AIAA SciTech Forum
6-10 January 2025, Orlando, FL
AIAA SCITECH 2025 Forum

10.2514/6.2025-2049

Field Inversion Machine Learning for Accurate Predictions of Time-Resolved Unsteady Flows Over Airfoils

Zilong Li*, Lean Fang†, Anupam Sharma‡, and Ping He§
Iowa State University, Ames, IA 50011

Reynolds-averaged Navier-Stokes (RANS) models are widely used in practical aerospace engineering designs because of their low computational costs. However, RANS models' assumptions and simplifications may not provide accurate predictions for complex flows with separation and strong adverse pressure gradients. This study uses field inversion machine learning (FIML) to improve RANS turbulence models' accuracy in predicting separated flows over airfoils. Most existing FIML studies focused on steady-state or time-averaged unsteady flows, and this paper is a step forward to improve prediction accuracy for time-resolved unsteady flows. We augment the Spalart-Allmaras (SA) turbulence model's production term with a spatial scalar field. We then compute the above scalar field using a solver-imbedded neural network (NN) model, which takes the local flow features as the inputs at each time step. We optimize the weights and biases of the built-in NN model to minimize the regulated temporal prediction error between the augmented flow solver and reference data. We consider the unsteady separated flow over a NACA0012 airfoil at a large angle of attack. We use only the time series of drag coefficient as the training data, and the trained model can accurately predict the spatial-temporal evolutions of other surface variables, such as lift, pitching moment, and pressure distribution around the airfoil, as well as field variables, such as velocity and pressure. The FIML-trained model has the potential to be generalized to accurately predict airfoil aerodynamics for different shapes and flow conditions.

I. Introduction

Computational fluid dynamics (CFD) is a powerful tool for analyzing three-dimensional flow fields. It offers useful insights into flow physics to support aerospace system designs. CFD can be performed at various levels of fidelity, including low-fidelity Reynolds-averaged Navier-Stokes (RANS) simulations, medium-fidelity large-eddy simulations (LES), and high-fidelity direct numerical simulations (DNS). However, the medium- and high-fidelity simulations are often computationally too expensive for practical engineering design and optimization problems, which may need to run CFD hundreds of times. In the foreseeable future, engineering design will probably still rely on RANS simulations with turbulence models, such as the one-equation Spalart-Allmaras (SA) [1] model and two-equations $k-\varepsilon$ [2] and $k-\omega$ [3] model. However, the RANS turbulence models have many assumptions, often resulting in inaccurate predictions of complex flows, particularly for those involving flow separation and strong adverse pressure gradients.

To address the inaccurate RANS model issue in engineering designs, one can use the multi-fidelity method to calibrate the low-fidelity model's design space response using a handle of high-fidelity data [4–7]. Alternatively, one can use the machine learning (ML) method to correct the defects in a RANS turbulence model's governing equation. In contrast to traditional, human-intuition-based turbulence model development, ML methods utilize data to expedite the development of accurate and broadly applicable turbulence models. Field inversion machine learning (FIML) is a model-consistent approach originally proposed by Duraisamy and co-workers [8–10]. One of FIML's key advantages is its integration of a CFD solver during the training phase (field inversion), which ensures consistency between training and prediction phases at the discretized level, thereby enhancing

*PhD student, Department of Aerospace Engineering, AIAA Student Member.
†PhD student, Department of Aerospace Engineering, AIAA Student Member.
‡Professor, Department of Aerospace Engineering, AIAA Associate Fellow.
§Assistant Professor, Department of Aerospace Engineering, AIAA Senior Member. Email: phe@iastate.edu

both accuracy and generalizability [11]. Additionally, FIML can use various types of training data, including integrated values, surface variables, and sparse or partial field data. A brief review of existing FIML research is presented as follows.

Singh et al. [9] enhanced the Spalart-Allmaras (SA) turbulence model by training it with experimental lift coefficient data to predict flow separation over airfoils. The augmented model significantly improved lift prediction accuracy across various angles of attack and demonstrated generalizability to untrained flow conditions, geometries, and solvers. To improve model consistency, Holland et al. [12] introduced a coupled field inversion and machine learning framework, combining inference and learning processes in a unified approach. He et al. [13] developed a continuous adjoint method for field inversion, showcasing reasonably good performance across diverse 2D and 3D flow scenarios. Ferrero et al. [14] used the wall isentropic Mach number data to augment the SA model for accurate flow predictions in gas turbine cascades. Their trained model demonstrated good prediction accuracy for various unseen Mach numbers and blade geometries.

Instead of using an adjoint-based approach, Michelen Strofer et al. [15] developed an open-source framework that employed the ensemble Kalman filtering (EnKF) method for field inversion. This approach was further explored by Yang and Xiao [16], who used the EnKF method to study the laminar-to-turbulent flow transition over airfoils. They augmented a four equation $k-\omega-\gamma-A_r$ turbulence model, which demonstrated reasonably good performance in predicting the transition location across various angles of attack. The EnKF method has also been applied to improve turbulence model predictions for the interaction between the shock wave and boundary layer [17], laminar-to-turbulent flow transition in hypersonic boundary layer [18], and flow over a hump [19].

Despite the above progress, all the FIML studies cited above have been limited to steady-state flow problems. This limitation is largely due to the significant challenges associated with FIML, particularly the need for an adjoint solver to compute gradients for a large number of design variables [11]. Developing an efficient adjoint solver requires access to source codes, a deep understanding of the CFD solver, and considerable time investment. Although the EnKF method [15] was proposed as an alternative to avoid the need for adjoint solvers in FIML, no EnKF-based FIML study for unsteady flows has been reported, likely due to the high computational cost of generating numerous unsteady flow samples. Recently, Fidkowski [20] 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, our recent work [21] showed that there is no guarantee that a steady-state trained FIML model can accurately predict the time history of general, non-periodic unsteady flow. In that study, we developed a FIML framework to improve the RANS model for predicting time-resolved unsteady flows over a ramp. This paper takes a further step by evaluating the FIML framework's capability to improve the RANS model's performance in predicting time-resolved unsteady flows over airfoils.

The FIML framework has been integrated into our open-source CFD-based optimization framework, DAFoam [22]. Information regarding the download, installation, and utilization of DAFoam can be accessed on its documentation website at `https://dafoam.github.io`. Examples of DAFoam-based steady and unsteady FIML studies can be found in these publications [21, 23–25].

The remainder of the paper is structured as follows. In Section II, we provide detailed explanations of the proposed unsteady FIML framework and its components. The results of the unsteady FIML approach are presented and analyzed in Section III, followed by a summary of our findings in Section IV.

II. Method

In this section, we provide a brief review of the proposed unsteady FIML framework, including unsteady flow simulations, unsteady adjoint computation, the integrated neural network, and the extraction of flow features. More details about the unsteady FIML framework can be found in our previous work [21].

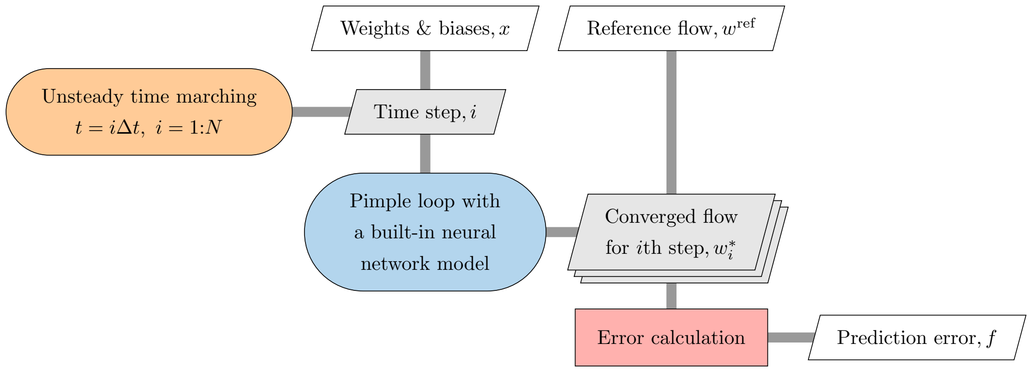
Weights & biases, x
Reference flow, w^{ref}
Unsteady time marching $t = i\Delta t,\ i = 1{:}N$
Time step, i
Pimple loop with a built-in neural network model
Converged flow for ith step, w_i^*
Error calculation
Prediction error, f

Fig. 1 FIML framework for time-resolved unsteady flow. It employs the extended design structure matrix (XDSM) representation introduced by Lambe and Martins [26], where diagonal blocks represent components and off-diagonal blocks denote data exchanges between components. The unsteady FIML optimizes the weights and biases (x) of its integrated neural network model to minimize the regularized prediction error of the CFD solver. This error is calculated as the difference in the selected flow field between the CFD and reference data across all time steps, combined with a regularization term. This figure is adopted from [21].

A. FIML framework for time-resolved unsteady flows

Figure 1 illustrates the FIML framework for time-resolved unsteady flow. We perform unsteady simulations starting from an initial flow and then march the solution with a time step Δt. At each time step, multiple iterations of the PIMPLE loop are performed to minimize flow residuals (details on the PIMPLE algorithm are provided in the next section).

During each iteration of the PIMPLE loop (Fig. 2), a feature extraction component calculates local flow features (η) from the latest flow field (w). A built-in neural network model then takes the flow features η along with weights and biases as inputs to compute the augmentation scalar field β. The augmentation field β will be incorporated into the governing equation of the turbulence model to improve its prediction accuracy. The PIMPLE loop continues until the flow fields for this time step i have converged. The prediction error f is quantified as the difference between the CFD prediction and the reference data across all time steps. To prevent overfitting, a regularization term is added to the objective function to ensure that the augmentation field β remains as close to one as possible. Finally, we solve an inverse problem by minimizing the regularized prediction error f (objective function) through optimization of the weights and biases x (design variables) of the neural network model.

FIML performs large-scale gradient-based optimizations to determine the optimal neural network weights and biases by minimizing the regularized prediction error. In the subsequent subsections, we will provide detailed explanations of the three primary elements of the FIML framework: (1) Unsteady flow simulations, (2) Built-in neural network model for augmentation β field computation, and (3) PIMPLE-Krylov unsteady adjoint formulation for gradient computation.

B. Unsteady flow simulation using the PIMPLE method

We use OpenFOAM's [27] built-in solver `pimpleFoam` to simulate the unsteady flow. It solves three-dimensional, unsteady, turbulent flow governed by the incompressible Navier–Stokes equations:

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

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

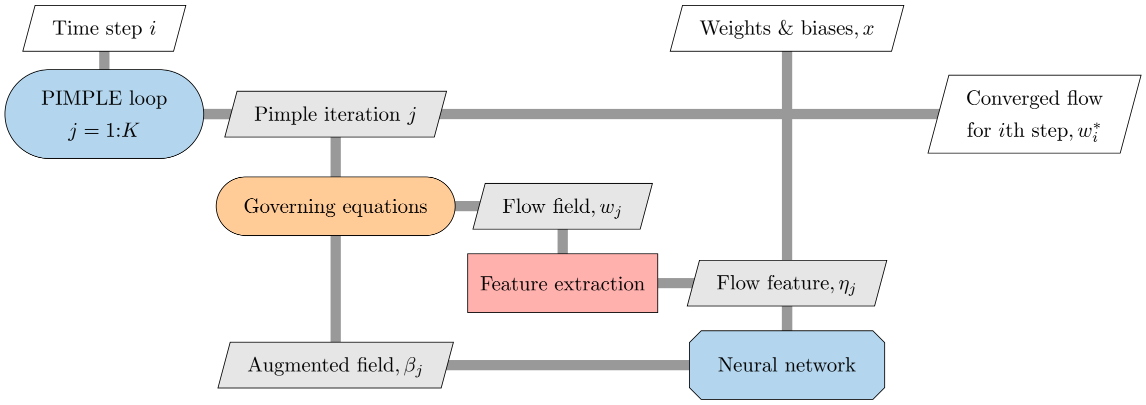

Fig. 2 Detailed structure of the PIMPLE loop component in Fig. 1. It involves performing multiple iterations until all flow residuals are sufficiently small. During each PIMPLE iteration j, the process includes solving the governing equations, calculating the local flow features (η_j), determining the augmentation field β_j through the neural network model, and updating the turbulence model's governing equation with the augmented field in the subsequent iteration. This figure is adopted from [21].

where t is the time, p is the kinematic 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 previously indicated, we solve the Navier-Stokes equations using the PIMPLE method, which combines features of the PISO and SIMPLE algorithms. The procedures involved are briefly outlined below.

First, the momentum equation is discretized, and an intermediate velocity field is solved using either the pressure field obtained from the previous iteration ($p^{t-\Delta t}$) or an initial guess. For simplicity, we assume the first-order Euler scheme is employed 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. By 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, from which an updated pressure field is calculated. The discretized form of the continuity equation is given as:

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

Rather than employing a straightforward linear interpolation, $\boldsymbol{U}_f$ in this equation is determined by interpolating the cell-centered velocity $\boldsymbol{U}_P$, which is derived from the discretized momentum equation (3), onto the cell face using the following method:

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

This idea of momentum interpolation was initially proposed by Rhie and Chow [28] and is effective in mitigating the oscillating pressure (checkerboard) issue resulting from the collocated mesh configuration. Substituting Eq. (7) into Eq. (6), the pressure Poisson equation can be obtained:

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

Solving Eq. (8) provides an updated pressure field p^t. The new pressure field p^t is then 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

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

Here, the $\boldsymbol{H}(\boldsymbol{U})$ term depends on $\boldsymbol{U}$ but has not yet been updated. To account for this, we need to iteratively solve the Eqs. (4) to (10) within the PISO corrector loop. In this study, we employ two iterations of the PISO corrector loop.

To close the system and connect turbulent viscosity to the mean flow variables, a turbulence model is required. The Spalart-Allmaras (SA) model is used, which solves the following equation:

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

where $\tilde{\nu}$ is the modified viscosity, and it is related to the turbulent eddy viscosity as

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

Refer to Spalart and Allmaras [29] for a more detailed description of the terms and parameters in the SA model. In our framework, the production term is multiplied by an augmentation field β.

In addition to the PISO corrector loop mentioned above, the PIMPLE algorithm iteratively solves Eqs. (3) to (11) multiple times until all flow residuals are reduced to a sufficiently low level (PIMPLE corrector loop). To ensure the stability of PIMPLE, it is necessary to under-relax the solutions of momentum equation (3), turbulence equation (11) and the pressure is updated after solving the pressure Poisson equation (8), except for the last PIMPLE corrector loop. The PIMPLE method allows for the use of relatively large time steps (CFL>1). While this feature may not significantly accelerate unsteady flow simulations due to the need for multiple PIMPLE iterations per time step, it is highly advantageous for the unsteady adjoint solver. Larger time steps reduce the number of adjoint equations to solve and the amount of intermediate flow data to manage. In this study, we iterate the PIMPLE corrector loop until flow residuals decrease by 8 orders of magnitude or a maximum of 50 iterations is reached.

C. Built-in neural network model for augmentation field computation

As previously stated, we integrate a multilayer perceptron (MLP) neural network model into the CFD solver, as illustrated in Fig. 3. The neural network takes local flow features η as inputs and outputs the augmentation field β for each iteration. It consists of an input layer, one or more hidden layers, and an output layer. Each

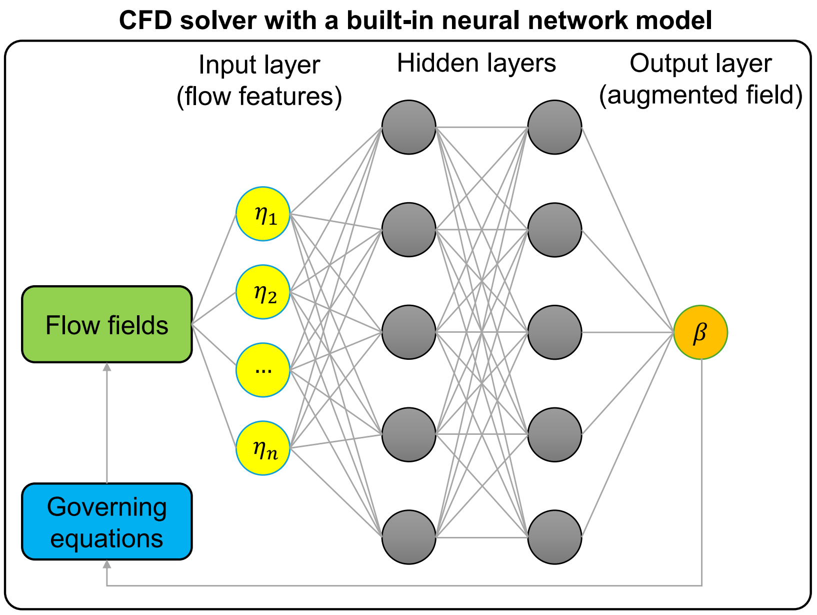
CFD solver with a built-in neural network model
Input layer (flow features)
Hidden layers
Output layer (augmented field)
Flow fields
Governing equations
η_1
η_2
...
η_n
β

Fig. 3 Schematic of the solver-built-in, multilayer perceptron neural network model. This figure is adopted from [21].

neuron is computed as a weighted sum of the neurons from the previous layer and passed through 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 represent 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 neurons in the $k-1$ layer. Various options are available for the activation function f_a, including tanh, sigmoid, and ReLU. In this study, we select the hyperbolic tangent activation function (tanh), which is expressed as

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

We determine the optimal weights and biases using a sequential quadratic programming (SQP) algorithm, which employs second-order derivatives to compute search directions.

D. PIMPLE-Krylov unsteady adjoint formulation for gradient computation

As previously stated, the PIMPLE flow simulation maintains tight convergence of all flow residuals 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, $\boldsymbol{x} \in \mathbb{R}^{n_x}$ represents 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. In this study, we employ an implicit second-order time discretization scheme for all time steps except the initial one, where a first-order time scheme is utilized. In the primal unsteady flow solution, Eq. (15) is solved in a forward manner to determine the state variable for all time steps, i.e., $\boldsymbol{w}^1, \boldsymbol{w}^2, \dots, \boldsymbol{w}^K \in \mathbb{R}^{n_w}$.

The objective function F depends on both the design variables $\boldsymbol{x}$ and the state variable $\boldsymbol{w}$ solved in Eq. (15). In many applications, including the current study, the objective function F can be represented as the average of a time series, i.e.,

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

where for each $1 \leq i \leq K$, f^i depends only on the design variables $\boldsymbol{x}$ and the corresponding state variable $\boldsymbol{w}^i$ at that specific time step. This formulation allows the partial derivative $\partial F / \partial \boldsymbol{w}$ to 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 typical forms of objective functions, such as the variance of a time series, the partial derivatives of F can be similarly simplified.

To compute the total derivative $\mathrm{d}F/\mathrm{d}\boldsymbol{x}$ for gradient-based optimization, the chain rule is applied in the following manner:

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

where the partial derivatives $\partial F/\partial \boldsymbol{x}$ and $\partial F/\partial \boldsymbol{w}$ are relatively cheap to compute because they only entail explicit computations. In contrast, evaluating the total derivative $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$ matrix is computationally expensive because $\boldsymbol{w}$ and $\boldsymbol{x}$ are implicitly connected through the residual equations $\boldsymbol{R}(\boldsymbol{w}, \boldsymbol{x}) = 0$.

To compute $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$, we can apply the chain rule for $\boldsymbol{R}$. Given that the residual equations should always hold, irrespective 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, the following linear system is derived:

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

Then substituting the solution for $\mathrm{d}\boldsymbol{w}/\mathrm{d}\boldsymbol{x}$ from Eq. (20) into Eq. (18) provides:

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

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*. Subsequently, the total derivative can be computed 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 only need to solve the adjoint equation once for each objective function. Therefore, the computational cost is independent of the number of design variables but is instead proportional to the number of objective functions. This method of computing derivatives, as introduced, is also known as the *adjoint method*. It is beneficial for field inversion since only one objective function is typically present, even though thousands of design variables may be employed.

The adjoint equation given 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 manner:

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

which effectively breaks down the original adjoint equation Eq. (22) into K much smaller sub-equations. The right-hand side terms in Eq. (25) can be efficiently computed using 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 immediately after solving 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 manner, then they are passed to the right-hand side of the sub-equation for $\boldsymbol{\psi}^i$. Note that we solve the adjoint equation (25) in a reverse mode, and computing the matrix-vector products requires access to state variables for all time steps. However, storing all state variables in memory is excessively costly. Therefore, we write 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 from the disk.

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:

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

Hence, we can calculate the total derivative accumulatively as we sequentially solve the sub-equations in Eq. (25). This on-the-fly computation of total derivatives is advantageous as it obviates the need to store 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

Table 1 Four local flow features used in this study.

Feature	Formulation	Description	Scaling
η_1	P/D	Ratio of the turbulence production and destruction term	1e-8
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 2 Optimization formulation for the unsteady FIML problem.

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

(PETSc) [30] library to solve the adjoint linear equations. The GMRES solver achieves quadratic convergence and is notably faster than the fixed-point adjoint solver (linear convergence) employed in our previous studies [31–33]. In addition, the GMRES solver's convergence does not require the linear iteration matrix's eigenvalues to be within the unit circle. This renders it more robust in practical scenarios, especially when we need to solve the adjoint equations repeatedly. For the details implementation of GMRES, one can refer to the study of Fang and He [21].

III. Results and Discussion

In this subsection, we consider the spatial-temporal evolution of unsteady flow over a NACA0012 airfoil, which is caused by a sudden change in the angle of attack (from 20° to 25°). We first run the unsteady CFD simulation 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 drag around the airfoil as the training data. We will evaluate the trained model's prediction accuracy for drag and other surface variables, such as lift, pressure, and pitching moment, as well as field variables, such as velocity and pressure fields, which are not used in training.

A. Time-resolved CFD simulations for unsteady flows over the NACA0012 airfoil

We use unsteady turbulent flows over the NACA0012 airfoil as the benchmark, as shown in Fig. 4. The airfoil has a chord length of $c = 1.0$ m and an angle of attack (α) of 25° (α is initially set as 20°). The computational domain is a cylinder with a radius of $20c$ for the base and $0.1c$ in height, respectively. We generate a coarse structured mesh with 4032 cells. The inlet velocity is $U_{in} = 10$ m/s, and the corresponding Reynolds number is 10^5. We first run the flow solver for 100 seconds until the unsteady flow reaches equilibrium. Then, we use the above equilibrium flow field as the initial condition, twist the α to 25°, and run unsteady simulations for an additional 2 s. Note that in the original SA model and trained FIML model, we use the same field variables obtained from the $k-\omega$ SST model as the initial conditions. The airfoil drag time series from 0 to 2 s is used for the unsteady FIML training data. The time step size is $\Delta t = 0.001$ s, with a corresponding CFL number of about 5. As previously mentioned, the PIMPLE algorithm allows unsteady simulations with CFL > 1. Using a relatively large CFL number should not significantly compromise 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 LES and 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 to make its prediction results match those of the $k-\omega$ SST model. Note that in practical FIML applications, experimental data or high-fidelity data (such as LES and DNS), are commonly

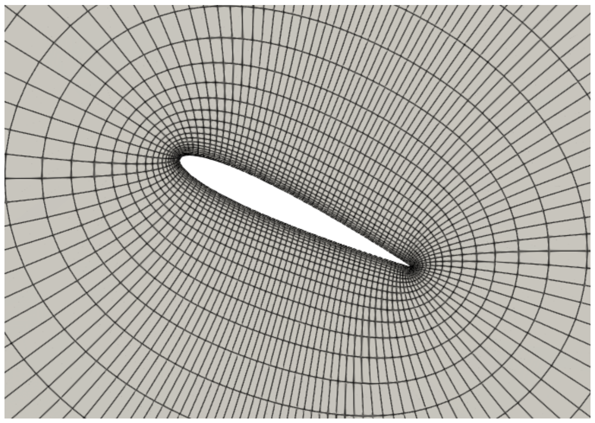

Fig. 4 Structured mesh for the unsteady flow over a NACA0012 airfoil with 4032 cells. The airfoil has an angle of attack $\alpha = 25°$.

used as reference values. To simplify the analysis (e.g., mesh interpolation), this paper chooses the SST model's simulation data as the reference to showcase the proposed unsteady FIML framework. However, the proposed FIML framework can be expanded to incorporate experimental or LES and DNS data.

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

The objective of this paper is to showcase the unsteady FIML framework by enhancing the original SA model to predict both surface and field variables to match those of the $k - \omega$ SST model. To achieve this, we establish a composite objective function F according to the following:

$$F = \frac{1}{K} \sum_{t=1:K} [c_1 (f_t^{\text{CFD}} - f_t^{\text{ref}})^2 + \frac{c_2}{N_i} \sum_{i=1:N_i} (\beta_{i,\,t} - 1)^2] \tag{27}$$

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 airfoil drag time series as f. To avoid over-fitting, we also add a regularization term to minimize the spatial variations of the augmentation scalar field β with respect to its original value (1.0). So, the subscript i is the mesh cell index, with N_i 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 regularization.

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 [34]). We compute the gradients using the proposed PIMPLE-Krylov adjoint method. As mentioned earlier, the PIMPLE-Krylov method solves adjoint equations in reverse, starting from the last time instance $t = 2$ s. The adjoint solver uses the same step size $\Delta t = 0.001$ s as the primal solver. To speed up the Krylov-based adjoint equation solution, we pre-compute the preconditioners with a time interval of 0.5 s. We require the adjoint equation residuals to drop five orders of magnitude for each time step.

Through trial and error, we choose four local flow features as inputs for the neural network, as listed in Table 1. Although these features are already normalized and non-dimensional, we further scale them to make their standard deviation (among all mesh cells and time steps) close to one. This scaling, as mentioned earlier, facilitates the gradient-based optimization in FIML. To better capture the relationship between the local flow features and the augmentation scalar field β, our built-in neural network model includes two hidden layers, each with 20 neurons. As previously stated, the weights and biases in the neural network serve as design variables, and we have 540 design variables in total. Note that the number of design variables is determined by the neural network's architecture and does not depend on the number of mesh cells or time steps. The optimization

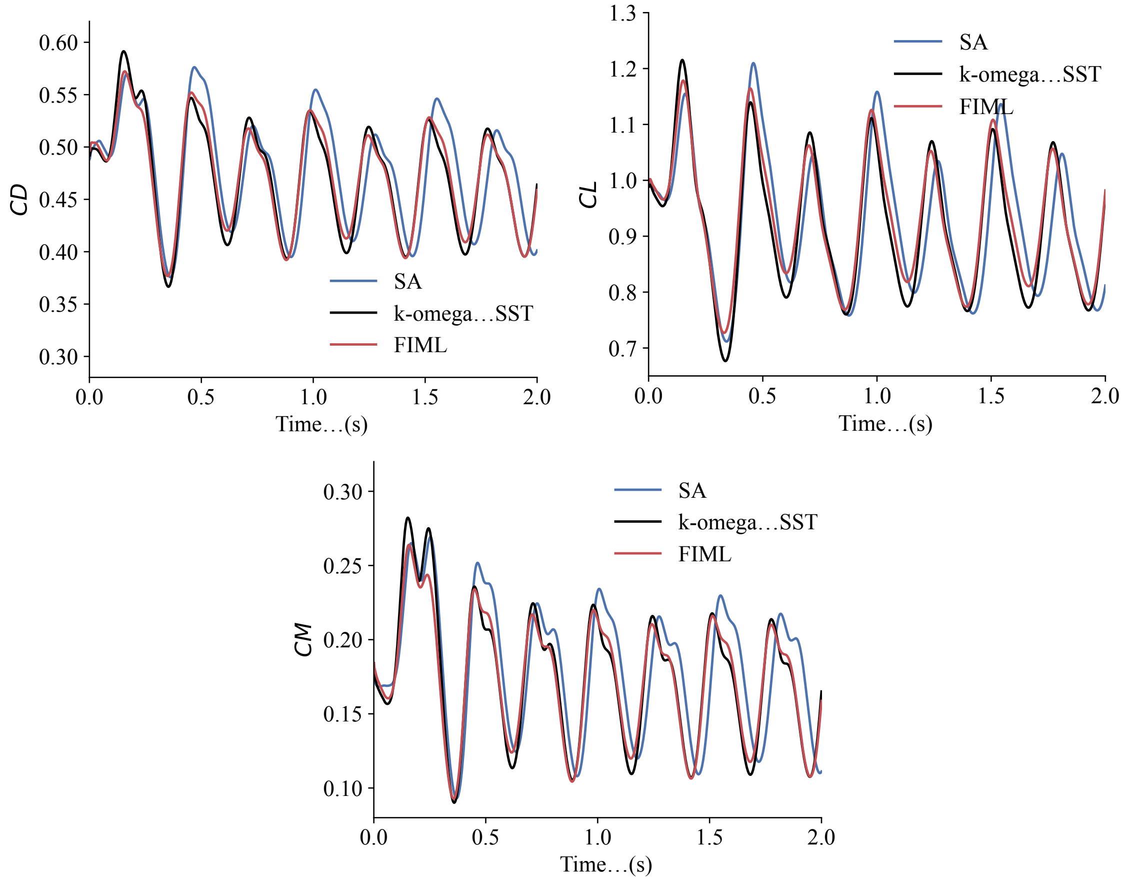

Fig. 5 Temporal evolution of airfoil drag, lift, and pitching moment among SA (original model), $k-\omega$ SST (reference data), and FIML (trained model). The FIML-trained model has a much better agreement with the reference data than the original model.

formulation for the unsteady FIML problem is summarized in Table 2. The unsteady FIML optimization runs for 34 iterations. The baseline and optimized objective functions are 2.22E-02 and 2.06E-03, respectively; the objective function reduces by 90.7%. The optimality drops more than three orders of magnitude, reducing from 2.4E-01 to 1.5E-04. This indicates that the optimization converges tightly.

Fig. 5 illustrates the temporal evolution of airfoil drag, lift, and pitching moment among SA (original model), $k-\omega$ SST (reference data), and FIML (trained model). All surface variables decrease after increasing the angle of attack from 20° to 25°, indicating that the airfoil is in a stall condition. The flow experiences a transient period and does not reach a full equilibrium state within the simulation time. The FIML-trained model has a much better agreement with the reference data than the original model. For the beginning 0.5 s, the SA model agrees well with the reference data, and this is expected because we use the same field variables obtained from the $k-\omega$ SST model as the initial conditions. The amplitude and phase of the surface variables in the original SA model deviate from the reference data after 0.5 s. However, the unsteady-trained-FIML model captures the surface variables' phase well through the simulation process, except that some peaks and troughs are not well captured. Overall, the unsteady-trained-FIML model has a better performance than the original model.

Fig. 6 plots the comparison of pressure around the airfoil at various time instances. The original and trained model agree well with the reference data at $t = 0.5$ s. As mentioned previously, we use the same field variables obtained from the $k-\omega$ SST model as the initial conditions, so their differences are small from the beginning. But as time evolves, the trained model has a much better agreement with the reference data than the original

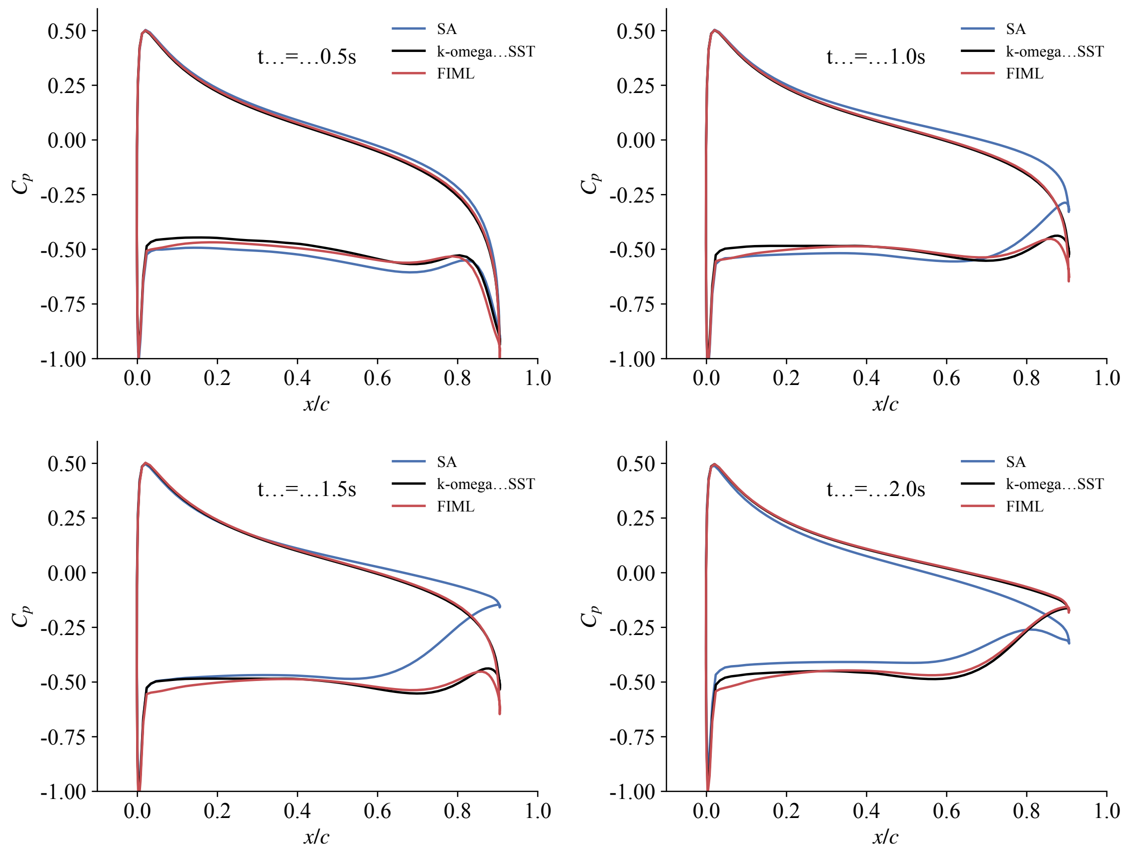

Fig. 6 Pressure around the airfoil at various time instances among SA (original model), $k-\omega$ SST (reference data), and FIML (trained model). The FIML-trained model has a much better agreement with the reference data than the original model at various time instances.

model. Note that the surface pressure is not used as training data in our unsteady FIML model; this indicates that the FIML-trained model can accurately predict variables not used in the training.

To further evaluate the prediction capability of the FIML-trained model, we show the contours of velocity and pressure fields in Figs. 7 and 8. In the beginning, both the original and trained models present visually non-distinguishable spatial-temporal variations of velocity and pressure fields as the reference model (Fig. 7 and 8 left). But as time goes on, the vortex structure behind the trailing edge of the airfoil evolves differently (Fig. 7 and 8 right). It clearly shows that the trained model agrees better with the reference data than the original model. To better quantify this good agreement, we plot the velocity profiles $0.5c$ downstream of the trailing edge at various time instances, which is shown in Fig. 9. The original model has large prediction errors after 0.5 s, while The FIML-trained model agrees reasonably well with the reference data at all time instances.

The above results indicate that using only the drag time series as the training data can improve both the velocity and pressure 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 drag around the airfoil can lead to the corrected velocity and pressure 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 [14, 24, 35]). 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 our proposed FIML's applications.

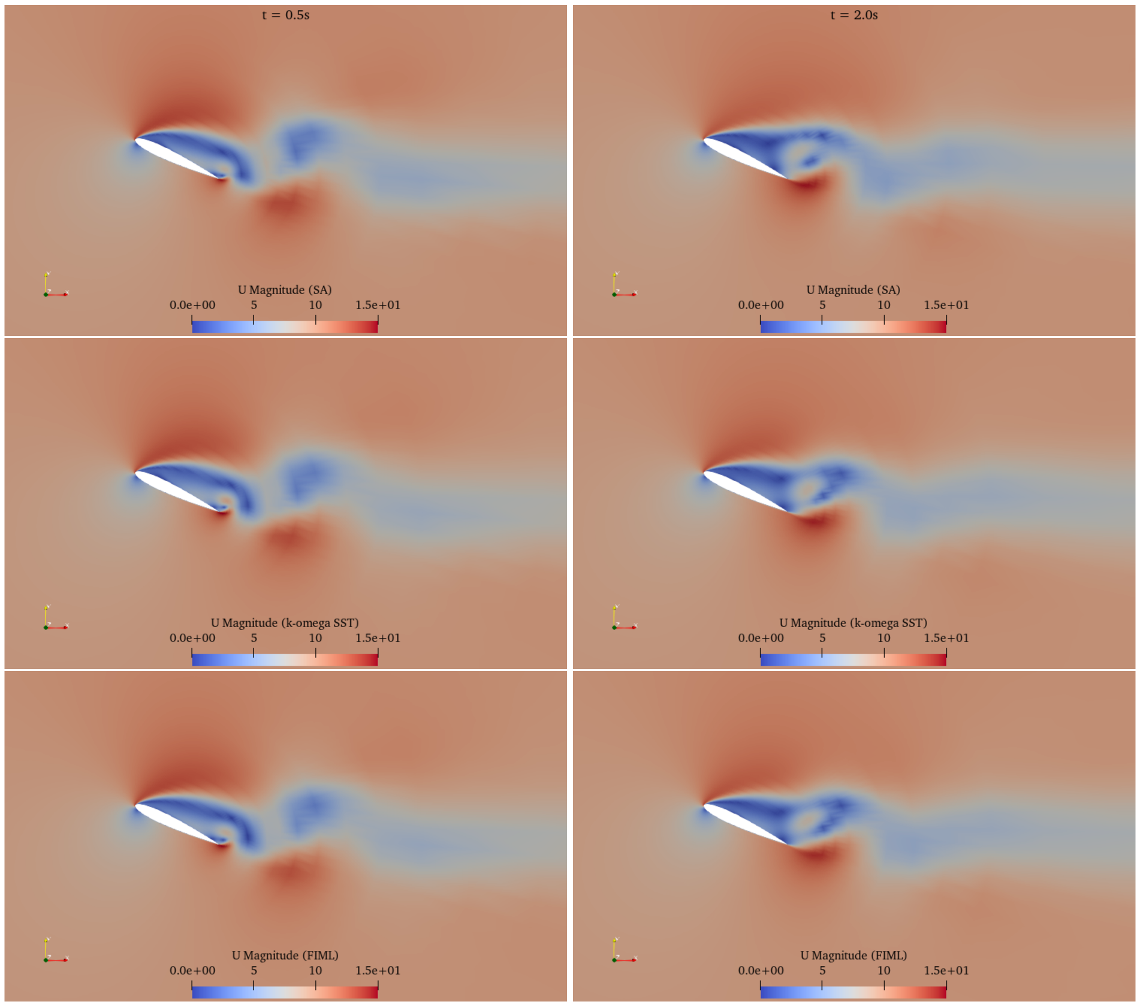

Fig. 7 Velocity fields at 0.5s (left) and 2.0s (right) among SA (original model), $k - \omega$ SST (reference data), and FIML (trained model). The FIML-trained model has a much better agreement with the reference data than the original model at various time instances.

IV. Conclusion

In this study, we use field inversion machine learning (FIML) to improve the SA turbulence model for predicting the spatial-temporal evolution of unsteady flow. This framework augments the SA model's production term with a scalar field (β). Then, it utilizes a built-in neural network model, which takes the local flow features as the inputs, to compute the augmentation β field. The framework then solves an inverse problem by optimizing the neural network's weights and biases to minimize the regularized prediction error of the augmented turbulence model. The prediction error is quantified as the spatial-temporal flow field difference between the CFD simulations and the reference data. To prevent overfitting, the framework incorporates a regularization term into the objective function.

To evaluate the unsteady FIML framework, we consider the unsteady flow over a NACA0012 airfoil under a large angle of attack. We only use the drag time series as the training data, and the FIML-trained model can accurately predict other surface variables, such as lift, pitching moment, and pressure distribution around the airfoil, as well as field variables, such as velocity and pressure fields. In practical FIML applications, it is more common to use either experimental data or high-fidelity data (such as LES and DNS) as reference values. Here, we use the simulation data from the $k - \omega$ SST model as a reference to showcase FIML's prediction capability

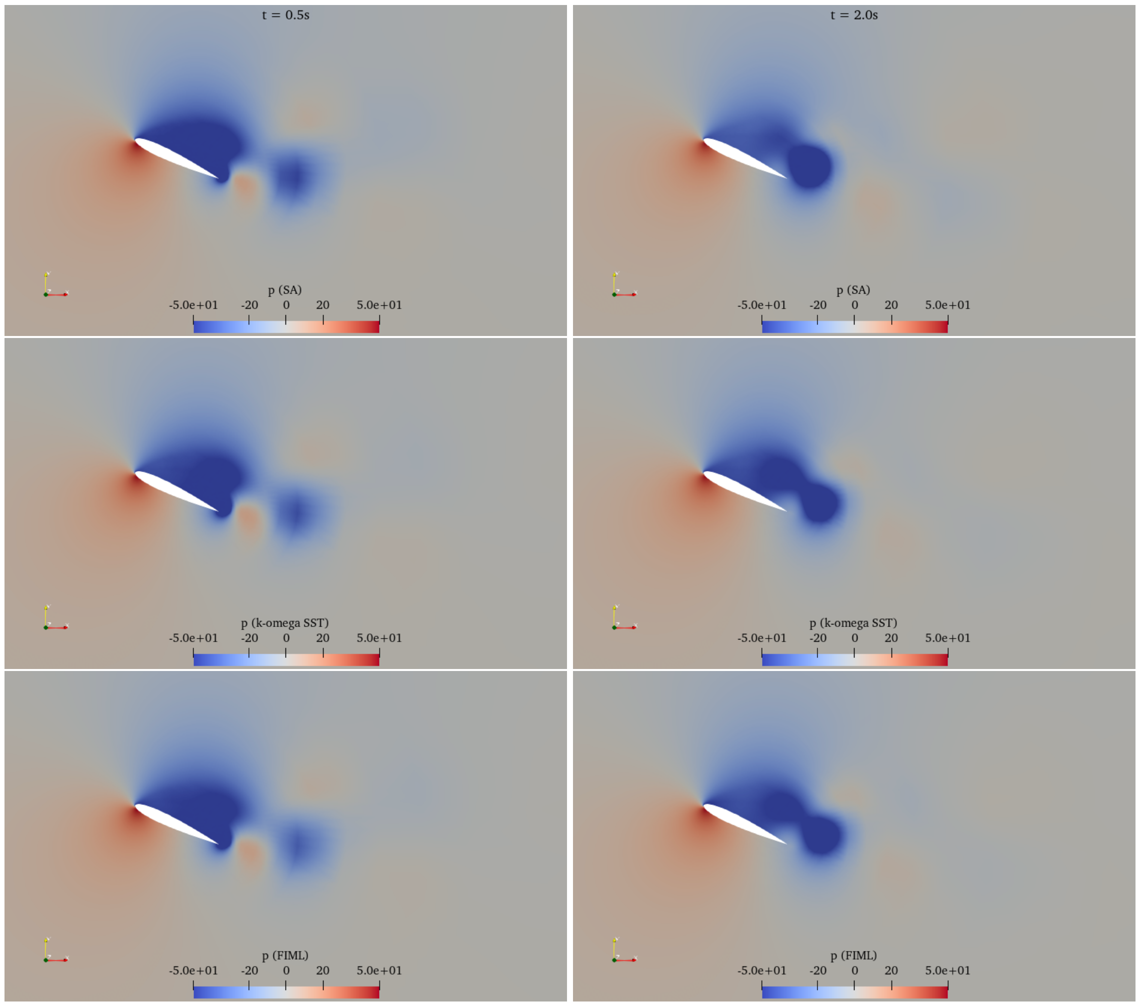

Fig. 8 Pressure fields at 0.5s (left) and 2.0s (right) among SA (original model), $k-\omega$ SST (reference data), and FIML (trained model). The FIML-trained model has a much better agreement with the reference data than the original model at various time instances.

for unsteady flow. In the future, we will extend the proposed FIML framework to utilize high-fidelity data as the reference values. The FIML-trained model has the potential to be generalized to accurately predict airfoil aerodynamics for different shapes and flow conditions.

V. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Number 2223676. This work used the Stampede 3 supercomputer at Texas Advanced Computing Center through allocation ATM-140019 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by U.S. National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

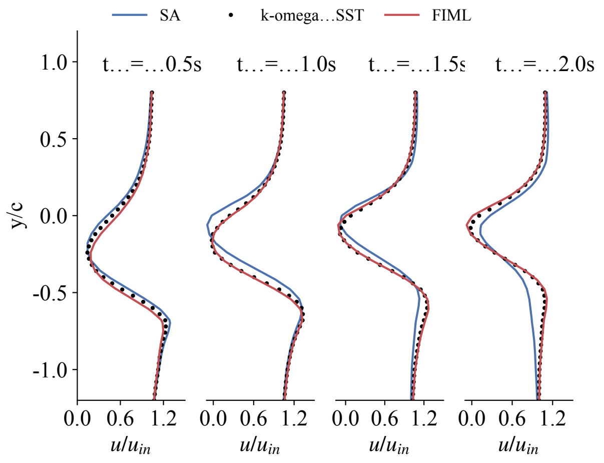

Fig. 9 Velocity profile 0.5c downstream of the trailing edge at various time instances. The FIML-trained model agrees much better with the reference data than the original model at various time instances.

References

[1] Spalart, P., and Allmaras, S., "A one-equation turbulence model for aerodynamic flows," *30th aerospace sciences meeting and exhibit*, 1992, p. 439.

[2] Launder, B. E., and Sharma, B. I., "Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc," *Letters in heat and mass transfer*, Vol. 1, No. 2, 1974, pp. 131–137.

[3] Wilcox, D. C., "Reassessment of the scale-determining equation for advanced turbulence models," *AIAA journal*, Vol. 26, No. 11, 1988, pp. 1299–1310.

[4] Robinson, T. D., Eldred, M. S., Willcox, K. E., and Haimes, R., "Surrogate-based optimization using multifidelity models with variable parameterization and corrected space mapping," *AIAA Journal*, Vol. 46, No. 11, 2008, pp. 2814–2822. https://doi.org/10.2514/1.36043.

[5] Leifsson, L., and Koziel, S., "Multi-fidelity design optimization of transonic airfoils using physics-based surrogate modeling and shape-preserving response prediction," *Journal of Computational Science*, Vol. 1, No. 2, 2010, pp. 98–106. Publisher: Elsevier.

[6] Feldstein, A., Lazzara, D., Princen, N., and Willcox, K., "Multifidelity data fusion: Application to blended-wing-body multidisciplinary analysis under uncertainty," *AIAA Journal*, Vol. 58, No. 2, 2020, pp. 889–906.

[7] Nagawkar, J. R., Leifsson, L. T., and He, P., "Aerodynamic shape optimization using gradient-enhanced multifidelity neural networks," *AIAA SciTech 2022 Forum*, 2022, p. 2350.

[8] Parish, E. J., and Duraisamy, K., "A paradigm for data-driven predictive modeling using field inversion and machine learning," *Journal of computational physics*, Vol. 305, 2016, pp. 758–774. Publisher: Elsevier.

[9] Singh, A. P., Medida, S., and Duraisamy, K., "Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils," *AIAA journal*, Vol. 55, No. 7, 2017, pp. 2215–2227.

[10] Singh, A. P., and Duraisamy, K., "Using field inversion to quantify functional errors in turbulence closures," *Physics of Fluids*, Vol. 28, No. 4, 2016.

[11] Duraisamy, K., "Perspectives on machine learning-augmented Reynolds-averaged and large eddy simulation models of turbulence," *Physical Review Fluids*, Vol. 6, No. 5, 2021, p. 050504.

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

[13] He, C., Liu, Y., and Gan, L., "A data assimilation model for turbulent flows using continuous adjoint formulation," *Physics of fluids*, Vol. 30, No. 10, 2018.

[14] Ferrero, A., Iollo, A., and Larocca, F., "Field inversion for data-augmented RANS modelling in turbomachinery flows," *Computers & Fluids*, Vol. 201, 2020, p. 104474.

[15] Michelen Strofer, C., Zhang, X.-L., and Xiao, H., "DAFI: An Open-Source Framework for Ensemble-Based Data Assimilation and Field Inversion," *Communications in Computational Physics*, Vol. 29, No. 5, 2021, pp. 1583–1622. https://doi.org/https://doi.org/10.4208/cicp.OA-2020-0178.

[16] Yang, M., and Xiao, Z., "Improving the k–ω–γ–Ar transition model by the field inversion and machine learning framework," *Physics of Fluids*, Vol. 32, No. 6, 2020.

[17] Tang, D., Zeng, F., Zhang, T., Yi, C., and Yan, C., "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*, Vol. 35, No. 6, 2023.

[18] Zhang, T.-X., Chen, J.-Q., Zeng, F.-Z., Tang, D.-G., and Yan, C., "Improvement of transition prediction model in hypersonic boundary layer based on field inversion and machine learning framework," *Physics of Fluids*, Vol. 35, No. 2, 2023.

[19] Yi, C., Tang, D., Zeng, F., Li, Y., and Yan, C., "Improvement of the algebraic stress model for separated flows based on field inversion and machine learning," *Physics of Fluids*, Vol. 35, No. 11, 2023.

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

[21] Fang, L., and He, P., "Field inversion machine learning augmented turbulence modeling for time-accurate unsteady flow," *Physics of Fluids*, Vol. 36, No. 5, 2024.

[22] He, P., Mader, C. A., Martins, J. R. R. A., and Maki, K. J., "DAFoam: An open-source adjoint framework for multidisciplinary design optimization with OpenFOAM," *AIAA Journal*, Vol. 58, No. 3, 2020, pp. 1304–1319. https://doi.org/10.2514/1.J058853.

[23] Bidar, O., He, P., Aderson, S., and Qin, N., "An open-source adjoint-based field inversion tool for data-driven RANS modelling," *AIAA aviation 2022 forum*, 2022.

[24] Wu, C., and Zhang, Y., "Development of a Generalizable Data-driven Turbulence Model: Conditioned Field Inversion and Symbolic Regression," *AIAA Journal*, 2024.

[25] Wu, C., and Zhang, Y., "Enhancing the shear-stress-transport turbulence model with symbolic regression: A generalizable and interpretable data-driven approach," *Physical Review Fluids*, Vol. 8, No. 8, 2023, p. 084604. https://doi.org/10.1103/PhysRevFluids.8.084604.

[26] Lambe, A. B., and Martins, J. R. R. A., "Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes," *Structural and Multidisciplinary Optimization*, Vol. 46, 2012, pp. 273–284. https://doi.org/10.1007/s00158-012-0763-y.

[27] Weller, H. G., Tabor, G., Jasak, H., and Fureby, C., "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Computers in Physics*, Vol. 12, No. 6, 1998, pp. 620–631. Publisher: AIP Publishing.

[28] Rhie, C. M., and Chow, W. L., "Numerical study of the turbulent flow past an airfoil with trailing edge separation," *AIAA Journal*, Vol. 21, No. 11, 1983, pp. 1525–1532. https://doi.org/10.2514/3.8284.

[29] Spalart, P., and Allmaras, S., "A one-equation turbulence model for aerodynamic flows," *30th aerospace sciences meeting and exhibit*, 1992. https://doi.org/10.2514/6.1992-439.

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

[31] Fang, L., and He, P., "A consistent fixed-point discrete adjoint method for segregated Navier–Stokes solvers," *AIAA AVIATION 2022 forum*, 2022, p. 4000.

[32] Fang, L., and He, P., "A duality-preserving adjoint method for segregated Navier–Stokes solvers," *Journal of Computational Physics*, 2024, p. (under review).

[33] Fang, L., and He, P., "A Segregated Time-Accurate Adjoint Method for Field Inversion of Unsteady Flow," *AIAA SCITECH 2024 Forum*, 2024, p. 0158.

[34] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Review*, Vol. 47, No. 1, 2005, pp. 99–131. https://doi.org/10.1137/S0036144504446096, publisher: SIAM.

[35] Hafez, A. M., El-Rahman, A., Ahmed, I., and Khater, H. A., "Field inversion for transitional flows using continuous adjoint methods," *Physics of Fluids*, Vol. 34, No. 12, 2022.