, Iowa 50011, USA*

Field inversion machine learning (FIML) has the advantages of model consistency and low data dependency and has been used to augment imperfect turbulence models. However, the solver-intrusive field inversion has a high entry bar, and existing FIML studies focused on improving only steady-state or time-averaged periodic flow predictions. To break this limit, this paper develops an open-source FIML framework for time-accurate unsteady flow, where both spatial and temporal variations of flow are of interest. We augment a Reynolds-Averaged Navier-Stokes (RANS) turbulence model's production term with a scalar field. We then integrate a neural network (NN) model into the flow solver to compute the above augmentation scalar field based on local flow features at each time step. Finally, we optimize the weights and biases of the built-in NN model to minimize the regulated spatial-temporal prediction error between the augmented flow solver and reference data. We consider the spatial-temporal evolution of unsteady flow over a 45-degree ramp and use only the surface pressure as the training data. The unsteady-FIML-trained model accurately predicts the spatial-temporal variations of unsteady flow fields. In addition, the trained model exhibits reasonably good prediction accuracy for various ramp angles, Reynolds numbers, and flow variables (e.g., velocity fields) that are not used in training, highlighting its generalizability. The FIML capability has been integrated into our open-source framework DAFoam. It has the potential to train more accurate RANS turbulence models for other unsteady flow phenomena, such as wind gust response, bubbly flow, and particle dispersion in the atmosphere.

[a)] Corresponding author. phe@iastate.edu

NOMENCLATURE

C_p	=	Pressure coefficient $(p-p_0)/(0.5\rho_0 U_0^2)$
f	=	Objective function at each time step
F	=	Time-averaged objective function
h	=	Channel height at the outlet, m
$\boldsymbol{H}$	=	Velocity influence vector
K	=	Total number of time step
p	=	Pressure, Pa
$\boldsymbol{R}$	=	Flow residual vector
Re	=	Reynolds number
$\boldsymbol{S}$	=	Mesh face area vector, m^2
t	=	Time, s
t^*	=	Non-dimensional time, tU_0/h
Δt	=	Time step, s
$\boldsymbol{U}$	=	Velocity vector, m/s
U_0	=	Velocity at the inlet, m/s
$U_x,\ U_y$	=	Velocity in the x and y directions, m/s
$U_x^*,\ U_y^*$	=	Normalized velocity $U_x/h,\ U_y/h$
$\boldsymbol{w}$	=	Flow state variable vector
$w,\ b$	=	Neural network's weights and biases
$\boldsymbol{x}$	=	Design variable vector
$x,\ y$	=	Streamwise and vertical coordinates, m
$x^*,\ y^*$	=	Normalized coordinates, $x/h,\ y/h$
β	=	Augmented flow field variable
η	=	Local flow features
$\nu,\ \nu_t$	=	Kinematic and turbulent viscosity, m^2/s
ν_{eff}	=	Effective viscosity $(\nu+\nu_t)$, m^2/s
ϕ	=	Face flux, m^3/s
ψ	=	Adjoint vector

I. INTRODUCTION

Computational fluid dynamics (CFD) is a powerful tool for analyzing three-dimensional flow fields, which allows us to better understand flow physics and design high-performance systems. One can run CFD with various levels of fidelity, ranging from the Reynolds-averaged Navier-Stokes (RANS) approach to direct numerical simulation (DNS). Because of its relatively low computational cost, the RANS approach is commonly used in the design and analysis of engineering systems, such as aircraft[1–3], spacecraft[4–6], cars[7–9], and wind turbines[10–12]. However, it is known that the RANS approach has imperfect physical models (e.g., turbulence models) that degrade its simulation accuracy for challenging flow conditions[13,14], such as flow separation and turbulence transition.

To correct the imperfect physical models for RANS CFD solvers, researchers have proposed various machine learning (ML) methods, such as physics-informed ML[15,16], evolutionary algorithms[17,18], various neural network models[19–22], and surrogate-based optimization[23,24]. More comprehensive reviews can be found in[25–27]. Compared with traditional human-intuition-based physical model development, ML methods can leverage data to accelerate the development of accurate and generalizable physical models. Field inversion machine learning (FIML) is a solver-embedded method proposed by Duraisamy and co-workers[28–30]. The main advantage of FIML is that it incorporates the entire CFD solver in the training phase (field inversion), so the training and prediction are consistent at the discretized level, which improves its prediction accuracy and generalizability[27]. Moreover, FIML allows one to flexibly use a wide range of available data for training, such as integrated values, surface variables, and sparse or partial field data. In the following, we conduct a brief literature review of existing FIML studies.

Singh, Medida, and Duraisamy[29] used experimental lift coefficients as training data and augmented the Spalart–Allmaras (SA) for predicting separated flow over airfoils. The augmented model significantly improved the lift prediction accuracy for various angles of attack. The trained model was also found to be generalizable for flow conditions, geometries, and flow solvers not used in training. To further improve the model consistency, Holland, Baeder, and Duraisamy[31] proposed a coupled field inversion and machine learning framework where the inference and learning were conducted simultaneously. He, Liu, and Gan[32] developed a continuous adjoint method to conduct field inversion and showed reasonably good performance for a wide range of 2D and 3D flow configurations. Ferrero, Iollo, and Larocca[33] used the wall isentropic Mach number data to augment the SA model for accurate flow predictions in gas turbine cascades. The trained model was shown to have good predictive accuracy with various unseen Mach numbers and blade geometries. Michelen Strofer, Zhang, and Xiao[34] developed an open-source framework that used an ensemble Kalman filtering (EnKF) method, instead of an adjoint-based method, to conduct field inversion. Yang and Xiao[35] used the EnKF method to consider the laminar-to-turbulent flow transition over airfoils. They augmented a four equation $k-\omega-\gamma-A_r$ turbulence model, which showed reasonably good performance in predicting the transition location for various angles of attack. The EnKF method was also used to augment the turbulence model's prediction accuracy for the interaction between the shock wave and boundary layer[36], laminar-to-turbulent flow transition in hypersonic boundary layer[37], and flow over a hump[38]. Hafez *et al.*[39] augmented the $k-\omega$ SST turbulence model for predicting traditional flow over flat-plate T3 series cases. The augmented model was found to significantly improve the prediction accuracy in the surface friction and boundary layer thickness. Fidkowski[40] considered laminar-to-turbulent flow transition by augmenting an algebraic model, which multiplies an intermittency factor to the production term. He evaluated the effectiveness of FIML with various training options, such as using integrated or surface-distributed data and a combination of optimizing correction fields and tunable parameters for the transitional model. Ho and West[41] augmented the $k-\omega$ SST turbulence model for three-dimensional flows over bumps. The trained model was found to improve mean velocity and turbulent kinetic energy predictions for unseen flows and geometries. Yan, Zhang, and Chen[42] augmented the SA turbulence model for predicting 3D flow over hills and proposed a method to validate the parameters used in the FIML to maximize its efficiency. Bidar *et al.*[43] evaluated the impact of using various combinations of multiple sources of data on the accuracy of the field inversion results for a hump and a periodic hill case. Wu and Zhang[44] proposed a symbolic regression method to improve the interpretability of FIML results. Their augmented $k-\omega$ SST turbulence model exhibits reasonably good generalizability for predicting separated flow for various 2D and 3D configurations. Later, Wu and Zhang[45] proposed a conditioned FIML method for generalizing the augmented turbulence model's prediction for a wide range of flows. Instead of allowing the augmentation variable to change everywhere in the flow field, they switched it off in the boundary layer to maintain the trained model's accuracy for attached flows. Recently, Bidar, Anderson, and Qin[46] proposed a greedy algorithm that can optimize the sensor placement in flow regions with high uncertainty to maximize the FIML's efficiency. The uncertainty map was based on the eigenspace perturbation of the baseline turbulence model.

Despite the above progress, all the FIML papers cited above considered only steady-state flow problems. This is primarily because FIML has a relatively high entry bar and requires an adjoint solver to compute gradients with respect to a large number of design variables[27]. Implementing an efficient adjoint solver requires the source codes and detailed knowledge of the CFD solver and is known to be time-consuming. Although the EnKF method[34] was proposed to circumvent the need for an adjoint solver in FIML, we are not aware of an EnKF-based FIML study for unsteady flow, probably due to its high computational cost for generating a large number of unsteady flow samples. Recently, Fidkowski[47] demonstrated that a steady-state trained FIML model could improve prediction accuracy for periodic unsteady flow. He used time-averaged data from periodic unsteady flow (ignoring the unsteady flow time history) and conducted steady-state FIML to correct the turbulence modeling error. He then used the corrected model to predict unsteady periodic flow and incorporated it into aerodynamic shape optimization. However, as will be shown in Sec. III D, there is no guarantee that a steady-state trained FIML model can accurately predict the time history of general, non-periodic unsteady flow. Accurately predicting the time history of general unsteady flow is important for many engineering and scientific problems, such as wind gust response, aircraft maneuver, multiple-phase flow evolution (e.g., bubbly flow), and particle transport in the atmosphere. In a recent study[48], we demonstrated that time-accurate unsteady field inversion could correct the turbulence model's prediction error and accurately capture the entire time history of unsteady flow evolution. This paper is a step forward in this direction and incorporates machine learning in our previous unsteady field inversion framework for predicting unseen unsteady flow conditions and geometries.

The objective of this paper is to develop an open-source FIML framework that can augment RANS turbulence models for accurate prediction of time-accurate unsteady flow, where both spatial and temporal variations of flow are of interest. We multiply the production term of the SA turbulence model by a scalar field β. Then, we integrate a deep neural network model into the CFD solver to compute the augmentation field β based on local flow features η at each time step. The unsteady FIML is formulated as an inverse problem where we optimize the weights and biases of the neural network model to minimize the CFD's regulated prediction error. The error is quantified by the difference between the CFD prediction and reference data at the selected mesh cells or boundaries for all time steps. To make the large-scale unsteady FIML optimization efficient, we propose a new PIMPLE-Krylov adjoint algorithm. First, we use the coupled pressure-implicit with splitting of operators (PISO) and semi-implicit method for pressure-linked equations (SIMPLE) algorithm (aka PIMPLE) to simulate the unsteady flow. Then, we use a Krylov method to solve the unsteady adjoint equations in reverse. The PIMPLE-Krylov adjoint algorithm's main benefit is that it can relax the CFL number to be greater than one, which significantly reduces the number of adjoint equations to be solved and accelerates the adjoint speed. This paper will elaborate on the proposed unsteady FIML framework and demonstrate its capability by augmenting turbulence modeling of unsteady flow over a ramp. We use the spatial-temporal distribution of pressure on the bottom wall as the reference data. We will evaluate the augmented model's accuracy by comparing the inverse flow fields (pressure and velocity) with the reference data. We will also evaluate the trained model's generalizability for predicting flow conditions and geometries not used in the training.

The proposed unsteady FIML framework has been implemented into our open-source, CFD-based optimization framework called DAFoam[49]. Details about downloading, installing, and using DAFoam can be found on its documentation website `https://dafoam.github.io`. The configuration files for reproducing the FIML results presented in this paper are archived on Mendeley data[50].

The rest of the paper is organized as follows. In Section II, we elaborate on the proposed unsteady FIML framework and all its components. The unsteady FIML results are presented and discussed in Section III and we summarize our findings in Section IV.

II. METHOD

In this section, we elaborate on the proposed unsteady FIML framework, followed by the details of its components, e.g., unsteady flow simulations, unsteady adjoint computation, built-in neural network, and flow feature calculation. We also discuss the effective numerical configurations to facilitate the unsteady FIML optimization with a built-in neural network model.

A. Proposed FIML framework for time-accurate unsteady flow

Figure 1 shows the proposed FIML framework for time-accurate unsteady flow. We conduct unsteady simulations by marching the solution with a time step Δt, starting from an initial flow field. At each time step, we conduct multiple iterations (PIMPLE loop) to converge the flow residuals as tightly as possible (see the next section for the details of the PIMPLE algorithm).

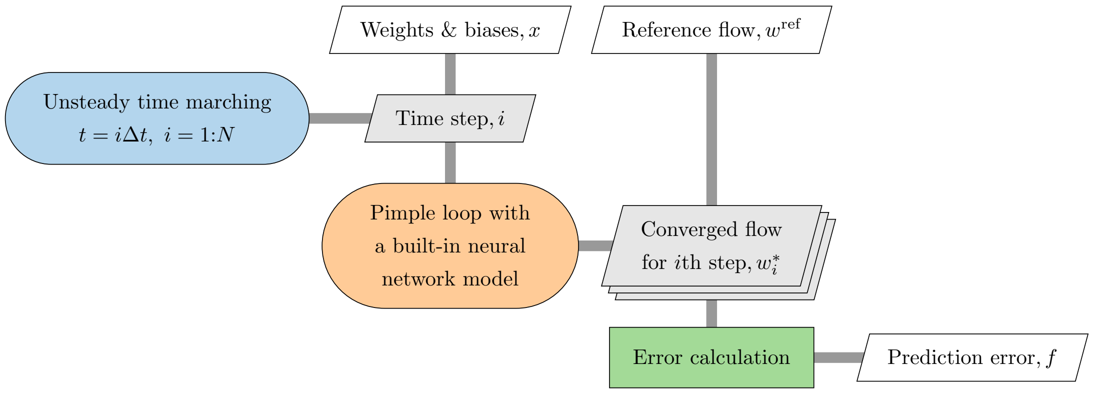
Weights & biases, x
Reference flow, w^{ref}
Unsteady time marching $t = i\Delta t,\ i = 1{:}N$
Time step, i
Pimple loop with a built-in neural network model
Converged flow for ith step, w_i^*
Error calculation
Prediction error, f

FIG. 1: Proposed field inversion machine learning framework for time-accurate unsteady flow. We use the extended design structure matrix (XDSM) representation proposed by Lambe and Martins[51]. The diagonal blocks are the components and the off-diagonal blocks are data transferred between components. The unsteady FIML optimizes the weights and biases (x) of the built-in neural network model to minimize the CFD solver's regulated prediction error. The regulated prediction error is computed as the selected flow field difference between the CFD and reference at all time steps, along with a regulation term.

For each PIMPLE iteration (Fig. 2), we use a feature extraction component to compute local flow features (η) based on the latest flow field (w). A built-in neural network model will then use the flow feature η and weights and biases as the inputs to compute an augmentation scalar field β. The augmentation β field will be added to the governing equation of turbulence models to correct its prediction. The above PIMPLE loop will be repeated to obtain converged flow fields for this time step i. The prediction error f is computed as the flow field difference between the CFD prediction and reference data for all time steps. To prevent overfitting, we also add a regulation term to the objective function to force the augmentation β field to be as close to one as possible. Finally, an inverse problem is solved by minimizing the regulated prediction error f (objective function) by optimizing the neural network model's weights and biases x (design variables).

The above unsteady FIML formulation couples the field inversion and neural network model (machine learning) and is similar to the method proposed in Holland, Baeder, and Duraisamy[31] for steady-state flow. However, most of the existing FIML studies (steady-state flow) decouple the field inversion and machine learning, as reviewed in the introduction. To be more specific, multiple field inversion problems are first solved to find the optimal augmentation field β for various

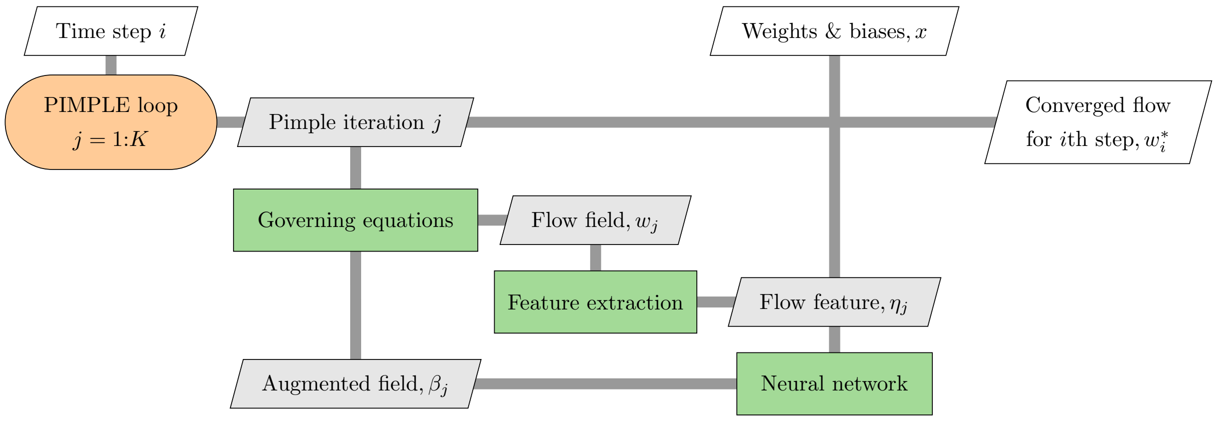

FIG. 2: Detailed structure of the PIMPLE loop component in Fig. 1. We conduct multiple PIMPLE iterations until all the flow residuals are small. In each PIMPLE iteration j, we solve the governing equation, compute the local flow features (η_j), compute the augmentation field β_j using the neural network model, and assign the updated augment field to the turbulence model's governing equation for the next iteration.

configurations (geometries and boundary conditions). Then, an off-line neural network model is trained by using local flow features (η) as the inputs and the above optimized augmentation field β as the outputs. Compared with the decoupled approach, the coupled FIML has the advantage of maximizing the model consistency between the field inversion (inference) and machine learning (augmentation) steps[31]. In addition, the number of design variables (weights and biases) is independent of the problem size (e.g., mesh size and unsteady time steps). However, the coupled FIML's disadvantage is that it incorporates a highly nonlinear neural network model in the optimization; therefore, the FIML problem is generally more challenging to converge. In addition, exploring different flow features and neural network architectures requires re-resolving the entire FIML problem, which is computationally expensive. In contrast, the decoupled approach can easily explore various flow features and neural network architectures because the inputs (flow features η) and outputs (optimal β fields) have been already pre-computed in the inference step.

Although the coupled and decoupled FIML approaches have their own advantages and disadvantages for steady-state problems, the former appears to be the only computationally feasible option for time-accurate unsteady flow problems. This is because the local flow features are time-dependent for unsteady problems, making the augmentation field β time-dependent. So, a decoupled FIML would need to use the β fields of all time steps as the design variables, and the total number of the design variable can quickly become prohibitively large (e.g., millions), especially when the mesh size and the number of unsteady time steps are large. Given the above reason, we use the coupled FIML approach in this paper.

Our proposed FIML framework will conduct large-scale gradient-based optimization to find the best neural network weights and biases to minimize the regulated prediction error. In the following subsection, we will elaborate on the three main components of the FIML framework: (1) Unsteady flow simulations, (2) Built-in neural network to compute the augmentation fields, and (3) Effective adjoint gradient computation for unsteady flow. At the end of this section, we will also discuss the effective numerical configurations to facilitate the highly nonlinear coupled FIML optimization.

B. Unsteady flow simulation using the PIMPLE method

We consider incompressible, unsteady turbulent flow governed by the Navier–Stokes (N-S) equations:

$$\nabla \cdot \boldsymbol{U} = 0, \tag{1}$$

$$\frac{\partial \boldsymbol{U}}{\partial t} + (\boldsymbol{U} \cdot \nabla)\boldsymbol{U} + \nabla p - \nabla \cdot \nu_{\text{eff}}(\nabla \boldsymbol{U} + [\nabla \boldsymbol{U}]^T) = 0, \tag{2}$$

where t is the time, p is the pressure, $\boldsymbol{U}$ is the velocity vector $\boldsymbol{U} = [u, v, w]$, $\nu_{\text{eff}} = \nu + \nu_t$ with ν and ν_t being the kinematic and turbulent eddy viscosity, respectively.

As mentioned above, we solve the above N-S equations using the PIMPLE method, which is a combination of the PISO and SIMPLE algorithms. The steps are briefly summarized as follows.

First, the momentum equation is discretized, and an intermediate velocity field is solved using the pressure field obtained from the previous iteration ($p^{t-\Delta t}$) or an initial guess. Without loss of generality, we assume the first-order Euler scheme is used for temporal discretization.

$$a_P \boldsymbol{U}_P^t = -\sum_N a_N \boldsymbol{U}_N^t + \frac{\boldsymbol{U}_P^{t-\Delta t}}{\Delta t} - \nabla p^{t-\Delta t} = \boldsymbol{H}(\boldsymbol{U}) - \nabla p^{t-\Delta t}, \tag{3}$$

where a is the coefficient resulting from the finite-volume discretization, subscripts P and N denote the control volume cell and all of its neighboring cells, respectively, $\boldsymbol{U}^{t-\Delta t}$ is the velocity from the previous time step, and

$$\boldsymbol{H}(\boldsymbol{U}) = -\sum_N a_N \boldsymbol{U}_N^t + \frac{\boldsymbol{U}_P^{t-\Delta t}}{\Delta t} \tag{4}$$

represents the influence of velocity from all the neighboring cells and from the previous iteration. A new variable ϕ (face flux) is introduced to linearize the convective term:

$$\int_S \boldsymbol{U}\boldsymbol{U}\cdot \mathrm{d}\boldsymbol{S} = \sum_f \boldsymbol{U}_f\boldsymbol{U}_f\cdot\boldsymbol{S}_f = \sum_f \phi\boldsymbol{U}_f, \tag{5}$$

where the subscript f denotes the cell face. ϕ can be obtained from the previous iteration or an initial guess. Solving the momentum equation (3), we obtain an intermediate velocity field that does not yet satisfy the continuity equation.

Next, the continuity equation is coupled with the momentum equation to construct a pressure Poisson equation, and a new pressure field is computed. The discretized form of the continuity equation is

$$\int_S \boldsymbol{U}\cdot \mathrm{d}\boldsymbol{S} = \sum_f \boldsymbol{U}_f\cdot\boldsymbol{S}_f = 0. \tag{6}$$

Instead of using a simple linear interpolation, $\boldsymbol{U}_f$ in this equation is computed by interpolating the cell-centered velocity $\boldsymbol{U}_P$—obtained from the discretized momentum equation (3)—onto the cell face as follows:

$$\boldsymbol{U}_f = \left(\frac{\boldsymbol{H}(\boldsymbol{U})}{a_P}\right)_f - \left(\frac{1}{a_P}\right)_f (\nabla p)_f. \tag{7}$$

This idea of momentum interpolation was initially proposed by Rhie and Chow[52] and is effective in mitigating the oscillating pressure (checkerboard) issue resulting from the collocated mesh configuration. Substituting Eq. (7) into Eq. (6), we obtain the pressure Poisson equation:

$$\nabla\cdot\left(\frac{1}{a_P}\nabla p\right) = \nabla\cdot\left(\frac{\boldsymbol{H}(\boldsymbol{U})}{a_P}\right). \tag{8}$$

Solving Eq. (8), we obtain an updated pressure field p^t. Then, the new pressure field p^t is used to correct the face flux

$$\phi^t = \boldsymbol{U}_f\cdot\boldsymbol{S}_f = \left[\left(\frac{\boldsymbol{H}(\boldsymbol{U})}{a_P}\right)_f - \left(\frac{1}{a_P}\right)_f (\nabla p^t)_f\right]\cdot\boldsymbol{S}_f, \tag{9}$$

and velocity field

$$\boldsymbol{U}^t = \frac{1}{a_P}[\boldsymbol{H}(\boldsymbol{U}) - \nabla p^t]. \tag{10}$$

The $\boldsymbol{H}(\boldsymbol{U})$ term depends on $\boldsymbol{U}$ but has not been updated so far. To account for this, we need to repeatedly solve the Eqs. (4) to (10) (PISO corrector loop). We use two PISO corrector loops in this paper.

To connect the turbulent viscosity to the mean flow variables and close the system, a turbulence model must be used. The Spalart–Allmaras (SA) model solves:

$$\frac{\partial \tilde{\nu}}{\partial t} + \nabla \cdot (\boldsymbol{U}\tilde{\nu}) - \frac{1}{\sigma}\{\nabla \cdot [(\nu + \tilde{\nu})\nabla \tilde{\nu}] + C_{b2}|\nabla \tilde{\nu}|^2\} - \beta C_{b1}\tilde{S}\tilde{\nu} + C_{w1} f_w \left(\frac{\tilde{\nu}}{d}\right)^2 = 0. \tag{11}$$

where $\tilde{\nu}$ is the modified viscosity, which can be related to the turbulent eddy viscosity via

$$\nu_t = \tilde{\nu}\frac{\chi^3}{\chi^3 + C_{v1}^3}, \quad \chi = \frac{\tilde{\nu}}{\nu}. \tag{12}$$

Refer to Spalart and Allmaras[53] for a more detailed description of the terms and parameters in the SA model. As mentioned above, we multiply the production term by an augmentation field β.

In addition to the PISO corrector loop mentioned above, the PIMPLE algorithm repeatedly solves Eqs. (3) to (11) multiple times until all the flow residuals are small (PIMPLE corrector loop). To ensure the PIMPLE stability, we need to under-relax the momentum equation (3) and turbulence equation (11) solutions and the pressure update after solving the pressure Poisson equation (8), except for the last PIMPLE corrector loop. We use Eqs. 3, 8, 9, and 11 for the residuals of velocity, pressure, face flux, and turbulence variables, respectively. The PIMPLE method allows us to use a relatively large time step size (CFL>1). This feature may not directly benefit the unsteady flow simulation speed, because we need multiple PIMPLE iterations to converge the flow residual at each time step. However, using a relatively large time step is highly desirable for the unsteady adjoint solver because (1) we need to solve a much smaller number of adjoint equations, and (2) we need to read and write a much smaller amount of intermediate flow data to the disk. In this study, we run the PIMPLE corrector loop until all the flow residuals drop 8 orders of magnitude or a maximal of 100 PIMPLE corrector iterations. Note that converging the flow residuals as tightly as possible is critical for the accuracy of the adjoint method because our proposed unsteady adjoint uses a residual-based formulation (see the details in the next subsection). The PISO algorithm, which is commonly used for solving incompressible unsteady flow, does not repeatedly solve the momentum and pressure equations. Therefore, its flow residuals are not as tightly converged as the PIMPLE's.

C. Built-in neural network model to compute the augmentation field

As mentioned above, we incorporate a multilayer perceptron (MLP) neural network model into the CFD solver (Fig. 3). The inputs are the local flow features η, and the output is the augmentation

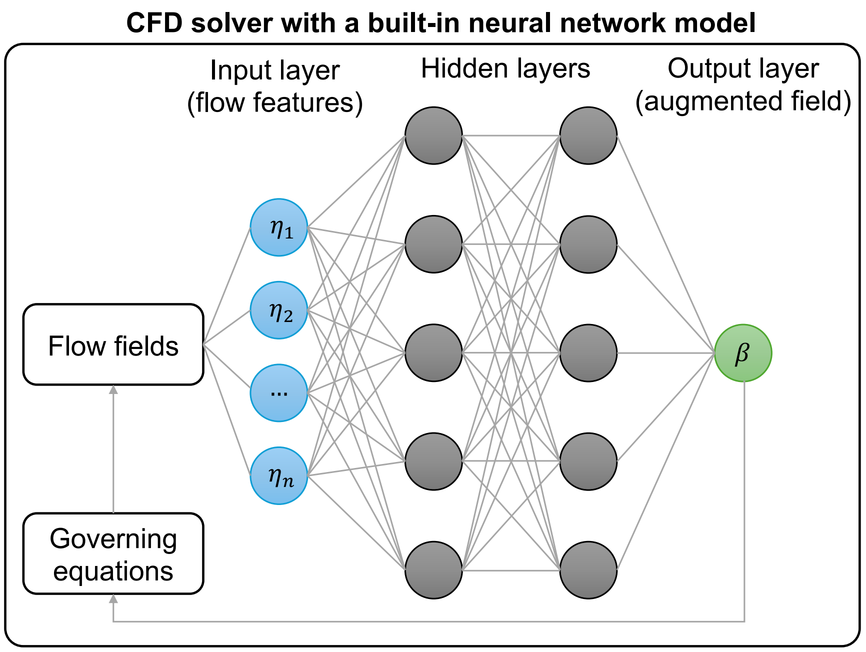

FIG. 3: Schematic of the solver-built-in, multilayer perceptron neural network model used in this paper.

field β for each iteration. The MLP has multiple layers of fully connected neurons (hidden layers) between the input and output layers. Each neuron can be computed as a weighted summation of the neuron from the previous layer, nested in a nonlinear activation function

$$x_i^k = f_a(b_i + \sum_{j=1}^{N_{k-1}} w_j x_j^{k-1}), \tag{13}$$

where w and b are the weights and biases, respectively. The superscripts k and $k-1$ denote the current and previous layers, respectively. N_{k-1} is the total number of neuron of the $k-1$ layer. Various options can be used for the activation function f_a, such as tanh, sigmoid, and ReLU. We choose the hyperbolic tangent activation function (tanh) in the form of

$$f_a(y) = \frac{1-e^{-2y}}{1+e^{-2y}}. \tag{14}$$

As will be shown later, we find the optimal weight and biases using a sequential quadratic programming (SQP) algorithm, which uses second-order derivatives to compute search directions. Compared with ReLU, the tanh activation function is twice differentiable and should be more compatible with SQP. In addition, we will use initial weights and biases that are close to zeros to start the optimization. The tanh activation function's gradients are larger than the sigmoid's at the starting point, so it can converge to the optimal point faster. A comprehensive evaluation of various activation functions and optimization algorithms is beyond the scope of this paper and will be conducted in our future work.

Our FIML framework has implemented many flow features, including the vorticity-to-strain ratio, turbulence production-to-destruction ratio, turbulence convection-to-production ratio normalized turbulence viscosity, pressure gradient along the streamline, pressure stress to normal stress ratio, streamline curvature, velocity orthogonality, normalized turbulence intensity, wall distance based Reynolds number, and total-to-normal Reynolds stress ratio[54]. In practice, users may need to explore various combinations of flow features to obtain the best neural network performance. All features are normalized to maximize the FIML's generalizability for unseen flow conditions. To facilitate the optimization, users can also prescribe scaling factors for the flow features such that their standard deviations (across all mesh cells and time instances) are close to one. In addition to the input layer and output layer, users can add any number of hidden layers and the number of neurons for each hidden layer. All the layers have the hyperbolic tangent activation function except for the output layer.

D. PIMPLE-Krylov unsteady adjoint formulation for gradient computation

As mentioned above, the PIMPLE flow simulation converges all flow residuals tightly at each time step.

$$\boldsymbol{R}(\boldsymbol{x},\boldsymbol{w}) = \begin{bmatrix} \boldsymbol{R}^1(\boldsymbol{x},\boldsymbol{w}^1,\boldsymbol{w}^0) \\ \boldsymbol{R}^2(\boldsymbol{x},\boldsymbol{w}^2,\boldsymbol{w}^1,\boldsymbol{w}^0) \\ \vdots \\ \boldsymbol{R}^K(\boldsymbol{x},\boldsymbol{w}^K,\boldsymbol{w}^{K-1},\boldsymbol{w}^{K-2}) \end{bmatrix} = \boldsymbol{0}, \tag{15}$$

where the superscript denotes the time step index with K being the total number of time steps, $\boldsymbol{x} \in \mathbb{R}^{n_x}$ is the design variable vector with n_x being the total number of design variables, $\boldsymbol{w} \in \mathbb{R}^{Kn_w}$ is the state variable vector with n_w being the total number of state variable for each time step, and $\boldsymbol{R} \in \mathbb{R}^{Kn_w}$ is the flow residual vector. Here we use an implicit second-order time discretization scheme for all time steps except for the first one, where the first-order time scheme is used. In the primal unsteady flow solution, Eq. (15) is solved in a forward fashion to determine the state variable for all time steps, i.e., $\boldsymbol{w}^1,\boldsymbol{w}^2,\ldots,\boldsymbol{w}^K \in \mathbb{R}^{n_w}$.

The objective function F also depends on both the design variables $\boldsymbol{x}$ and the state variable $\boldsymbol{w}$

solved in Eq. (15), and in many applications, including this study, the objective function F can be expressed as the average of a time series, that is:

$$F(\boldsymbol{x},\boldsymbol{w}) = \frac{1}{K}\sum_{i=1}^{K} f^i(\boldsymbol{x},\boldsymbol{w}^i), \tag{16}$$

where for each $1 \le i \le K$, f^i only depends on the design variables $\boldsymbol{x}$ and the corresponding state at the current time step $\boldsymbol{w}^i$. This implies that the partial derivative $\partial F/\partial \boldsymbol{w}$ can be simplified as:

$$\underbrace{\frac{\partial F}{\partial \boldsymbol{w}}}_{1\times Kn_w} = \frac{1}{K}[\underbrace{\frac{\partial f^1}{\partial \boldsymbol{w}^1}}_{1\times n_w}, \underbrace{\frac{\partial f^2}{\partial \boldsymbol{w}^2}}_{1\times n_w}, \cdots, \underbrace{\frac{\partial f^K}{\partial \boldsymbol{w}^K}}_{1\times n_w}]. \tag{17}$$

For other common types of objective functions, e.g., the variance of a time series, the partial derivatives of F can also be simplified in a similar manner.

To obtain the total derivative $\mathrm{d}F/\mathrm{d}\boldsymbol{x}$ for gradient-based optimization, we apply the chain rule as follows:

$$\underbrace{\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{x}}}_{1\times n_x} = \underbrace{\frac{\partial F}{\partial \boldsymbol{x}}}_{1\times n_x} + \underbrace{\frac{\partial F}{\partial \boldsymbol{w}}}_{1\times Kn_w} \underbrace{\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}}}_{Kn_w\times n_x}, \tag{18}$$

where the partial derivatives $\partial F/\partial \boldsymbol{x}$ and $\partial F/\partial \boldsymbol{w}$ are relatively cheap to evaluate because they only involve explicit computations. The total derivative $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$ matrix, on the other hand, is expensive, because $\boldsymbol{w}$ and $\boldsymbol{x}$ are implicitly linked by the residual equations $\boldsymbol{R}(\boldsymbol{w},\boldsymbol{x}) = 0$.

To solve for $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$, we can apply the chain rule for $\boldsymbol{R}$. We then use the fact that the governing equations should always hold, independent of the values of design variables $\boldsymbol{x}$. Therefore, the total derivative $\mathrm{d}\boldsymbol{R}/\mathrm{d}\boldsymbol{x}$ must be zero:

$$\frac{\mathrm{d}\boldsymbol{R}}{\mathrm{d}\boldsymbol{x}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}} = 0. \tag{19}$$

Rearranging the above equation, we get the linear system

$$\underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}}_{Kn_w\times Kn_w} \cdot \underbrace{\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}\boldsymbol{x}}}_{Kn_w\times n_x} = - \underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}}_{Kn_w\times n_x}. \tag{20}$$

We can then substitute the solution for $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$ from Eq. (20) into Eq. (18) to get

$$\underbrace{\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{x}}}_{1\times n_x} = \underbrace{\frac{\partial F}{\partial \boldsymbol{x}}}_{1\times n_x} - \overbrace{\underbrace{\frac{\partial F}{\partial \boldsymbol{w}}}_{1\times Kn_w} \underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^{-1}}_{Kn_w\times Kn_w}}^{\psi^T} \underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}}_{Kn_w\times n_x}. \tag{21}$$

Now we can transpose the Jacobian and solve with $[\partial F/\partial \boldsymbol{w}]^T$ as the right-hand side, which yields the *adjoint equation*,

$$\underbrace{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{w}}^T}_{Kn_w\times Kn_w} \cdot \underbrace{\boldsymbol{\psi}}_{Kn_w\times 1} = \underbrace{\frac{\partial F}{\partial \boldsymbol{w}}^T}_{Kn_w\times 1}, \tag{22}$$

where $\boldsymbol{\psi}$ is the *adjoint vector*. Then, we can compute the total derivative by substituting the adjoint vector into Eq. (21):

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{x}} = \frac{\partial F}{\partial \boldsymbol{x}} - \boldsymbol{\psi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}. \tag{23}$$

Since the design variables are not explicitly present in Eq. (22), we need to solve the adjoint equation only once for each objective function. Therefore, the computational cost is independent of the number of design variables but proportional to the number of objective functions. This approach of computing derivatives introduced so far is also known as the *adjoint method*. It is advantageous for field inversion because typically, there is only one objective function, but thousands of design variables may be used.

The adjoint equation in Eq. (22) can be simplified for the time-marching primal problem. As indicated in Eq. (15), for each $1 \le i \le K$, $\boldsymbol{R}^i$ has dependency only on $\boldsymbol{x}$, $\boldsymbol{w}^i$, $\boldsymbol{w}^{i-1}$, and $\boldsymbol{w}^{i-2}$. Together with the simplification in Eq. (17), Eq. (22) can be rewritten as:

$$\begin{bmatrix} \frac{\partial \boldsymbol{R}^1}{\partial \boldsymbol{w}^1}^T & \frac{\partial \boldsymbol{R}^2}{\partial \boldsymbol{w}^1}^T & \frac{\partial \boldsymbol{R}^3}{\partial \boldsymbol{w}^1}^T & & \\ & \frac{\partial \boldsymbol{R}^2}{\partial \boldsymbol{w}^2}^T & \frac{\partial \boldsymbol{R}^3}{\partial \boldsymbol{w}^2}^T & \frac{\partial \boldsymbol{R}^4}{\partial \boldsymbol{w}^2}^T & \\ & & \ddots & \ddots & \\ & & & \frac{\partial \boldsymbol{R}^{K-1}}{\partial \boldsymbol{w}^{K-1}}^T & \frac{\partial \boldsymbol{R}^K}{\partial \boldsymbol{w}^{K-1}}^T \\ & & & & \frac{\partial \boldsymbol{R}^K}{\partial \boldsymbol{w}^K}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{\psi}^1 \\ \boldsymbol{\psi}^2 \\ \vdots \\ \boldsymbol{\psi}^{K-1} \\ \boldsymbol{\psi}^K \end{bmatrix} = \frac{1}{K} \begin{bmatrix} \frac{\partial f^1}{\partial \boldsymbol{w}^1}^T \\ \frac{\partial f^2}{\partial \boldsymbol{w}^2}^T \\ \vdots \\ \frac{\partial f^{K-1}}{\partial \boldsymbol{w}^{K-1}}^T \\ \frac{\partial f^K}{\partial \boldsymbol{w}^K}^T \end{bmatrix}, \tag{24}$$

where the adjoint vector $\boldsymbol{\psi} \in \mathbb{R}^{Kn_w}$ is broken down into K parts that correspond to the time steps, i.e., $\boldsymbol{\psi}^1, \boldsymbol{\psi}^2, \ldots, \boldsymbol{\psi}^K \in \mathbb{R}^{n_w}$. Then, Eq. (24) can be solved sequentially in a backward fashion as:

$$\begin{aligned} &\frac{\partial \boldsymbol{R}^K}{\partial \boldsymbol{w}^K}^T \cdot \boldsymbol{\psi}^K = \frac{1}{K}\frac{\partial f^K}{\partial \boldsymbol{w}^K}^T, \\ &\frac{\partial \boldsymbol{R}^{K-1}}{\partial \boldsymbol{w}^{K-1}}^T \cdot \boldsymbol{\psi}^{K-1} = \frac{1}{K}\frac{\partial f^{K-1}}{\partial \boldsymbol{w}^{K-1}}^T - \frac{\partial \boldsymbol{R}^K}{\partial \boldsymbol{w}^{K-1}}^T \cdot \boldsymbol{\psi}^K, \\ &\frac{\partial \boldsymbol{R}^i}{\partial \boldsymbol{w}^i}^T \cdot \boldsymbol{\psi}^i = \frac{1}{K}\frac{\partial f^i}{\partial \boldsymbol{w}^i}^T - \frac{\partial \boldsymbol{R}^{i+1}}{\partial \boldsymbol{w}^i}^T \cdot \boldsymbol{\psi}^{i+1} - \frac{\partial \boldsymbol{R}^{i+2}}{\partial \boldsymbol{w}^i}^T \cdot \boldsymbol{\psi}^{i+2},\ K-2 \ge i \ge 1, \end{aligned} \tag{25}$$

which effectively breaks down the original adjoint equation in Eq. (22) into K much smaller sub-equations. The right-hand side terms in Eq. (25) can be efficiently evaluated with reverse-mode automatic differentiation (AD). In particular, the matrix-transpose-vector product $[\partial \boldsymbol{R}^{i+2}/\partial \boldsymbol{w}^i]^T \boldsymbol{\psi}^{i+2}$

is evaluated in a Jacobian-free manner right after we solve for $\boldsymbol{\psi}^{i+2}$, and $[\partial \boldsymbol{R}^{i+1}/\partial \boldsymbol{w}^{i}]^T \boldsymbol{\psi}^{i+1}$ is evaluated in a similar fashion, then they are passed to the right-hand side of the sub-equation for $\boldsymbol{\psi}^{i}$. Note that we solve the above adjoint equation (25) in reverse and computing the matrix-vector products requires access to state variables for all time steps. Saving all state variables in memory is prohibitively expensive. Therefore, we write the state variables to the disk for all time steps during the primal simulation. Then, during the adjoint computation, we read the state variables for each time step. The use of file IO in our unsteady adjoint method is acceptable because (1) OpenFOAM has a parallel file IO interface that scales well with large meshes, and (2) the file IO runtime is much smaller than the actual adjoint computation runtime.

The assumption that the objective function F is of the average type in Eq. (16) also simplifies the expression of the total derivative $\mathrm{d}F/\,\mathrm{d}\boldsymbol{x}$ in Eq. (23) as:

$$\frac{\mathrm{d}F}{\mathrm{d}\boldsymbol{x}} = \sum_{i=1}^{K} \left(\frac{1}{K}\frac{\partial f_i}{\partial \boldsymbol{x}} - \boldsymbol{\psi}^{i^T}\frac{\partial \boldsymbol{R}^i}{\partial \boldsymbol{x}}\right). \tag{26}$$

Therefore, we can calculate the total derivative accumulatively as we sequentially solve the sub-equations in Eq. (25). Computing the total derivatives on the fly is desirable because we do not need to save the adjoint vectors for all time steps.

Finally, we elaborate on how to effectively solve the adjoint linear equations in Eq. (25). We use the generalized minimal residual (GMRES) solver from the Portable, Extensible Toolkit for Scientific Computation (PETSc)[55] library to solve the adjoint linear equations. The GMRES solver converges quadratically and is significantly faster than the fixed-point adjoint solver (linear convergence) used in our previous studies[48,56,57]. In addition, the GMRES solver's convergence does not require the linear iteration matrix's eigenvalues to be within the unit circle. It is more robust in practice, especially when we need to repeatedly solve the adjoint equations.

We use the GMRES as the top-level solver with a nested preconditioning strategy. For the global preconditioner, we use a one-level-overlap additive Schwartz method (ASM). The ASM approach decomposes the linear equation into sub-blocks and allows us to solve it in parallel. For the local preconditioner in each sub-block, we use the incomplete lower and upper (ILU) factorization approach with one level of fill-in.

We develop a Jacobian-free GMRES approach to solve the adjoint linear equations, similar to that proposed in our previous work for steady-state problems[58]. The Jacobian-free GMRES can directly compute the matrix-vector products to construct the Krylov subspace without forming or saving $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$. The benefit of Jacobian-free GMRES is that it saves the computational time and memory involved in computing and storing $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T$.

To make the GMRES solver efficient, we need a strong preconditioner to improve the eigenvalues clustering. The right-preconditioned adjoint equations reads

$$\left(\frac{\partial \boldsymbol{R}^i}{\partial \boldsymbol{w}^i}^T \left[\frac{\partial \boldsymbol{R}^i}{\partial \boldsymbol{w}^i}^T_{PC}\right]^{-1}\right)\left(\frac{\partial \boldsymbol{R}^i}{\partial \boldsymbol{w}^i}^T_{PC}\psi^i\right) = \frac{\partial f^i}{\partial \boldsymbol{w}^i}^T, \tag{27}$$

where $[\partial \boldsymbol{R}^i/\partial \boldsymbol{w}^i]^T_{PC}$ is the preconditioner matrix for the *i*th time step, and the superscript -1 denotes an approximated inverse. The preconditioner matrix should be an approximation of $[\partial \boldsymbol{R}^i/\partial \boldsymbol{w}^i]^T$ but easily invertible. To this end, we use the first-order upwind scheme to compute the convective term of the momentum equation (2), which effectively reduces the preconditioning matrix's stiffness. To reduce the matrix bandwidth, we shrink the pressure and face flux residual stencils by reducing the maximal level of connected states by one. We scale the preconditioner matrix for better diagonal dominance. This is achieved by normalizing the residuals (Jacobian's rows) by the cell volume or face area and states (Jacobian's columns) by their corresponding reference values at the far field. $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ is a large sparse matrix. To efficiently compute it, we use the finite-difference method with a heuristic graph coloring algorithm, as proposed in our previous work[9].

For steady-state problems, the converged flow fields typically stay similar throughout the optimization iterations. Therefore, we can compute the above preconditioner matrix only once using the flow fields of the baseline design and reuse it for all following optimization iterations. This aforementioned strategy needs to be modified for unsteady problems, because the transient flow may undergo significant changes in space and time, and the flow fields at the end of an unsteady simulation can be significantly different from those at the beginning. We need to have an appropriate preconditioner for each time step to ensure fast GMRES convergence. Naively recomputing the preconditioner for each time step is prohibitively expensive (computing $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ may be much more expensive than solving the adjoint equation for one time step). Therefore, we develop a two-level preconditioner method for the PIMPLE-Krylov adjoint solver, as shown in Fig. 4.

On the top level, we pre-compute the full preconditioner matrices at selected time steps and save them in memory. For example, for an unsteady simulation with non-dimensional time t^* from 0 to 10, we may pre-compute four $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ matrices using the flow state variables at $t^* =$ 10, 7.5, 5, and 2.5, respectively. During the reverse unsteady adjoint solution process, we will use the first $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ matrix for $7.5 < t^* \leq 10$. We will switch to the second $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ matrix

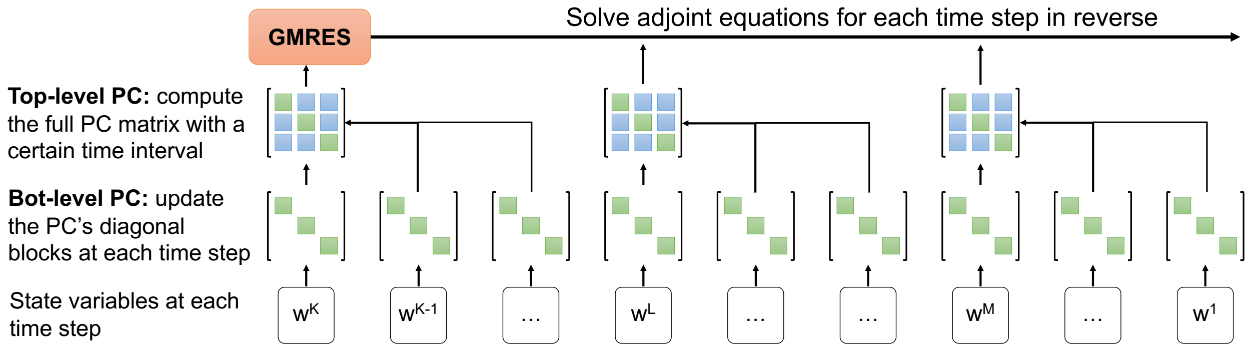
GMRES
Solve adjoint equations for each time step in reverse
Top-level PC: compute the full PC matrix with a certain time interval
Bot-level PC: update the PC's diagonal blocks at each time step
State variables at each time step
w^K
w^K-1
...
w^L
...
...
w^M
...
w^1

FIG. 4: Schematic of the proposed two-level GMRES preconditioner computation method for accelerating unsteady adjoint equation solution.

for $5 < t^* \leq 7.5$, and similarly for the other intervals. This strategy ensures each adjoint solution time interval has an effective preconditioner matrix, which is computed using flow states similar to that interval.

Using the above top-level preconditioning strategy alone does not always guarantee fast adjoint equation solutions. This is because the flow may still undergo significant changes during a smaller time interval. To address this issue, we develop a bottom-level approach that updates part of the $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ matrix (diagonal blocks) at each time step. The bottom-level update needs to be fast and the heuristic graph coloring approach mentioned above is not computationally feasible. Therefore, we reuse the inner iteration coefficient matrices from the primal flow solution process to construct the bottom-level $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ updates:

$$\begin{bmatrix} {\boldsymbol{A}^i_U}^T & & & \\ & {\boldsymbol{A}^i_p}^T & & \\ & & -\boldsymbol{I}_\phi & \\ & & & {\boldsymbol{A}^i_{\tilde{\nu}}}^T \end{bmatrix} \rightarrow \frac{\partial {\boldsymbol{R}^i}^T}{\partial \boldsymbol{w}^i}_{PC} \tag{28}$$

where ${\boldsymbol{A}^i_U}^T$, ${\boldsymbol{A}^i_p}^T$, and ${\boldsymbol{A}^i_{\tilde{\nu}}}^T$ are the transpose of the left-hand-side matrices for the velocity (Eq. (3)), pressure (Eq. (8)), and turbulence variable (Eq. (11)) equations, respectively, and $\boldsymbol{I}_\phi$ is the unit vector for the face flux residual equation (9). Because unsteady primal solvers typically have built-in interfaces (e.g., the `fvMatrix` in OpenFOAM) that can readily assemble $\boldsymbol{A}^i_U$, $\boldsymbol{A}^i_p$, and $\boldsymbol{A}^i_{\tilde{\nu}}$, we need to only transpose these matrices and update $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ with their values. This update is orders of magnitude faster than the top-level full $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ matrix computation.

In summary, the top-level preconditioner computation recomputes the whole $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ matrix with a given time interval, while the bottom-level computation updates the $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ matrix's diagonal blocks at each time step. We compute the top-level preconditioner matrices only once from the baseline flow fields and reuse them throughout the entire optimization. On the other hand, we update the bottom-level preconditioner matrix at each time step and each optimization iteration. We find that the above preconditioner computation strategy is effective in converging the unsteady adjoint equations for all time steps. In practice, we may need to tweak the top-level $[\partial \boldsymbol{R}/\partial \boldsymbol{w}]^T_{PC}$ computation interval to achieve the optimal adjoint computation speed.

Our proposed PIMPLE-Krylov adjoint method exhibits reasonable speed, memory usage, scalability, and accuracy, as shown in Appendix A. The adjoint solver's runtime is about three times as large as the flow's runtime, and the average error in the adjoint derivatives is about 1%. This level of performance is acceptable for practical unsteady gradient-based optimization for coupled FIML problems.

Note that our usage of the adjoint method in data-driven unsteady flow modeling is conceptually similar to the study by Wang, Wang, and Zaki[59]. They developed a discrete adjoint formulation for a fractional time step unsteady flow solver. Instead of using it for correcting the turbulence model's defects, they conducted a data-assimilation study to recover the flow initial condition for circular Couette flows.

E. Effective numerical configurations for highly nonlinear coupled FIML

As mentioned above, one disadvantage of incorporating a highly nonlinear neural network model in FIML is that it makes the gradient-based optimization more challenging to converge. This subsection discusses our numerical configurations to facilitate the coupled FIML optimization.

Flow features and scaling. The selection of flow features has a large impact on the success of coupled FIML optimization. While previous studies have discussed how to choose flow features based on physical insights (to name a few[16,60]), there has not been a universal set of flow features that are guaranteed to work for every case. Therefore, we may still need trial and error in practice to get the optimal FIML performance. In addition to selecting flow features, setting proper scaling factors is critical for gradient-based optimization in coupled FIML. To maximize the prediction generalizability, the flow features are typically normalized, non-dimensional variables. However, additional scaling is still needed, especially for unsteady flow problems where local flow features' magnitude may vary significantly in time. For example, the turbulence production may be relatively small when the boundary layer is attached. When the flow further evolves in time and the boundary layer separates, the production term may increase rapidly. For decoupled FIML, the scaling is typically set in a neural network package (e.g., TensorFlow) offline; however, one needs to set the scaling in DAFoam on the fly for coupled FIML. Therefore, we typically run a test primal simulation and monitor the temporal evolution of all flow features. Then, one can set proper scaling factors for the flow features and make their standard deviations as close to each other as possible during the entire primal simulation.

Initial design variable values. Choosing initial design variable values is not an issue for the field inversion part of decoupled FIML because the most reasonable choice is to set them (augmentation field variables) to be ones, meaning no augmentation to the original turbulence model. However, this choice is not feasible for coupled FIML problems because the design variables are the weights and biases of a neural network model, which do not have clear physical meanings. To alleviate this issue, we offset the neural network's output by one such that setting all zeros as the inputs (weights and biases) will lead to ones as the output (augmentation field). However, this aforementioned setting (all zeros as the inputs) will create an issue for gradient-based optimization because the derivatives of the output with respect to the inputs are all zeros except for the output layer's bias (refer to Eqs. 13 and 14). To fix this problem, we randomize the input variables within a prescribed small bound, i.e., $[-0.05$ to $0.05]$. This setting will make the initial augmentation field variables close to ones while having non-zero derivatives to start the gradient-based optimization. In practice, we find that the coupled-FIML optimization's convergence strongly depends on the randomized initial design variable values. Therefore, trial and error is needed by prescribing different bounds and random variable seeds.

Neural network architecture. Similar to decoupled-FIML problems, one needs to explore various neural network architectures (e.g., the number of hidden layers and neurons for each layer) to maximize the performance for coupled-FIML problems. However, the extra challenge is that one needs to re-solve the entire FIML problem for each architecture configuration. The high computational cost prevents us from exploring many architectures. In addition, one has much less freedom in choosing the desired architecture. This is because the neural network model will be called for each PIMPLE iteration (Fig. 2), so increasing the number of hidden layers and neurons will significantly increase the computational cost for the coupled FIML. It will also increase the memory cost because DAFoam uses an operator-overloading AD tool to compute the matrix-vector products during the adjoint solution process. Therefore, a large neural network model requires saving many more intermediate variable values in the memory for each matrix-vector product calculation. We suggest conducting trial and error to optimize the number of hidden layers and neurons in practice, starting from a small neural network.

Objective function weights. As will be elaborated on in the next section, our objective function consists of two terms: a prediction error term and a regulation term. Similar to the decoupled FIML, the relative weight between these two terms is important for the FIML optimization convergence and its prediction accuracy for unseen conditions. Setting a too-small weight for the regulation term can help reduce the objective function value in gradient-based optimization and make it converge tighter. However, the neural network model may be overfitting by using highly non-uniform augmentation fields to correct the prediction for the training data while having poor prediction accuracy for unseen conditions. On the other hand, setting a too-large regulation weight will significantly limit the design freedom to correct the imperfect turbulence model. In practice, we need to explore various weights to balance the FIML optimization progress and the prediction performance for unseen conditions.

Invalid output from the neural network. For decoupled FIML problems, the optimizer can strictly control the bound of the augmentation field variables because they are the design variables. However, for coupled-FIML problems, the weights and biases are the design variables and the augmentation field variables become intermediate variables (output of the neural network model) that cannot be directly controlled by the optimizer. Although one can set a nonlinear constraint to limit the augmentation variables in coupled FIML, the optimizer may still explore unphysical augmentation variables during the line search process. For example, the optimizer may choose an unreasonable combination of weights and biases such that the computed augmentation variable is much larger or smaller than its normal values (e.g., > 100). In addition, some flow features may not be well defined (e.g., divided by zero) at some local cells during PIMPLE iterations, so the computed augmentation variable will be invalid. Without proper treatments, the above scenarios may cause segmentation faults for the primal solver and abort the optimization. To avoid this, we add a safeguard function in the primal solution process. This function will evaluate if the augmentation field variables are valid for each unsteady time step. If any invalid augmentation field variable is found at a time step, we will inform the optimizer that the primal solution fails and ask it to backtrack the line search for a better design. This treatment significantly improves the robustness of the coupled-FIML optimization.

Optimizers and their configurations. One of the benefits of coupled FIML is that the number of design variables does not typically exceed a few thousand. However, for the decoupled approach, hundreds of thousands of design variables may be needed, depending on the mesh size. Therefore, the coupled approach is more flexible in choosing gradient-based optimizers, many of which may not be specially designed to scale well with a massive number of design variables. In theory, the coupled-FIML framework works with many optimizers, such as SLSQP[61], IPOPT[62], and SNOPT[63]. However, we only tested the framework using SNOPT. We found that using a loose tolerance for the line search (e.g., 0.999) and disabling the Hessian update helped the optimization convergence. More detailed discussion on various optimizer configurations can be found in Rojas-Labanda and Stolpe[64].

III. RESULTS AND DISCUSSION

In this section, we first describe the unsteady CFD simulations over a 45-degree ramp and compare the simulated flow fields between the original SA model and reference $k-\omega$ SST model. Then, we conduct an unsteady FIML to augment the SA model using only the surface pressure at the bottom wall as training data. We will evaluate the trained model's prediction accuracy for surface pressure and other variables not used in training, i.e., velocity fields. In addition, we will evaluate the trained model's performance for unseen geometries (different ramp angles) and flow conditions (different Reynolds numbers). Finally, we will justify the need for unsteady FIML by demonstrating that a steady-flow-trained model cannot accurately predict the time-accurate unsteady flow.

A. Time-accurate CFD simulations for unsteady flow over a 45-degree ramp

As mentioned before, we use unsteady turbulent flow over a ramp as the benchmark, as shown in Fig. 5. The inlet and outlet heights are 0.5 and 1.0 m, respectively, and the ramp length is 3.0 m. The ramp angle is 45°, and the top surface is a symmetry plane. We generate a structured mesh with 20,000 cells, and the maximal y^+ for the bottom wall is less than 1. The inlet velocity is $U_0 = 10$ m/s and the corresponding Reynolds number (based on the outlet height h) is 10^5. Choosing an appropriate initial flow field is important for unsteady FIML because we need to compute flow features at each time step. Naively using a uniform flow field to start unsteady flow simulations and FIML may result in ill-defined flow features (e.g., the denominator being

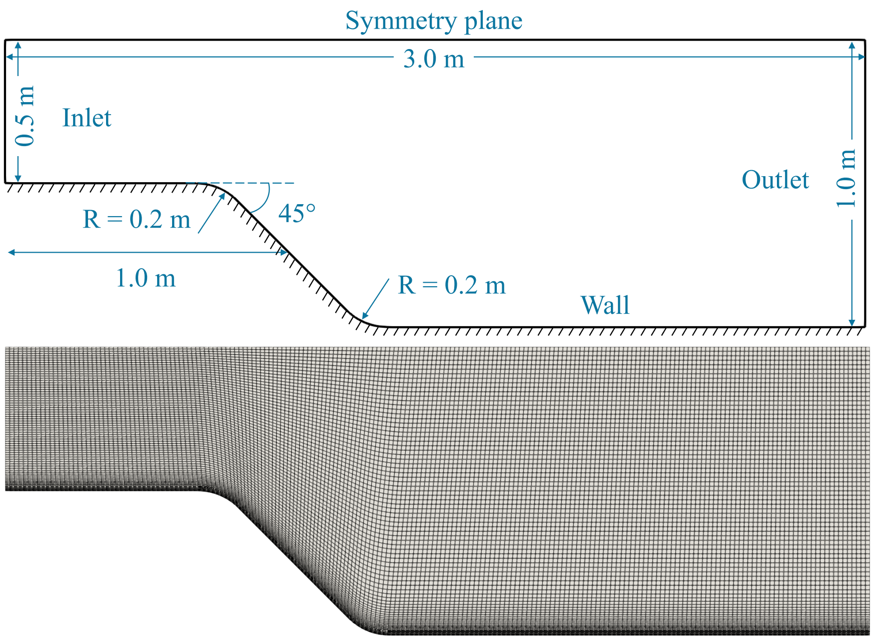

FIG. 5: Geometry, boundary conditions, and mesh for the unsteady-flow-over-ramp benchmark. The coordinate origin ($[x,y] = [0,0]$) is located at the bottom of the inlet boundary, where x and y are the streamwise and vertical directions, respectively.

zero). To avoid this, we use a uniform flow field of 10 m/s as the initial condition and run the flow solver for 0.1 s, or a non-dimensional time $t^* = tU_0/h$ of 1, to build up the boundary layer in the channel. Then, we use the above spun-up flow field as the initial condition and run unsteady simulations for 10 more non-dimensional time units. We use the flow data from $t^* = 0$ to 10 for the unsteady FIML training. The non-dimensional time step size is $\Delta t^* = 0.05$, and the corresponding CFL number is about 10. As mentioned previously, the PIMPLE algorithm allows stable unsteady simulations with CFL > 1. Using a relatively large CFL number should not significantly degrade the accuracy of unsteady RANS simulation results. This is because the temporal evolution of flow is not as significant as eddy-resolving simulations such as large-eddy or direct numerical simulation (LES/DNS), so the discretization error for the time derivative in RANS models is not sensitive to the time step size.

In this study, we use the unsteady flow fields computed by the $k-\omega$ SST turbulence model as the reference. We then augment the SA model and make its predictions match the SST's. Note that in practical FIML, it is more common to use either experimental or eddy-resolving high-fidelity (LES/DNS) data as the reference values. To simplify the analysis (e.g., mesh interpolation), this paper uses the SST model's simulation data as a reference to demonstrate the proposed unsteady

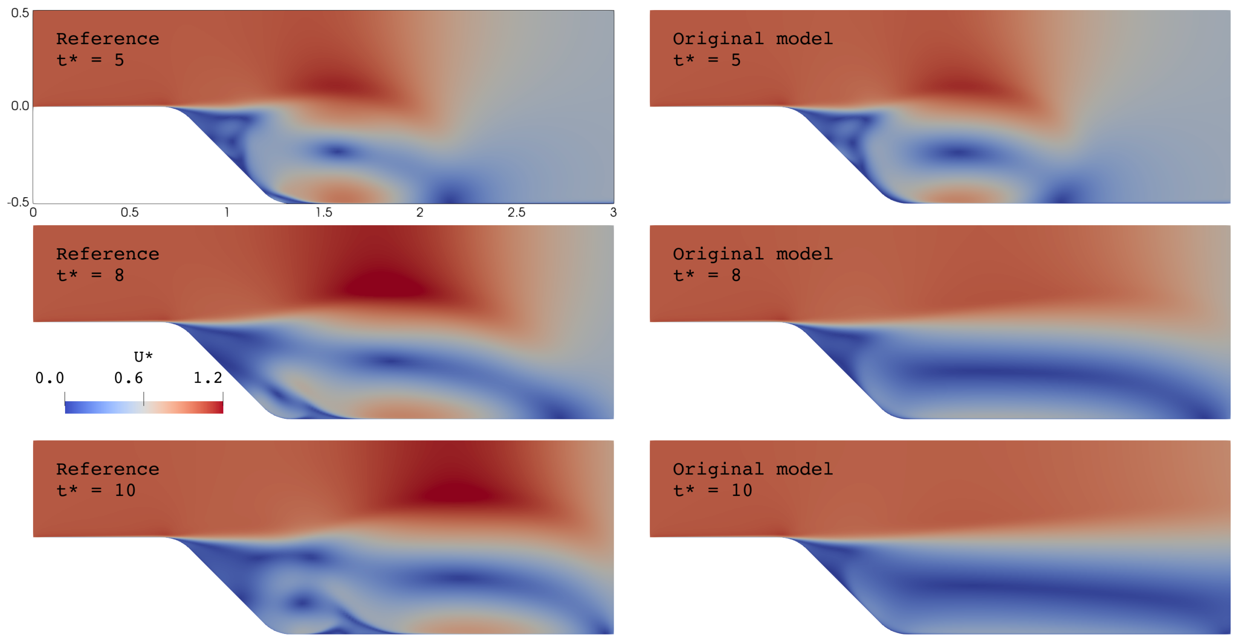

FIG. 6: Velocity magnitude contours at various times. Left: reference data generated by the $k-\omega$ SST model. Right: data generated by the original SA model. The original and reference models have similar flow fields at $t^* = 5$. However, they exhibit significantly different vortex structures at $t^* = 8$ and 10.

FIML framework. The proposed FIML can be easily extended to use experimental or LES/DNS data.

Figure 6 shows the comparison of velocity magnitude contours between the original SA model prediction and SST reference data at various times. The flow separates behind the ramp and forms a vortex downstream, which can be seen clearly at $t^* = 5$. The SA model and SST model predict similar ramp main vortex structures at $t^* = 5$ and $[x^*, y^*] = [1.5, -0.25]$, where $x^* = x/h$ and $y^* = y/h$. However, their predictions diverge quickly as the unsteady simulation goes further. The main vortex predicted by the original SA model dissipates quickly, and the velocity distribution in the wake area becomes relatively uniform at t^* =8 and 10. In contrast, the SST model predicts much more intriguing vortex structures and interactions. The ramp main vortex propagates downstream, and its strength does not dissipate as quickly as the SA's prediction at $t^* = 8$. Then, the main vortex interacts with the bottom wall and rolls up a secondary vortex at $t^* = 10$ and $[x^*, y^*] = [1.2, -0.3]$ (Fig. 6 bot-right). The above observations are consistent with previous work[65–68], where the SA model was found to predict significantly different flow fields and vortex structures than the SST model for separated flow.

B. Unsteady FIML for improving time-accurate unsteady flow predictions

The goal of this paper is to demonstrate the unsteady FIML framework by augmenting the original SA model and making it predict similar vortex structures (both spatially and temporally) as the SST model. To this end, we create a composite objective function F as follows:

$$F = \frac{1}{K}\sum_{t=1:K}[\frac{c_1}{N_i}\sum_{i=1:N_i}(f_{i,\,t}^{\text{CFD}} - f_{i,\,t}^{\text{ref}})^2 + \frac{c_2}{N_j}\sum_{j=1:N_j}(\beta_{j,\,t} - 1)^2] \tag{29}$$

where the subscript t denotes the time index with K being the number of time steps, β is the augmentation scalar field to the SA model's production term, Eq. (11), f^{CFD} could be any quantity computed by CFD, and f^{ref} is the corresponding reference value (also known as training data). This paper uses the pressure at the bottom wall as f. So, the subscript i denotes the ith mesh face, and N_i is the total number of mesh faces on the bottom wall. The error between SA and SST is quantified as the summation of bottom wall pressure at all surface mesh faces and all time steps (the first term on the right-hand side). To avoid over-fitting, we also add a regulation term to minimize the spatial variations of the augmentation scalar field β with respect to its original value (1.0). So, the subscript j is the mesh cell index with N_j being the total number of mesh cells. $c_1 = 0.02$ and $c_2 = 0.01$ are the weights for the two terms in the objective. c_1 is mainly used to scale F to be close to 1, and c_2 is mainly used to control the regulation.

Both f and β are implicit functions of the flow state variables $\boldsymbol{w}$ and design variable x (weights and biases). To minimize the composite objective function F, we run gradient-based optimization using the sparse nonlinear optimizer (SNOPT[63]). We compute the gradients using the proposed PIMPLE-Krylov adjoint method. As mentioned previously, the PIMPLE-Krylov method solves adjoint equations in reverse, starting from the last time instance $t^* = 10$. The adjoint solver uses the same step size $\Delta t^* = 0.05$ as the primal solver. To speed up the Krylov-based adjoint equation solution, we pre-compute the preconditioners (top-level PC in Fig. 4) with a non-dimensional time interval of 2.5. We ask the adjoint equation residuals to drop five orders of magnitude for each time step.

After trials and errors, we choose four local flow features as the neural network's inputs, as shown in Table II. Although these local features are already normalized, non-dimensional quantities, we further scale them to make their standard deviation (among all mesh cells and time steps) close to one. As mentioned above, the scaling facilitates the gradient-based optimization in FIML. To better capture the optimal relationship between the local flow features and the augmentation

TABLE II: Four local flow features used in this study.

Feature	Formulation	Description	Scaling
η_1	P/D	Ratio of the turbulence production and destruction term	0.001
η_2	$\|\Omega\|/\|S\|$	Ratio of the vorticity and strain magnitudes	1.0
η_3	$\tilde{\nu}/\nu$	Ratio of the turbulence and kinematic viscosity	0.01
η_4	$\sqrt{\frac{\partial p}{\partial x_i}\frac{\partial p}{\partial x_i}}/\frac{\partial U_k^2}{\partial x_k}$	Ratio of pressure normal stress to shear stress	1.0

TABLE III: Optimization formulation for the unsteady FIML problem. We use the flow data from $t^* = 0$ to 10 for the FIML training.

	Function/Variable	Description	Quantity
Min	F	CFD prediction error along with regulation	1
w.r.t.	w and b	Weights biases in the neural network	540

scalar field β, we use two hidden layers in the built-in neural network model, and each hidden layer has 20 neurons. As mentioned previously, we use the weights and biases in the neural network model as the design variables. In total, we have 540 design variables. Note that the total number of design variables depends on the neural network architecture and is independent of the number of mesh cells and time steps. Table III summarizes the optimization formulation of the unsteady FIML problem.

The unsteady FIML optimization runs for 141 iterations. The baseline and optimized objective functions are 1.14E0 and 2.09E-2, respectively; the objective function reduces by 98.2%. The optimality drops more than two orders of magnitude; reducing from 3.2E0 to 2.6E-2. This indicates that the optimization converges tightly. The optimization runs in parallel with 16 CPU cores, and it takes about 96 hours with 2.3G Hz Intel Skylake Xeon processors on our local high-performance computing (HPC) system Nova.

Figure 7 shows the time evolution of root-mean-square error (RMSE) for C_p on the bottom wall. Here the RMSE is computed as the difference between the CFD prediction and reference data in a prescribed simulation domain (e.g., the bottom wall or the entire flow field). For example, the RMSE of C_p on the bottom wall is computed as.

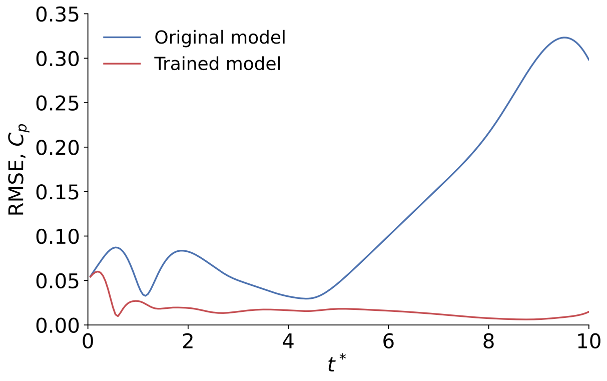

FIG. 7: Time evolution of root-mean-square error for C_p on the bottom wall. The unsteady-FIML-trained model significantly reduces the RMSE compared with the original model at all times.

$$RMSE_{C_p} = \sqrt{\sum_{i=1:N_f} \frac{({C_p}_i^{\text{CFD}} - {C_p}_i^{\text{ref}})^2}{N_f}} \tag{30}$$

where the superscript i denotes the face index, and N_f is the total number of mesh faces on the bottom wall. The original SA model has a relatively low RMSE in $t^* < 5$. Then, the RMSE rapidly increases. This trend is consistent with what we observe in Fig. 6, i.e., the original model's velocity prediction degrades rapidly for $t^* > 5$. The unsteady FIML-trained model significantly reduces the C_p RMSE at all time instances, compared with the original model. This is expected because we use the time-averaged C_p RMSE as the objective function. Note that the initial error at $t^* = 0$ is not zero for the original and trained models. This is because they both use the spun-up flow fields predicted by the SA model as the initial conditions. Their initial flow fields are slightly different from the ones used by the SST model (reference). This setup causes a small initial error for the C_p prediction in Fig. 7, as well as for all other time evolution plots in this paper.

To better illustrate the temporal evolution of pressure prediction error, we plot the pressure distribution on the bottom wall at a few time instances in Fig. 8. A low-pressure region is observed at $x^* \approx 1.5$ and $t^* = 5$, which corresponds to the ramp vortex ($x^* \approx 1.5$) observed in Fig. 6 top left. Both the original and trained models predict this low-pressure region well. However, the two models predict significantly different pressure profiles at $t^* = 8$ and 10. The unsteady-FIML-trained model accurately captures the propagation of the low-pressure region further downstream, while the original model predicts a relatively flat pressure distribution on the bottom wall. The above flat pressure distribution is mainly caused by the original model's overestimation of the ramp vortex dissipation, as mentioned previously and shown in Fig. 6 mid-right and bot-right.

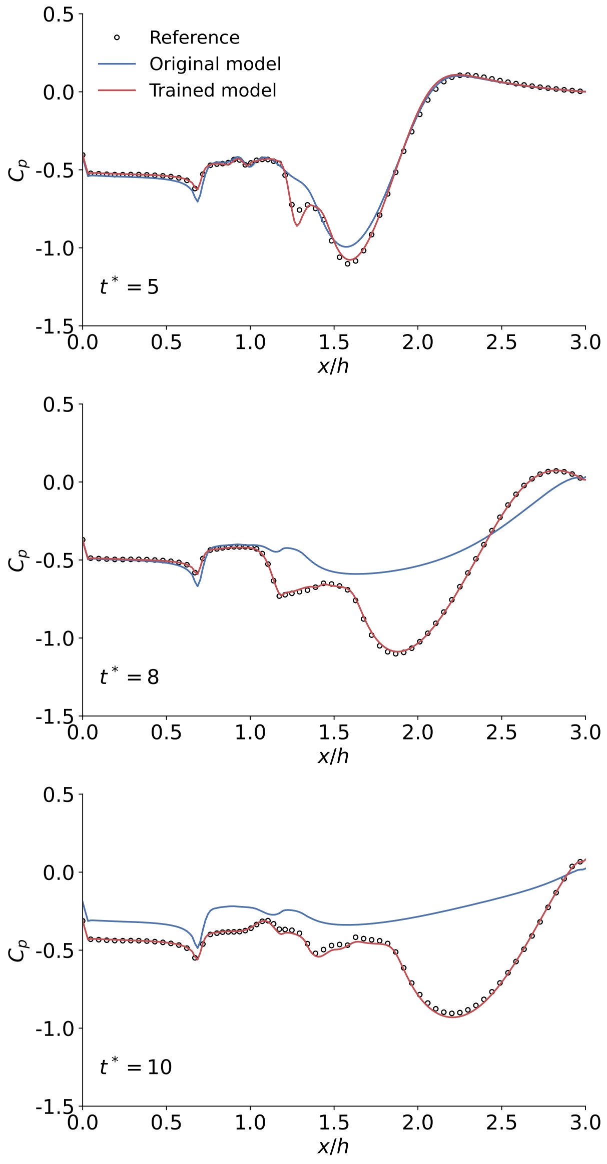

FIG. 8: Pressure distribution on the bottom wall. The unsteady-FIML-trained model has a much better agreement with the reference data than the original model at various time instances.

Having shown that the unsteady FIML successfully minimizes the spatial-temporal prediction error for the surface pressure (training data), we evaluate whether the trained model can accurately predict variables that are not used in training. Fig. 9 shows the time evolution of RMSE for the velocity field. Here we use a formulation similar to Eq. (30) to compute the velocity field RMSE, except that the domain is the whole velocity field instead of the bottom wall surface. Similar to the pressure RMSE, the original model's velocity field RMSE rapidly increases in $t^* > 5$. Again, this rapid increase in velocity RMSE is attributed to the overestimation of vortex dissipation by

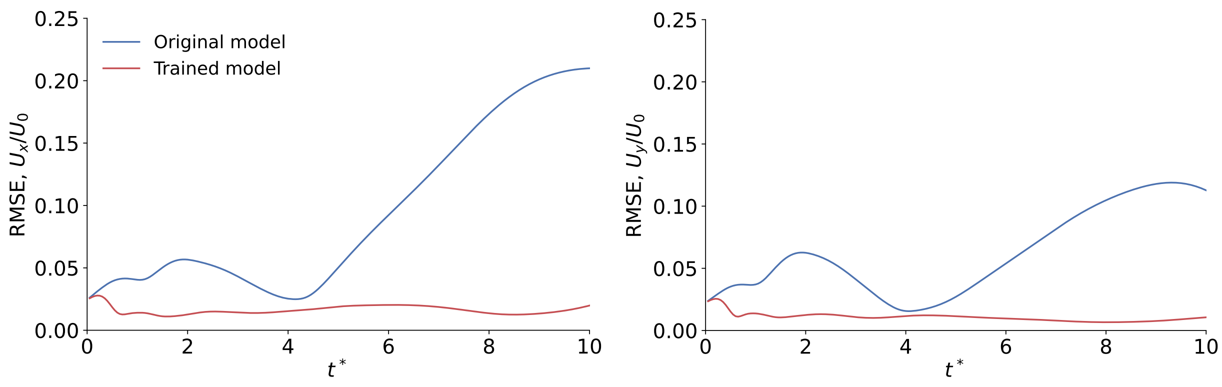

FIG. 9: Time evolution of root-mean-square error for the velocity field (Left: U_x^*. Right: U_y^*). The unsteady-FIML-trained model significantly reduces the RMSE compared with the original model.

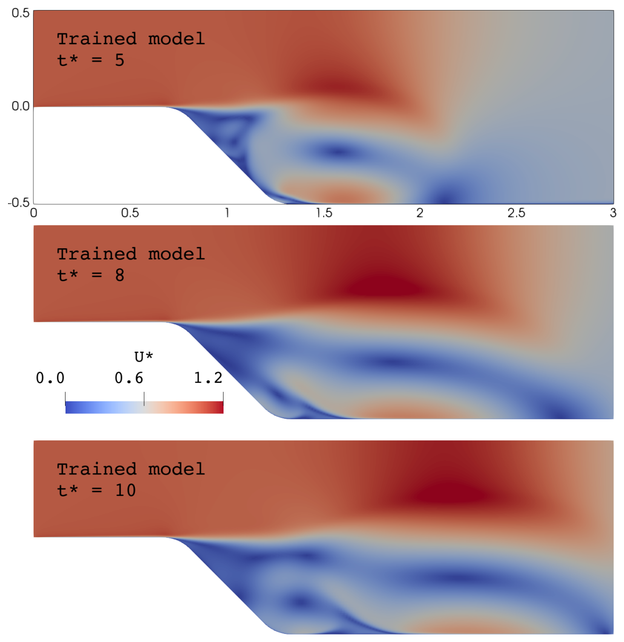

FIG. 10: Velocity magnitude contours of the unsteady-FIML-trained model at various times. The trained model predicts similar spatial-temporal variations of velocity fields as the reference model (comparing this figure with Fig. 6 left).

the original SA model, as shown in Fig. 6. The unsteady-FIML-trained model's RMSE always remains at a relatively low level.

To further evaluate the velocity field prediction accuracy, we plot the velocity magnitude contours of the unsteady-FIML-trained model in Fig. 10. The trained model predicts visually non-distinguishable spatial-temporal variations of velocity fields as the reference model (comparing this figure with Fig. 6 left). This indicates that the trained model correctly captures the ramp vortex shapes and their interactions. To further quantify this good agreement, we plot the velocity profiles at various streamwise locations in Fig. 11. Here, we overlap the ramp geometry with the streamwise (U_x^*; left column) and vertical (U_y^*; right column) velocity profiles. Here $U^* = U/U_0$ is the normalized velocity. To better illustrate the velocity profiles, we multiply U_x^* and U_y^* by a factor of 0.02, respectively, and then plot them with the ramp geometry. We use the same scaling for the rest of the velocity profile plots in this paper. The original SA model has large velocity prediction errors in $t^* = 8$ and 10, while the unsteady-FIML-trained model agrees reasonably well with the reference data at all time instances. Note that the trained model also accurately captures the shape of the boundary layer.

The above results indicate that using the surface pressure as the training data can improve the velocity prediction for the entire flow fields. This salient feature is made possible by the solver-embedded nature of the FIML method. Because the entire CFD solution process is embedded in the training process, the corrected pressure at the bottom wall can lead to the corrected velocity for the entire flow field. Note that the capability of using only surface data to improve flow field prediction accuracy has been shown in previous steady-FIML studies (to name a few[33,39,43]). This low data dependency is highly desirable in practice because many experiments can measure only surface data. Therefore, the ability to use only limited surface measurements to correct imperfect CFD models will significantly broaden FIML's applications. However, this conclusion has not been well generalized for any flow configurations. For example, in a previous study[43], we found that using only surface friction as training data could improve the velocity field prediction for the steady-state flow over a periodic hill. However, the velocity field prediction accuracy could be further improved if field data were used (e.g., velocity profile data). We recommended using both surface friction and velocity profile data. Whether using only surface data can improve flow field prediction accuracy for more general unsteady flow needs to be further studied in our future work.

From a numerical optimization standpoint, converging unsteady FIML is more challenging than steady FIML. This is because the unsteady FIML requires the CFD prediction to match the reference data at every time step, instead of only the final steady-state solution. Therefore, unsteady FIML typically needs more optimization iterations to converge. However, from a machine learning standpoint, unsteady FIML is desirable because it has hundreds of more flow field snapshots as training data than steady FIML. Thus, the turbulence model trained by unsteady FIML is less likely to overfit than the one trained by steady FIML. In addition, the unsteady FIML's train-

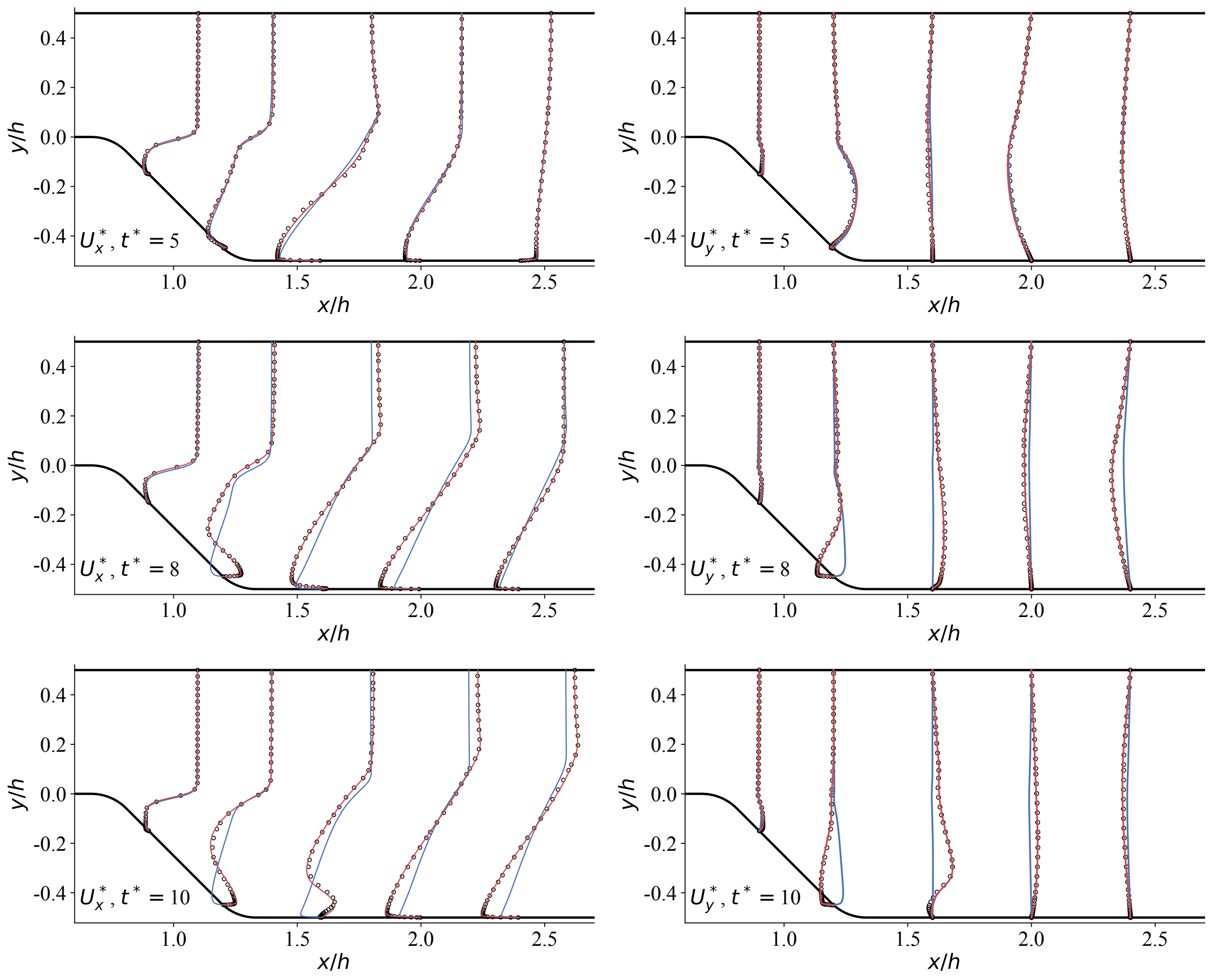

FIG. 11: Velocity profiles at various streamwise locations (left: U_x; right: U_y). ○ Reference data. – Original model. – Trained model. The unsteady-FIML-trained model agrees much better with the reference data than the original model at various time instances.

ing data are time-dependent; therefore, they cover a wider range of flow conditions than steady FIML's data. For example, one unsteady FIML problem may include transient training data for the attached boundary layer, boundary layer separation, vortex shedding, and vortex iterations. Therefore, it is plausible to expect the unsteady-FIML-trained model to have better generalizability than the steady-FIML-trained model. Such a topic will be discussed in the next subsection.

C. Generalizability of the unsteady-FIML-trained model for unseen conditions

In the above subsection, we evaluated the performance of the trained model in predicting flow fields with the same geometry and Reynolds number. To further evaluate its generalizability,

TABLE IV: Summary of training and prediction configurations.

	Training	Prediction 1	Prediction 2
Geometry	45-deg ramp	60-deg ramp	30-deg ramp
Reynolds number	10^5	10^5	5×10^4
Variables	P_{surface}	P_{surface} & U_{field}	P_{surface} & U_{field}

this subsection will use the unsteady-FIML-trained model to predict geometries and Reynolds numbers not used in training. Table IV summarizes the configurations for training and prediction. In the first prediction case, we change only the geometry (ramp angle changes from 45 to 60 degrees) and maintain the Reynolds number. In the second prediction case, we change both the geometry (ramp angle from 45 to 30 degrees) and Reynolds number (from 10^5 to 5×10^4). Note that for steady-state flow problems, we typically need training data from various geometries and flow conditions. However, this situation can be alleviated in unsteady cases because one unsteady FIML typically includes flow data for a few hundred flow field snapshots covering a wide range of flow conditions, as mentioned above. Therefore, we train our model using only one case (45-deg ramp with $Re = 10^5$).

Figure 12 shows the velocity magnitude contour predictions for an unseen 60-degree ramp geometry with $Re = 10^5$ (prediction case 1). Compared with the 45-deg ramp case, the 60-deg ramp exhibits larger flow separation and a stronger ramp main vortex at $t^* = 8$ and $x^* \approx 2$ (Fig. 12 top left). In addition, the ramp vortex rolls up a bigger secondary vortex near the bottom of the ramp ($x^* \approx 1.2$). At $t^* = 10$, the main vortex propagates further downstream ($x^* \approx 2.2$), and the center of the secondary vortex rises slightly (i.e., y^* from -0.4 to -0.1), resulting in an intriguing double-vortex structure (Fig. 12 top right). The unsteady-FIML-trained model qualitatively captures the above vortex structures and interactions (Fig. 12 bot), although it inaccurately predicts some detailed vortex structures, such as the location and shape of the secondary vortex at $t^* = 10$. On the other hand, the original model fails to predict any of the vortex structures at $t^* = 8$ and 10 (Fig. 12 mid); it predicts a relatively smooth wake instead. Overall, the trained model significantly outperforms the original model in predicting the velocity contours.

A more quantitative comparison of flow field predictions is shown in Fig. 13, where we plot the velocity profiles at various locations. The unsteady-FIML-trained model significantly improves the velocity field prediction at all three time instances, except for U_y^* in the $1.0 < x^* < 1.5$ region

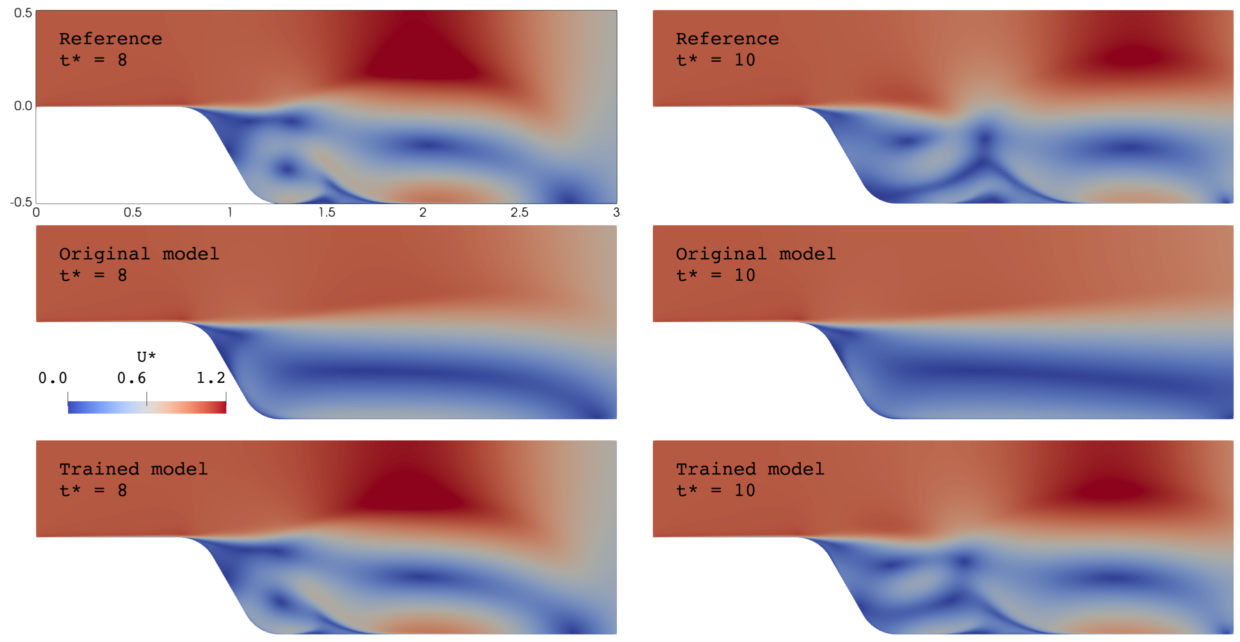

FIG. 12: Velocity magnitude contour predictions for an unseen 60-degree ramp geometry with $Re = 10^5$. The unsteady-FIML-trained model qualitatively captures the complex vortex structures and interactions, while the original model's prediction is poor.

at $t^* = 10$ (Fig. 13 bot-right). This relatively large error is related to the inaccurate prediction of the shape and location of the secondary vortex observed in the velocity contour (comparing Fig. 12 top right with bot right). Note that this level of error in the velocity field prediction is similar to previous steady-state FIML studies[33,39,43] that used a similar setup (i.e., using surface data for training and predicting field variables).

Next, we consider the velocity field prediction for an unseen geometry (30-deg ramp) and an unseen Reynolds number ($Re = 5 \times 10^4$); prediction case 2, as shown in Fig. 14. The reference SST model predicts a ramp main vortex at $t^* = 8$ and 10, similar to the previous cases. However, as the flow evolves, the main vortex does not roll up a strong secondary vortex, which is different from the 45-deg and 60-deg ramp cases shown before. This is mainly attributed to this case's weaker flow separation and smaller ramp angle. Compared with the previous case, the original SA model's prediction matches the reference data better. For example, the original model captures the structure of the ramp main vortex well at $t^* = 8$. Overall, the unsteady-FIML-trained model only slightly outperforms the original SA model, e.g., at $t^* = 10$. We can see this clearly in Fig. 15, where we plot the velocity profiles at various streamwise locations. For example, the trained model effectively reduces the prediction error at $t^* = 5$ and 8; however, its prediction becomes

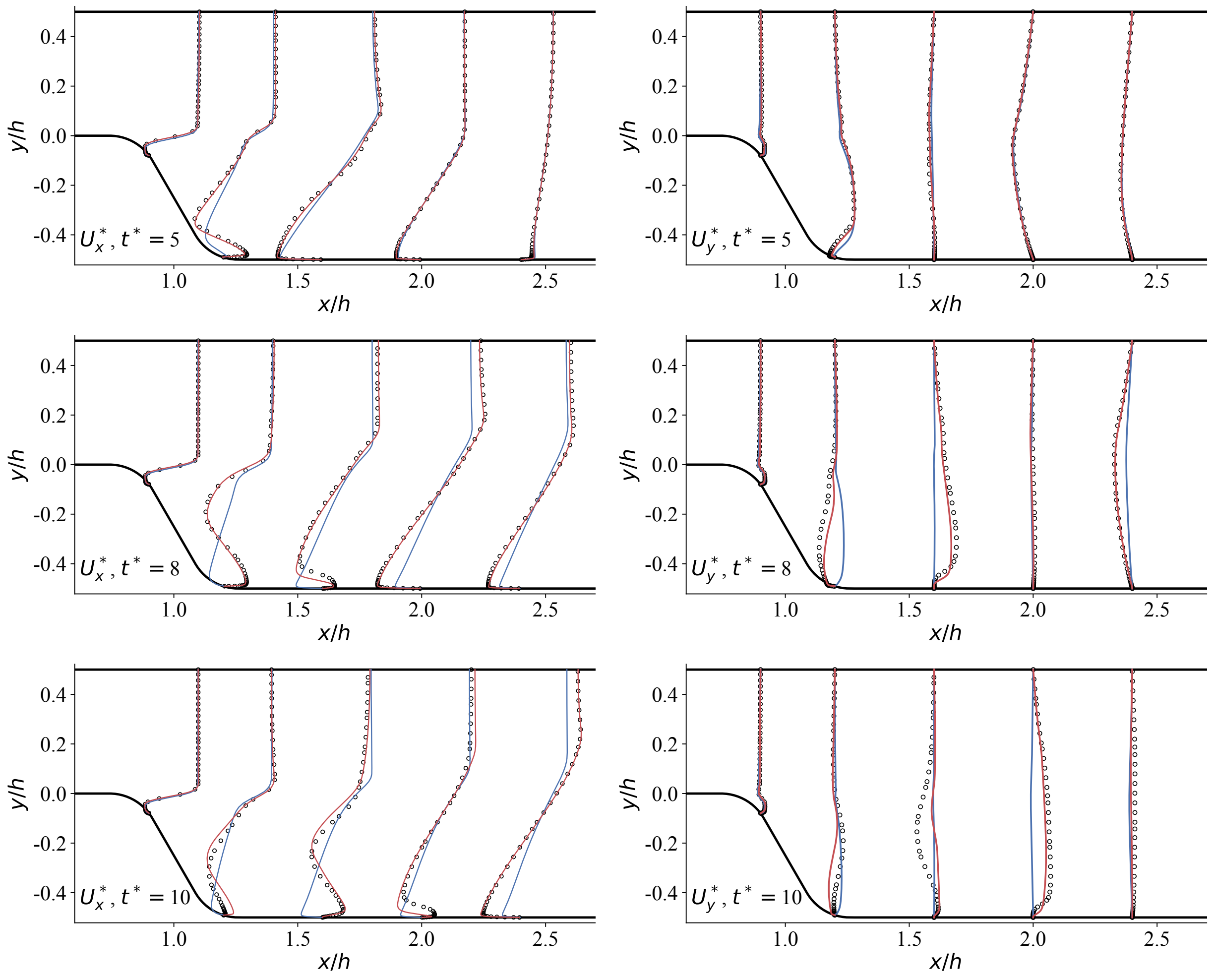

FIG. 13: Velocity profile prediction for an unseen 60-degree ramp geometry with $Re = 10^5$. Left: U_x^*; right: U_y^*. ○ Reference data. – Original model. – Trained model. The unsteady-FIML-trained model agrees much better with the reference data than the original model at various time instances.

less accurate at $t^* = 10$ and has similar errors as the original SA model. Overall, the difference between the original and trained SA models is much less than the 60-deg ramp case. This is expected because a lower ramp angle and a lower Reynolds number make the ramp separation prediction less challenging. The above results also indicate that our trained model does not overfit, and it performs well in cases where the original and reference models have similar predictions.

Figure 16 shows the time evolution of velocity field RMSE for the two prediction cases. Overall, the unsteady-FIML-trained model outperforms the original model, especially when intriguing transient vortex structures are formed ($t^* > 5$). For the 60-deg case, the original model has much larger errors than the 30-deg case, which is consistent with what we observed from the velocity

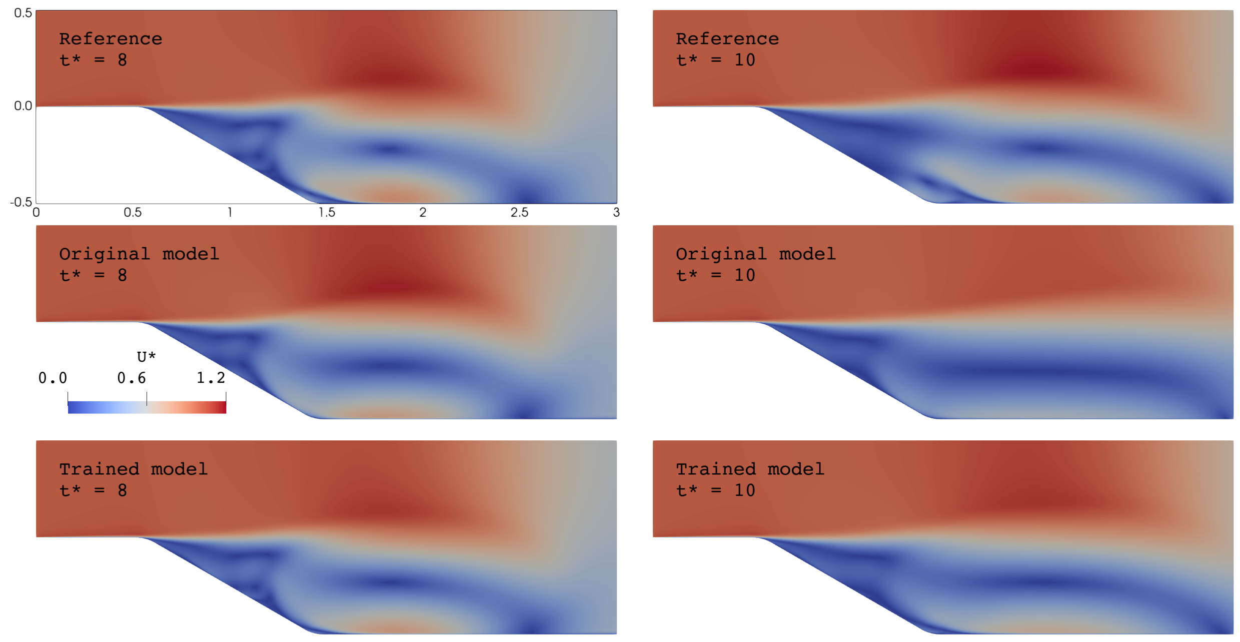

FIG. 14: Velocity magnitude contour predictions for an unseen 30-degree ramp geometry and an unseen Reynolds number $Re = 5 \times 10^4$. The trained model slightly outperforms the original model.

contour and profile plots. The trained model significantly lowers the error for all time instances. For the 30-deg case, the trained model is slightly worse than the original model in $t^* < 3$; however, both the original and trained models' error remains at a relatively low level, so this behavior is inconsequential. Another important trend we observe is that the trained model's error grows with time for predictive cases. This is expected because the prediction error generally accumulates with time for unsteady flow. For example, if the model predicts a wrong vortex location at $t^* = 5$, the prediction accuracy will be degraded for the rest of the simulation, even if the model then perfectly predicts how the vortex will evolve spatially and temporally. In this sense, the RMSE time series is a strict metric because even the reference and trained model predict the exact same vortex structure in space but with a time shift, this error will be reflected in the RMSE. In the future, we will evaluate more metrics to quantify the overall unsteady FIML performance, such as the correlation.

Having compared the velocity field predictions, we evaluate the pressure prediction capability for unseen geometries and Reynolds numbers, as shown in Fig. 17. Again, the trained model outperforms the original model in both cases. However, the trained model's surface pressure prediction has a relatively large error for the 60-deg ramp case at $t^* = 10$ (Fig. 17 bottom left). Although the trained model predicts the low-pressure region associated with the ramp main vortex

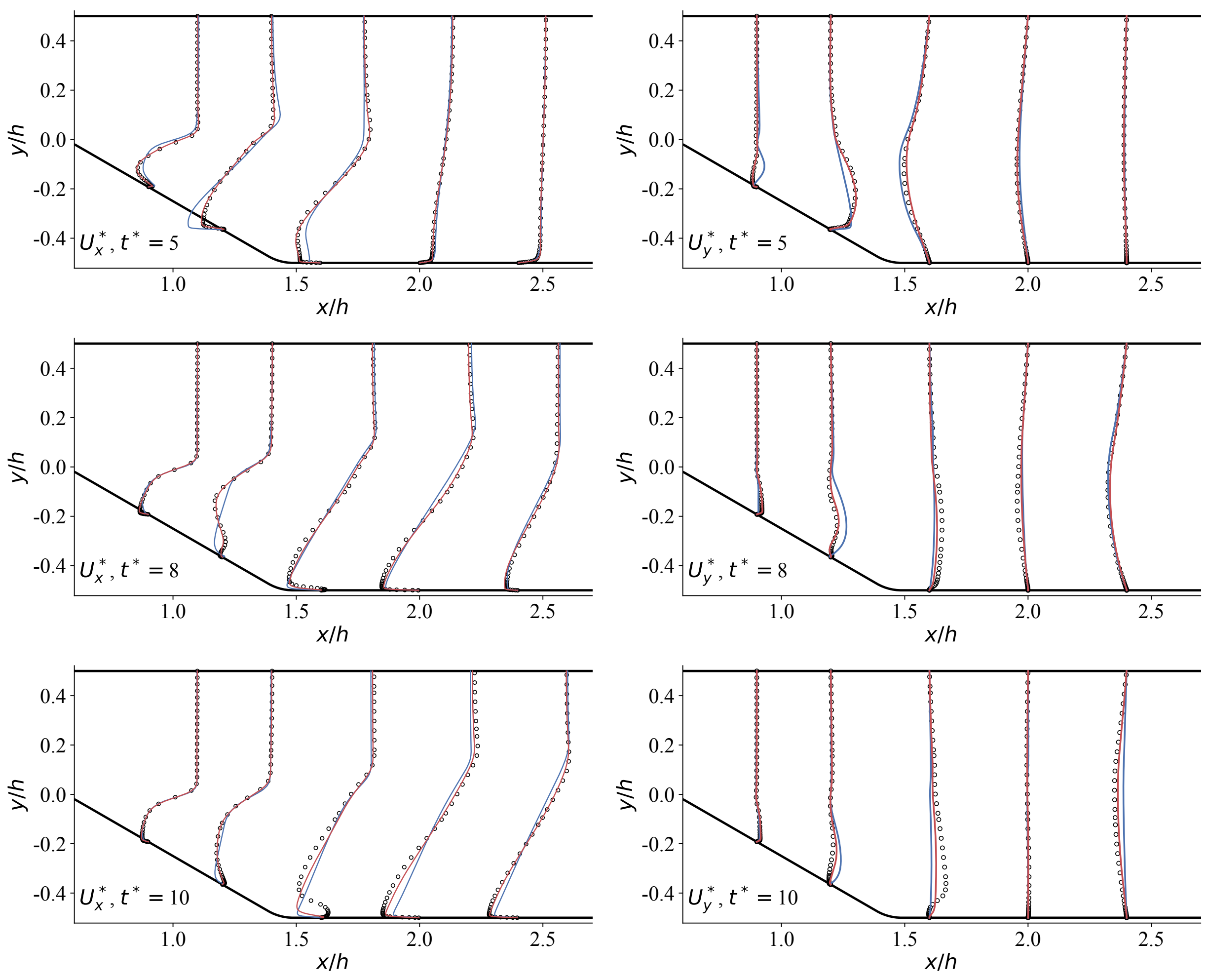

FIG. 15: Velocity profile prediction for an unseen 30-degree ramp geometry and an unseen Reynolds number $Re = 5 \times 10^4$. Left: U_x^*; right: U_y^*. ○ Reference data. – Original model. – Trained model. The trained model slightly outperforms the original model.

at $x^* = 2.5$, the overall pressure level is much lower than the reference data in $x^* < 2$. Again, this relatively large pressure error is associated with the less accurate prediction of the complex velocity field (vortex interaction) at $t^* = 10$, as observed in Fig. 12.

Figure 18 shows the time evolution of C_p RMSE for the two prediction cases. The overall trend of the pressure RMSE is similar to the velocity RMSE. However, the pressure prediction error increases more rapidly, especially for the 60-deg ramp case. This indicates that the model not only predicts the incorrect spatial distribution of pressure but also mispredicts how the pressure profile evolves in time at $t^* \approx 10$. This behavior is consistent with what we observed in the pressure profile distribution in Fig. 17; the pressure is much more challenging to predict for the large flow

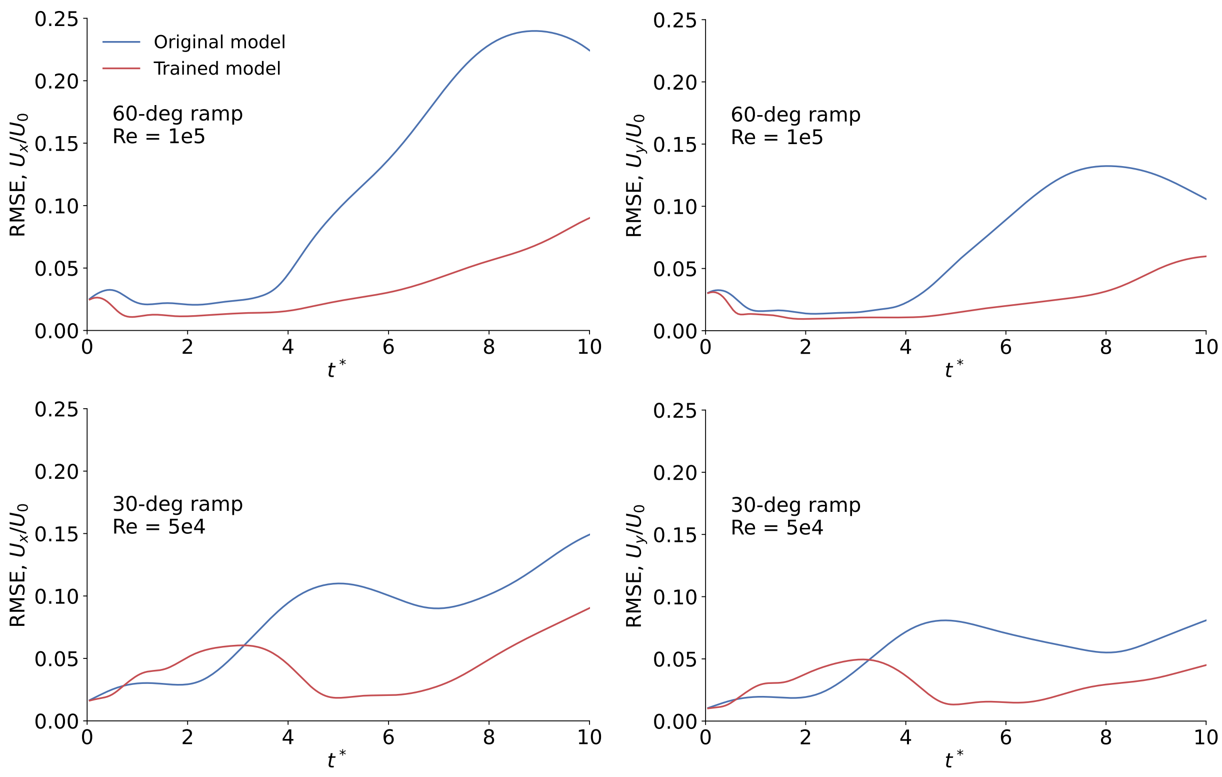

FIG. 16: Time evolution of velocity field root-mean-square error for unseen geometries (60-deg and 30-deg ramps) and an unseen Reynolds number ($Re = 5 \times 10^4$). Overall, the trained model outperforms the original model.

separation and complex vortex iteration at $t^* = 10$.

In summary, this section showcases the generalizability of the trained model for predicting unseen geometries and flow conditions. Overall, the trained model outperforms the original model. To further improve the trained model's performance, we can do the following. (1) Train the model using more geometries and flow conditions. Although one unsteady FIML contains many more flow field snapshots for training than the steady FIML, it will still be beneficial to include training data from more conditions, which essentially covers a wider range of possible flow features. (2) Train the model for more physical time. To save computational time, this paper trains the model using data from t^* from 0 to 10 and predicts unseen conditions for the same physical time. One potential strategy is to use a longer time in training than in predictions. For example, we can train our model using data from t^* from 0 to 12, and then use the trained model to predict t^* =0 to 10 for unseen conditions. This strategy will include more information about how the flow will evolve and can potentially improve the model's prediction capability. (3) Use more flow field variables

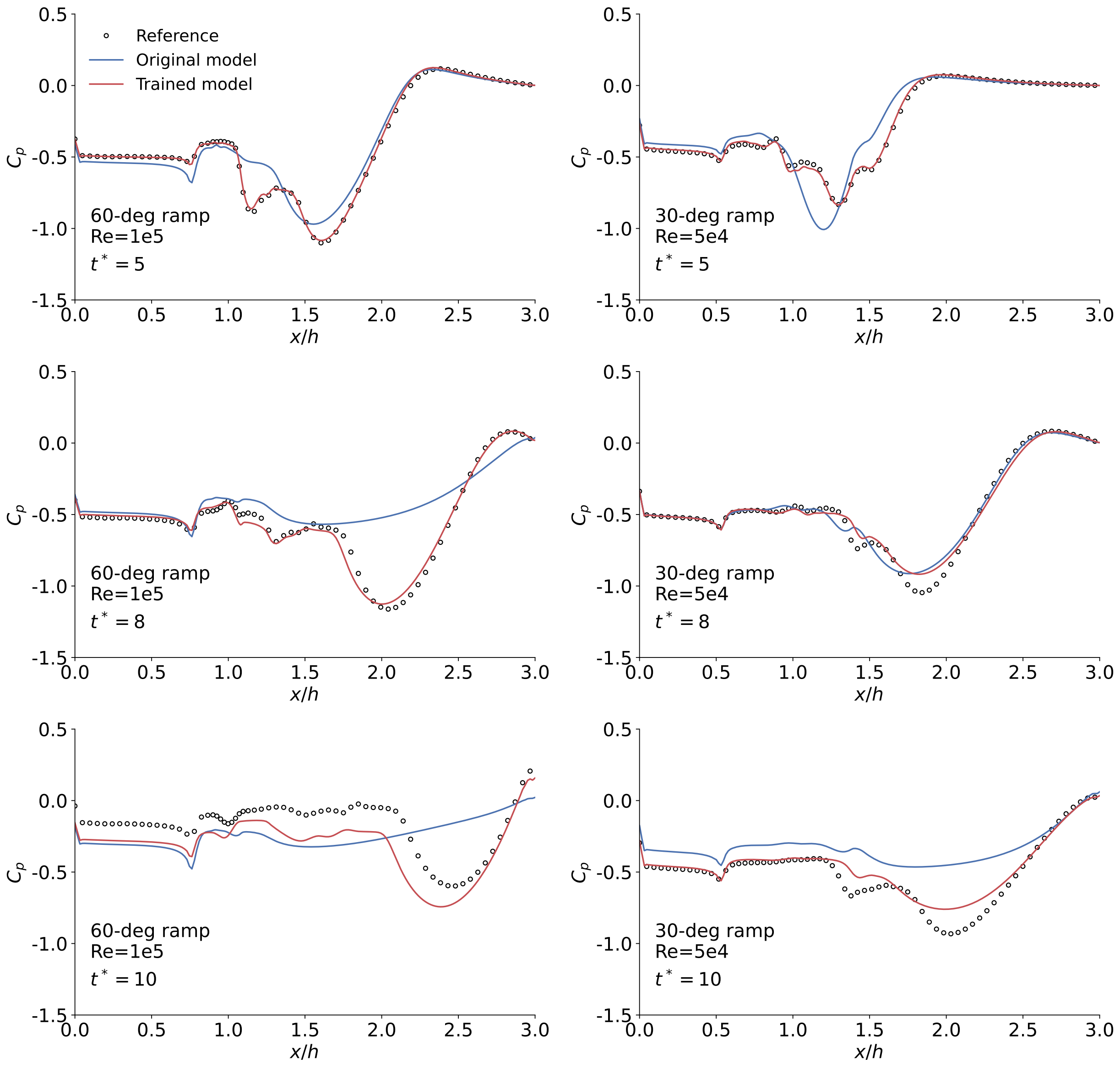

FIG. 17: Pressure profile prediction at the bottom wall for unseen geometries (60-deg and 30-deg ramps) and an unseen Reynolds number ($Re = 5 \times 10^4$). The trained model significantly reduces the error for $t^* = 5$ and 8 and has a relatively large error at $t^* = 10$.

as training data. This paper uses only the surface pressure as training data to predict velocity fields with unseen conditions. A more comprehensive analysis can be conducted to evaluate its effectiveness for predicting other variables, such as the Reynolds stress fields. Also, it is not clear to what extent using more variables (such as the velocity field) as training data will improve the trained model's prediction accuracy. We would like to highlight that the main objective of this paper is to introduce our open-source unsteady FIML framework and demonstrate its basic

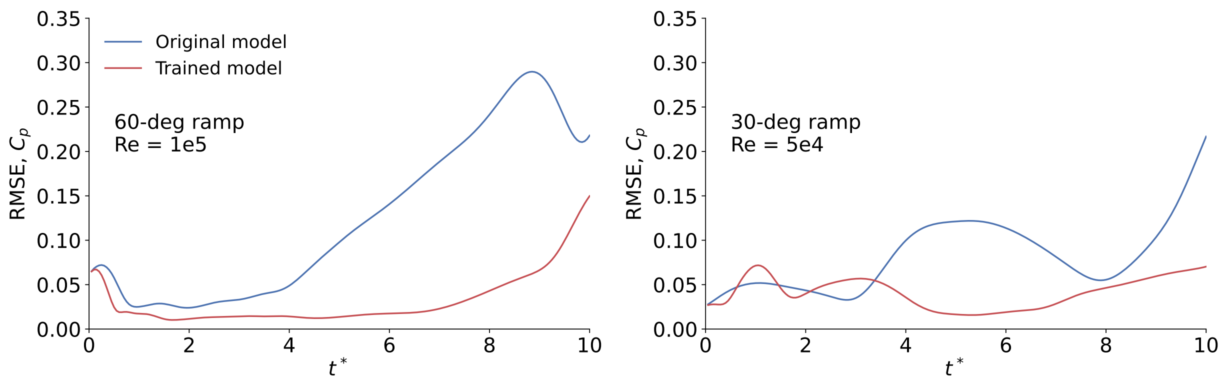

FIG. 18: Time evolution of C_p root-mean-square error for the two prediction cases. Overall, the trained model outperforms the original model.

prediction capability. A comprehensive study on how to optimize the unsteady FIML performance is outside the scope of this paper and will be conducted in our future work. In addition, this study assumes the trained model will be used to predict flow configurations that are similar to the training dataset. Generalizing the trained model for many other flow configurations, such as the flat plate flow, is outside the scope of this paper. Relevant studies on this topic can be found in[45,69,70].

D. Feasibility of using a steady-state trained model to predict time-accurate unsteady flow

As mentioned before, Fidkowski[47] demonstrated that a steady-state trained model (time-averaged flow data) could improve prediction accuracy for time-averaged periodic unsteady flow. In this subsection, we evaluate whether the Fidkowski[47]'s approach works for more general, time-accurate unsteady flow problems. To this end, we use the 45-deg ramp as the benchmark and run steady-state flow simulations and FIML using OpenFOAM's built-in `simpleFoam` solver. The `simpleFoam` solver has a similar code structure as `pimpleFoam`. It also solves Eqs. (3) to (11) iteratively, except that it does not have the time-derivative terms in the momentum and turbulence equations. Therefore, using `simpleFoam`-based FIML to optimize the weights and biases and then substitute them into `pimpleFoam` for prediction mimics the Fidkowski[47]'s approach.

Figure 19 shows the steady-state velocity magnitude contours for the reference SST model and the original SA model. Both models predict similar ramp wake at the steady state, although they predict significantly different temporal evolution of ramp vortex structures, as shown in Fig. 6. Therefore, we expect that using the steady-state flow training data will not be able to train an accurate model for predicting the spatial-temporal variations of unsteady flow for this case.

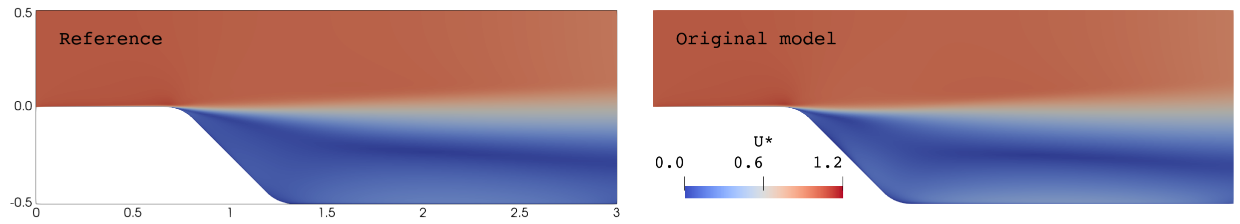

FIG. 19: Velocity magnitude contours from steady-state flow simulations. Left: reference data generated by the $k-\omega$ SST model. Right: data generated by the original SA model. The reference and original models predict a similar flow field at steady-state.

To confirm the above point, we conduct a steady-state FIML to train the SA model and make its prediction match the flow field predicted by the SST model. The objective function is similar to the one used for unsteady FIML, except that we consider only the final converged flow field, instead of flow fields for all time steps.

$$F_{\text{steady}} = \frac{c_1}{N_i} \sum_{i=1:N_i} (f_i^{\text{CFD}} - f_i^{\text{ref}})^2 + \frac{c_2}{N_j} \sum_{j=1:N_j} (\beta_j - 1)^2 \tag{31}$$

Other optimization configurations, including the design variables, neural network architecture, and flow features are the same as the unsteady FIML, except that: (1) we use a different scaling (0.0001) for the first flow feature (production over destruction). (2) we use different weights to balance the two terms in the objective function, i.e., $c_1 = 1$ and $c_2 = 0.001$. (3) we use the steady-state flow solver `simpleFoam` to simulate the flow. The optimization runs for 54 iterations, and the objective function and optimality drop by 96% and two orders of magnitude, respectively. We use 16 cores on the Nova HPC system, and the optimization takes about 2 hours.

Figure 20 compares the bottom-wall pressure profiles among the reference, original, and steady-FIML trained models. Note that we use a different y axis scaling than the unsteady cases because the pressure difference between the original and reference models is small. The trained model does improve the pressure prediction accuracy at the steady state. Then, we use the optimized weights and biases as the inputs and run unsteady simulations using `pimpleFoam` with a built-in neural network model. The neural network architecture (e.g., the flow features, input scaling, and number of hidden layers and neurons) used in the steady-FIML training is the same as the one used for prediction. The comparison of velocity profiles among the three models is shown in Fig. 21. The steady-trained model does not improve the unsteady flow simulation accuracy. This is probably because the original and reference models predict similar steady-state

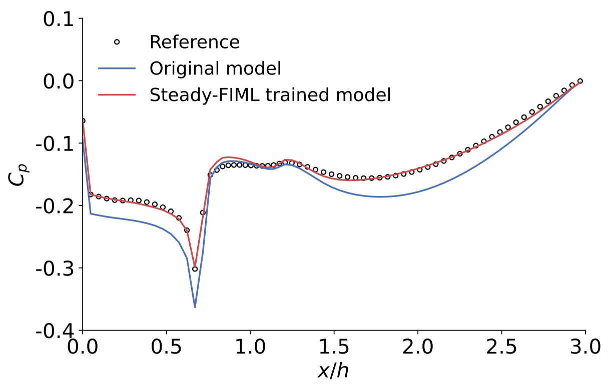

FIG. 20: Comparison of steady-state pressure profiles at the bottom wall among reference data, original SA model, and the steady-FIML-trained model. The trained model reduces the pressure prediction error at the steady state.

flow solutions, so the steady-FIML does not contain useful training data for correcting unsteady flow simulations.

In summary, we test the Fidkowski[47] method and find that using a steady-state trained model cannot guarantee to improve general unsteady flow where both spatial and temporal variations of flow are of interest. This conclusion further justifies the need for an unsteady FIML.

IV. CONCLUSION

In this paper, we develop an open-source field inversion machine learning (FIML) framework to augment RANS turbulence modeling for predicting time-accurate unsteady flow, where both spatial and temporal variations of flow are of interest. This framework allows us to augment a RANS turbulence model with a scalar field. Then, it uses a built-in neural network model to compute the augmentation scalar field using selected local flow features as inputs. Finally, it solves an inverse problem by optimizing the weights and biases of the neural network model to minimize the augmented turbulence model's regulated prediction error. The prediction error is quantified as the spatial-temporal flow field difference between the CFD simulations and the reference data. To avoid overfitting, this framework can also add a regulation term to the objective function. The FIML optimization leverages an efficient adjoint method called PIMPLE-Krylov to compute time-accurate unsteady gradient information. The PIMPLE-Krylov adjoint method's main advantage is that it uses the PIMPLE method to solve the unsteady flow, allowing the usage of a relatively large CFL number (e.g., 10) to minimize the number of time steps. Then, it uses a Krylov method

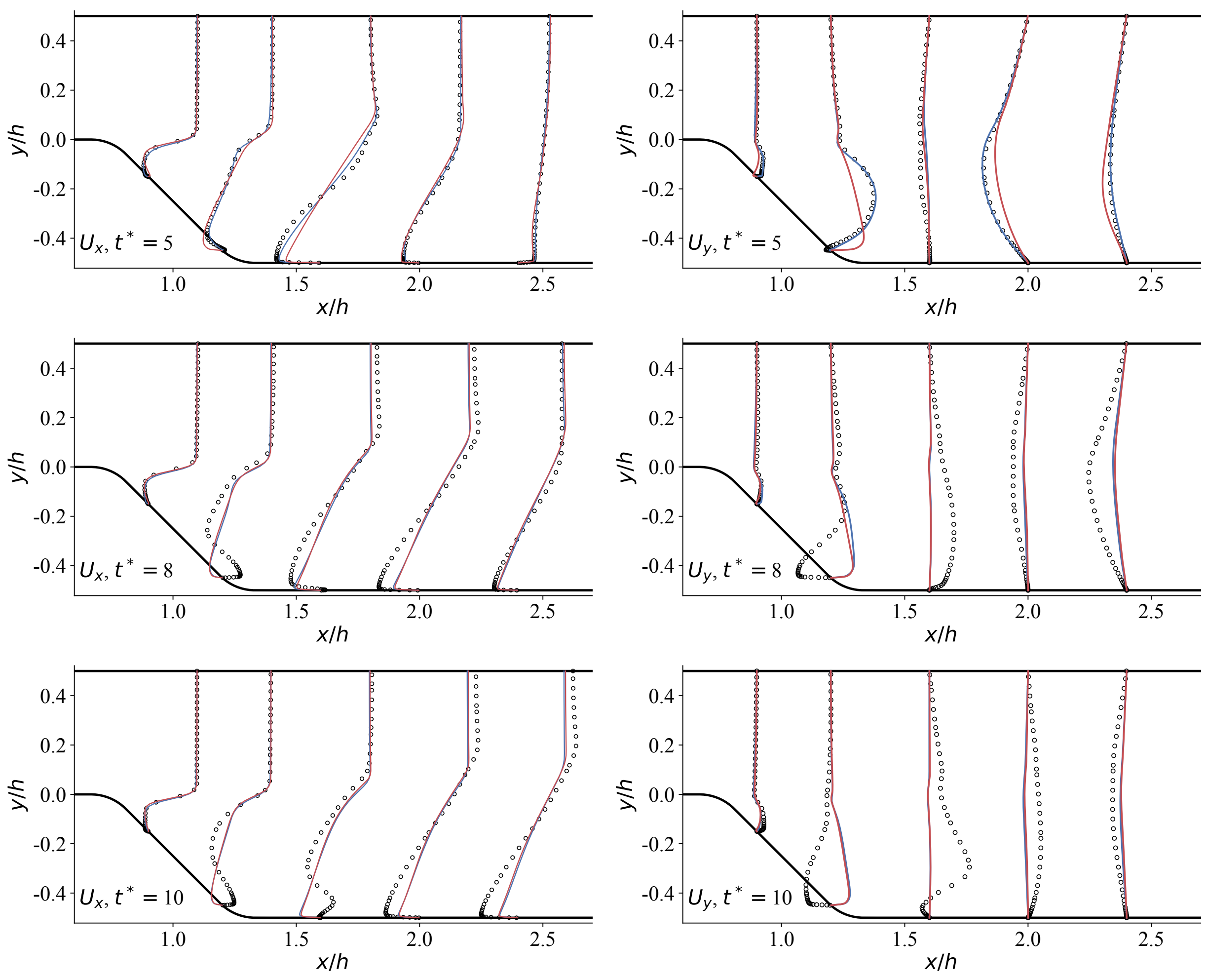

FIG. 21: Velocity profile prediction for unsteady flow evolution from steady-state FIML. Left: U_x^*; right: U_y^*. ○ Reference data. – Original model. – Steady-FIML trained model. The steady-FIML-trained model does not improve unsteady flow prediction accuracy.

to solve the adjoint equation for each time step. Because the number of time steps is reduced by using a relatively large CFL number, the PIMPLE-Krylov method needs to solve a relatively small number of adjoint equations and read/write a relatively small amount of data to the disk, which speeds up its computation.

To demonstrate the proposed unsteady FIML framework, we consider the spatial-temporal variations of unsteady flow over a 45-degree ramp with a Reynolds number of 10^5. Using the surface pressure as the training data, we augment the SA turbulence model's production term and make its prediction match the reference computed from the $k-\omega$ SST model. We conduct two FIML optimizations: steady and unsteady. For the steady FIML, the objective function is the regulated surface pressure prediction error for the converged, steady flow field, while for the unsteady FIML, the objective function is the regulated average prediction error for all time steps. We find that the model trained by the steady-state FIML cannot accurately predict the unsteady flow. On the other hand, the unsteady-FIML-trained model significantly reduces the prediction error for spatial-temporal variations of velocity fields and the dynamic vortex structures over the ramp. The above result justifies the need for an unsteady FIML. In addition, we find that the proposed unsteady FIML can use only the surface pressure data to train a model that accurately predicts the velocity fields.

Finally, we evaluate the generalizability of the unsteady-FIML model, which is trained using data only from the 45-deg ramp case with a Reynolds number of 10^5. We consider two predictive cases. The first predictive case considers an unseen geometry (60-deg ramp), and the second case considers an unseen geometry (30-deg ramp) along with an unseen Reynolds number (5×10^4). The unsteady-FIML-trained model significantly reduces the velocity field and surface pressure prediction errors for both predictive cases, compared with the original SA model.

The unsteady FIML capability has been integrated into our open-source CFD-based optimization framework DAFoam[49]. The proposed framework has the potential to train accurate and generalizable turbulence models for other unsteady flow phenomena, such as wind gust response, bubbly flow, and particle dispersion in the atmosphere.

One of the main limitations of this work is that the inverse problem is challenging to solve because it includes a highly nonlinear neural network model and a computationally expensive CFD solver. For a new problem, one may need to explore many numerical configurations mentioned in Sec. II to converge the inverse problem well. In the future, we will evaluate more regression models, such as radial basis functions, random forest, and symbolic regression, and compare their performance with the neural network model used in this paper. Another limitation of this work is the temporal discretization accuracy. We use the PIMPLE solver with a relatively large step size to simulate unsteady flow. The temporal discretization error can become large if the temporal evolution of the unsteady flow is drastic, e.g., at high Reynolds numbers. The large temporal discretization error can potentially downgrade the PIMPLE simulation's accuracy. We suggest users conduct a time step size sensitivity study for a new case. Lastly, the proposed unsteady FIML method is solver-intrusive, and extending its application requires significant code development effort. The current unsteady FIML framework supports only single-phase, incompressible turbulent flow. In the future, we will extend its capability for handling compressible flow and multiphase

TABLE V: Comparison of speed and memory between the unsteady flow and PIMPLE-Krylov adjoint. The adjoint's runtime is almost three times as large as the flow's runtime. In addition, the adjoint solver uses about seven times as much memory as the flow solver. The case runs in parallel with 8 CPU cores.

	Flow	Adjoint	Adjoint/Flow
Runtime, s	767	2192	2.9
Memory, GB	1.0	6.9	6.9

flow.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Number 2223676.

Appendix A: Performance evaluation for the proposed time-accurate PIMPLE-Krylov adjoint method

The adjoint gradient computation method's performance is crucial for large-scale gradient-based optimization in FIML. This section evaluates the speed, scalability, memory usage, and accuracy of the proposed PIMPLE-Krylov adjoint method. As mentioned before, we use unsteady turbulent flow over a ramp as the benchmark, as shown in Fig. 5. The mesh, boundary conditions, flow configurations, and adjoint configurations are the same as those used in Secs. III A and III B. The computation was done using 8 CPU cores with Intel Skylake Xeon processors running at 2.3G Hz. Table V shows the comparison of speed and memory usage between the unsteady flow and adjoint computation. The adjoint solver's runtime is almost three times as large as the flow's runtime, and it uses about seven times as much memory as the flow solver. This performance is acceptable for FIML optimization.

Next, we evaluate the parallel efficiency of the proposed PIMPLE-Krylov adjoint solver. We run the flow and adjoint solvers with various number of CPU cores, ranging from 1 to 16, and the scalability is shown in Fig. 22. The flow and adjoint solvers exhibit 77% and 85% parallel efficiency with 4 CPU cores or 5000 mesh cells per core. With more than 4 cores or fewer than

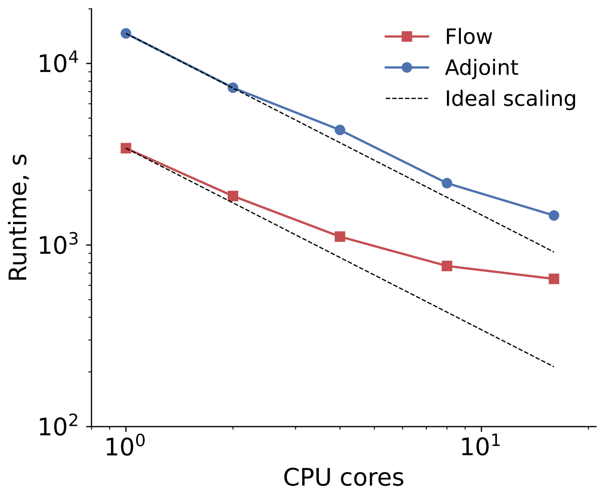

FIG. 22: Scalability of the unsteady flow and PIMPLE-Krylov adjoint solvers. The adjoint solver scales better than the flow solver.

TABLE VI: Verification of the PIMPLE-Krylov unsteady adjoint accuracy. The average error between the adjoint and reference derivatives is about 1%. The reference derivative is computed by using the forward-AD approach.

$\mathrm{d}F/\mathrm{d}x_i$	Adjoint	Reference	Error
000	1.3408e-3	1.3312e-3	0.72%
001	5.6034e-3	5.5070e-3	1.75%
002	2.3600e-2	2.3557e-2	0.18%
200	1.8240e-3	1.8121e-3	0.66%
539	5.3490e-2	5.4093e-2	1.11%
540	1.7589e-0	1.7224e-0	2.12%

5000 mesh cells per core, the parallel efficiency decreases rapidly. Overall, the adjoint solver scales better than the flow solver.

Finally, we evaluate the total derivative accuracy of the proposed PIMPLE-adjoint approach, as shown in Table VI. The accuracy of the total derivatives is important for the robustness of field inversion optimization. Inaccuracy derivatives will mislead the optimization and result in sub-optimal results. We run the unsteady field inversion to compute the total derivatives with respect to a few selected weights and biases, i.e., $\mathrm{d}F/\mathrm{d}x_i$. The objective function f is defined as the error between the CFD prediction and reference data, along with a regulation term (Eq. (29)). We use

TABLE VII: Verification of the steady-state adjoint accuracy. The adjoint derivatives match the forward-AD references with six significant digits.

$\mathrm{d}F_{\mathrm{steady}}/\mathrm{d}x_i$	Adjoint	Reference	Error
000	$1.04241\underline{64}$e-5	1.0424159e-5	<0.001%
001	$6.10020\underline{98}$e-6	6.1002087e-6	<0.001%
002	$1.56665\underline{59}$e-5	1.5666554e-5	<0.001%
200	$2.40082\underline{45}$e-4	2.4008238e-4	<0.001%
539	$6.45598\underline{47}$e-3	6.4559836e-3	<0.001%
540	$8.38653\underline{75}$e-1	8.3865361e-1	<0.001%

the forward-mode AD method to compute reference derivatives. The average error in the unsteady adjoint derivatives is less than 1%. As a reference, we also verify the derivative accuracy for the `simpleFoam` steady-state adjoint solver in Table VII. The derivatives of the steady-state objective function with respect to the neural network's weights and biases ($dF_{\mathrm{steady}}/dx_i$) match the forward mode AD reference with six significant digits. We speculate the relatively large error in the unsteady adjoint solver is primarily caused by the PIMPLE primal solver's under-relaxation strategy. Specifically, for each time step, we run multiple under-relaxed PIMPLE corrector iterations except for the last one. Suddenly turning off the under-relaxation at the last PIMPLE iteration may make the flow residuals deviate from zeros at each time step, and the accumulated error eventually downgrades the unsteady adjoint accuracy. However, this level of error is still acceptable for large-scale gradient-based optimization in this paper. We will address the above issue and improve the adjoint accuracy in future work. Overall, our PIMPLE-Krylov adjoint exhibits acceptable speed, scalability, and accuracy and is ready to conduct FIML optimization for various cases.

REFERENCES

[1]N. Qin, A. Vavalle, A. Le Moigne, M. Laban, K. Hackett, and P. Weinerfelt, "Aerodynamic considerations of blended wing body aircraft," Progress in Aerospace Sciences **40**, 321–343 (2004).

[2]T. R. Brooks, G. K. W. Kenway, and J. R. R. A. Martins, "Benchmark aerostructural models for the study of transonic aircraft wings," AIAA Journal **56**, 2840–2855 (2018).

[3]T. C. A. Stokkermans, N. van Arnhem, T. Sinnige, and L. L. M. Veldhuis, "Validation and comparison of RANS propeller modeling methods for tip-mounted applications," AIAA Journal **57**, 566–580 (2019).

[4]W. Shyy, N. Papila, R. Vaidyanathan, and K. Tucker, "Global design optimization for aerodynamics and rocket propulsion components," Progress in Aerospace Sciences **37**, 59–118 (2001).

[5]R. Votta, A. Schettino, and A. Bonfiglioli, "Hypersonic high altitude aerothermodynamics of a space re-entry vehicle," Aerospace Science and Technology **25**, 253–265 (2013).

[6]N. J. Falkiewicz, C. E. Cesnik, A. R. Crowell, and J. J. McNamara, "Reduced-order aerothermoelastic framework for hypersonic vehicle control simulation," AIAA journal **49**, 1625–1646 (2011).

[7]C. Othmer, "Adjoint methods for car aerodynamics," Journal of Mathematics in Industry **4**, 6 (2014).

[8]N. Ashton, A. West, S. Lardeau, and A. Revell, "Assessment of rans and des methods for realistic automotive models," Computers & fluids **128**, 1–15 (2016).

[9]P. He, C. A. Mader, J. R. R. A. Martins, and K. J. Maki, "An aerodynamic design optimization framework using a discrete adjoint approach with OpenFOAM," Computers & Fluids **168**, 285–303 (2018).

[10]M. O. L. Hansen, J. N. Sørensen, S. Voutsinas, N. Sørensen, and H. A. Madsen, "State of the art in wind turbine aerodynamics and aeroelasticity," Progress in aerospace sciences **42**, 285–330 (2006).

[11]B. J. Vanderwende, B. Kosović, J. K. Lundquist, and J. D. Mirocha, "Simulating effects of a wind-turbine array using les and rans," Journal of Advances in Modeling Earth Systems **8**, 1376–1390 (2016).

[12]M. H. A. Madsen, F. Zahle, N. N. Sørensen, and J. R. R. A. Martins, "Multipoint high-fidelity CFD-based aerodynamic shape optimization of a 10 MW wind turbine," Wind Energy Science **4**, 163–192 (2019).

[13]P. R. Spalart and V. Venkatakrishnan, "On the role and challenges of cfd in the aerospace industry," The Aeronautical Journal **120**, 209–232 (2016).

[14]K. Duraisamy, P. R. Spalart, and C. L. Rumsey, "Status, emerging ideas and future directions of turbulence modeling research in aeronautics," Tech. Rep. (2017).

[15]J.-X. Wang, J.-L. Wu, and H. Xiao, "Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data," Physical Review Fluids **2**,

034603 (2017).

[16]J.-L. Wu, H. Xiao, and E. Paterson, "Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework," Physical Review Fluids **3**, 074602 (2018).

[17]J. Weatheritt and R. Sandberg, "A novel evolutionary algorithm applied to algebraic modifications of the rans stress–strain relationship," Journal of Computational Physics **325**, 22–37 (2016).

[18]Y. Zhao, H. D. Akolekar, J. Weatheritt, V. Michelassi, and R. D. Sandberg, "Rans turbulence model development using cfd-driven machine learning," Journal of Computational Physics **411**, 109413 (2020).

[19]L. Zhu, W. Zhang, J. Kou, and Y. Liu, "Machine learning methods for turbulence modeling in subsonic flows around airfoils," Physics of Fluids **31** (2019).

[20]S. Bhushan, G. W. Burgreen, W. Brewer, and I. D. Dettwiller, "Assessment of neural network augmented reynolds averaged navier stokes turbulence model in extrapolation modes," Physics of Fluids **35** (2023).

[21]Z. Li, Y. Ju, and C. Zhang, "Machine-learning data-driven modeling of laminar-turbulent transition in compressor cascade," Physics of Fluids **35** (2023).

[22]P. S. Volpiani, M. Meyer, L. Franceschini, J. Dandois, F. Renac, E. Martin, O. Marquet, and D. Sipp, "Machine learning-augmented turbulence modeling for rans simulations of massively separated flows," Physical Review Fluids **6**, 064607 (2021).

[23]I. B. H. Saïdi, M. Schmelzer, P. Cinnella, and F. Grasso, "Cfd-driven symbolic identification of algebraic reynolds-stress models," Journal of Computational Physics **457**, 111037 (2022).

[24]A. Amarloo, M. J. Rincón, M. Reclari, and M. Abkar, "Progressive augmentation of turbulence models for flow separation by multi-case computational fluid dynamics driven surrogate optimization," Physics of Fluids **35** (2023).

[25]K. Duraisamy, G. Iaccarino, and H. Xiao, "Turbulence modeling in the age of data," Annual review of fluid mechanics **51**, 357–377 (2019).

[26]S. L. Brunton, B. R. Noack, and P. Koumoutsakos, "Machine learning for fluid mechanics," Annual review of fluid mechanics **52**, 477–508 (2020).

[27]K. Duraisamy, "Perspectives on machine learning-augmented reynolds-averaged and large eddy simulation models of turbulence," Physical Review Fluids **6**, 050504 (2021).

[28]E. J. Parish and K. Duraisamy, "A paradigm for data-driven predictive modeling using field

inversion and machine learning," Journal of computational physics **305**, 758–774 (2016), publisher: Elsevier.

[29] A. P. Singh, S. Medida, and K. Duraisamy, "Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils," AIAA journal **55**, 2215–2227 (2017).

[30] A. P. Singh and K. Duraisamy, "Using field inversion to quantify functional errors in turbulence closures," Physics of Fluids **28** (2016).

[31] J. R. Holland, J. D. Baeder, and K. Duraisamy, "Field inversion and machine learning with embedded neural networks: Physics-consistent neural network training," in *AIAA Aviation 2019 Forum* (2019) p. 3200.

[32] C. He, Y. Liu, and L. Gan, "A data assimilation model for turbulent flows using continuous adjoint formulation," Physics of fluids **30** (2018).

[33] A. Ferrero, A. Iollo, and F. Larocca, "Field inversion for data-augmented rans modelling in turbomachinery flows," Computers & Fluids **201**, 104474 (2020).

[34] C. Michelen Strofer, X.-L. Zhang, and H. Xiao, "Dafi: An open-source framework for ensemble-based data assimilation and field inversion," Communications in Computational Physics **29**, 1583–1622 (2021).

[35] M. Yang and Z. Xiao, "Improving the k–ω–γ–ar transition model by the field inversion and machine learning framework," Physics of Fluids **32** (2020).

[36] D. Tang, F. Zeng, T. Zhang, C. Yi, and C. Yan, "Improvement of turbulence model for predicting shock-wave–boundary-layer interaction flows by reconstructing reynolds stress discrepancies based on field inversion and machine learning," Physics of Fluids **35** (2023).

[37] T.-X. Zhang, J.-Q. Chen, F.-Z. Zeng, D.-G. Tang, and C. Yan, "Improvement of transition prediction model in hypersonic boundary layer based on field inversion and machine learning framework," Physics of Fluids **35** (2023).

[38] C. Yi, D. Tang, F. Zeng, Y. Li, and C. Yan, "Improvement of the algebraic stress model for separated flows based on field inversion and machine learning," Physics of Fluids **35** (2023).

[39] A. M. Hafez, A. El-Rahman, I. Ahmed, and H. A. Khater, "Field inversion for transitional flows using continuous adjoint methods," Physics of Fluids **34** (2022).

[40] K. Fidkowski, "Correcting an algebraic transition model using field inversion and machine learning," in *AIAA SCITECH 2024 Forum* (2024) p. 2739.

[41] J. Ho and A. West, "Field inversion and machine learning for turbulence modelling applied to three-dimensional separated flows," in *AIAA aviation 2021 forum* (2021) p. 2903.

[42]C. Yan, Y. Zhang, and H. Chen, "Data augmented turbulence modeling for three-dimensional separation flows," Physics of Fluids **34** (2022).

[43]O. Bidar, P. He, S. Anderson, and N. Qin, "Turbulent mean flow reconstruction based on sparse multi-sensor data and adjoint-based field inversion," in *AIAA AVIATION 2022 Forum* (2022) p. 3900.

[44]C. Wu and Y. Zhang, "Enhancing the shear-stress-transport turbulence model with symbolic regression: A generalizable and interpretable data-driven approach," Physical Review Fluids **8**, 084604 (2023).

[45]C. Wu and Y. Zhang, "Development of a generalizable data-driven turbulence model: Conditioned field inversion and symbolic regression," arXiv preprint arXiv:2402.16355 (2024).

[46]O. Bidar, S. Anderson, and N. Qin, "Sensor placement for data assimilation of turbulence models using eigenspace perturbations," Physics of Fluids **36** (2024).

[47]K. J. Fidkowski, "Gradient-based shape optimization for unsteady turbulent simulations using field inversion and machine learning," Aerospace Science and Technology **129**, 107843 (2022).

[48]L. Fang and P. He, "A segregated time-accurate adjoint method for field inversion of unsteady flow," in *AIAA SCITECH 2024 Forum* (2024) p. 0158.

[49]P. He, C. A. Mader, J. R. R. A. Martins, and K. J. Maki, "DAFoam: An open-source adjoint framework for multidisciplinary design optimization with OpenFOAM," AIAA Journal **58**, 1304–1319 (2020).

[50]L. Fang and P. He, "Field inversion machine learning of unsteady flow over a ramp," Mendeley Data (2024), doi: 10.17632/2hnxww96vb.

[51]A. B. Lambe and J. R. R. A. Martins, "Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes," Structural and Multidisciplinary Optimization **46**, 273–284 (2012).

[52]C. M. Rhie and W. L. Chow, "Numerical study of the turbulent flow past an airfoil with trailing edge separation," AIAA Journal **21**, 1525–1532 (1983).

[53]P. Spalart and S. Allmaras, "A one-equation turbulence model for aerodynamic flows," in *30th aerospace sciences meeting and exhibit* (1992).

[54]O. Bidar, P. He, S. Anderson, and N. Qin, "Aerodynamic shape optimisation using a machine learning-augmented turbulence model," in *AIAA SCITECH 2024 Forum* (2024) p. 1231.

[55]S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," (2009).

[56]L. Fang and P. He, “A consistent fixed-point discrete adjoint method for segregated Navier–Stokes solvers,” in *AIAA AVIATION 2022 forum* (2022) p. 4000.

[57]L. Fang and P. He, “A duality-preserving adjoint method for segregated Navier–Stokes solvers,” Journal of Computational Physics (2024).

[58]G. K. Kenway, C. A. Mader, P. He, and J. R. Martins, “Effective adjoint approaches for computational fluid dynamics,” Progress in Aerospace Sciences **110**, 100542 (2019), publisher: Pergamon.

[59]M. Wang, Q. Wang, and T. A. Zaki, “Discrete adjoint of fractional-step incompressible navier-stokes solver in curvilinear coordinates and application to data assimilation,” Journal of Computational Physics **396**, 427–450 (2019).

[60]Y. Yin, P. Yang, Y. Zhang, H. Chen, and S. Fu, “Feature selection and processing of turbulence modeling based on an artificial neural network,” Physics of Fluids **32** (2020).

[61]D. Kraft, “A software package for sequential quadratic programming,” Forschungsbericht-Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt (1988).

[62]A. Wächter and L. T. Biegler, “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming,” Mathematical programming **106**, 25–57 (2006).

[63]P. E. Gill, W. Murray, and M. A. Saunders, “SNOPT: An SQP algorithm for large-scale constrained optimization,” SIAM Review **47**, 99–131 (2005), publisher: SIAM.

[64]S. Rojas-Labanda and M. Stolpe, “Benchmarking optimization solvers for structural topology optimization,” Structural and Multidisciplinary Optimization **52**, 527–547 (2015).

[65]L. Larsson, F. Stern, and M. Visonneau, *Numerical ship hydrodynamics: an assessment of the Gothenburg 2010 workshop* (Springer, 2013).

[66]M. Elkhoury, “Assessment of turbulence models for the simulation of turbulent flows past bluff bodies,” Journal of Wind Engineering and Industrial Aerodynamics **154**, 10–20 (2016).

[67]E. Robertson, V. Choudhury, S. Bhushan, and D. K. Walters, “Validation of OpenFOAM numerical methods and turbulence models for incompressible bluff body flows,” Computers & Fluids **123**, 122–145 (2015), publisher: Elsevier.

[68]P. He, J. R. R. A. Martins, C. A. Mader, and K. Maki, “Aerothermal optimization of a ribbed U-Bend cooling channel using the adjoint method,” International Journal of Heat and Mass Transfer **140**, 152–172 (2019).

[69]Y. Bin, X. Hu, J. Li, S. J. Grauer, and X. I. Yang, “Constrained re-calibration of two-equation reynolds-averaged navier–stokes models,” Theoretical and Applied Mechanics Letters , 100503

(2024).

[70] Y. Bin, G. Huang, R. Kunz, and X. I. Yang, "Constrained recalibration of reynolds-averaged navier–stokes models," AIAA Journal , 1–13 (2023).