Contents lists available at ScienceDirect

Aerospace Science and Technology

www.elsevier.com/locate/aescte

Rapid airfoil design optimization via neural networks-based parameterization and surrogate modeling

Xiaosong Du [a,*], Ping He [b], Joaquim R.R.A. Martins [a]

[a] Department of Aerospace Engineering, University of Michigan, United States of America
[b] Department of Aerospace Engineering, Iowa State University, United States of America

ARTICLE INFO

Article history:
Received 29 December 2020
Received in revised form 15 March 2021
Accepted 26 March 2021
Available online 31 March 2021
Communicated by Cummings Russell

ABSTRACT

Aerodynamic optimization based on computational fluid dynamics (CFD) is a powerful design approach because it significantly reduces the design time compared with the human manual design. However, CFD-based optimization can still take hours to converge because it requires repeatedly running computationally expensive flow simulations. To further shorten the design optimization time, we propose a fast, interactive design framework that allows us to complete an airfoil aerodynamic optimization within a few seconds. This framework is made efficient through a B-spline-based generative adversarial network model for shape parameterization, which filters out unrealistic airfoils for a reduced design space that contains all relevant airfoil shapes. Moreover, we use a combination of multilayer perceptron, recurrent neural networks, and mixture of experts for surrogate modeling to enable both scalar (drag and lift) and vector (pressure distribution) response predictions for a wide range of Mach numbers (0.3 to 0.7) and Reynolds numbers (10^4 to 10^{10}). To verify our proposed framework, we compare the optimization results with the ones computed by direct CFD-based optimization for subsonic and transonic conditions. The results show that the optimal designs and the aerodynamic quantities (lift, drag, and pressure distribution) obtained by our proposed framework agree well with the ones computed by direct CFD-based optimizations and evaluations. The proposed framework is being integrated into a web-based interactive aerodynamic design framework that allows users to predict drag, lift, moment, pressure distribution, and optimal airfoil shapes for a wide range of flow conditions within seconds.

1. Introduction

Aerodynamic design optimization based on computational fluid dynamics (CFD) simulations partially automates the design process and significantly improves the aerodynamic performance compared with human-supervised design [1,2]. Both gradient-free and gradient-based algorithms have been used for aerodynamic optimization. Gradient-free algorithms are straightforward to use, but because aerodynamic shape optimization problems often involve a large number of design variables, the cost of using these algorithms is prohibitive [3]. Gradient-based algorithms scale much better with the number of design variables, especially when gradients are computed efficiently using an adjoint method [4–6]. However, even with these advances in modeling and optimization approaches, the design process can still be computationally expensive, requiring high-performance parallel computing hardware. The bottleneck is that even gradient-based optimization methods typically require hundreds of computationally expensive simulations, which can take hours or even days to complete.

Researchers have proposed using surrogate modeling algorithms to reduce the optimization's computational cost and accelerate the design process. While the upfront cost of training the surrogate model is considerable, this approach can pay off if the surrogate model performs many optimizations. Surrogate models aim to replace computationally expensive simulation models with much faster evaluations that are accurate enough [7,8]. Widely used surrogate models include data-driven surrogates and multi-fidelity surrogates. Data-driven surrogates treat the simulation model as a black box and directly construct the mapping between the input and output spaces using simple algebraic expressions [9]. Multi-fidelity surrogate modeling uses simulation models of different fidelities by adding correction terms to the low-fidelity predictions or considering the correlation between the points to be predicted and the multi-fidelity training data sets [10]. In contrast to the traditional surrogate models, such as kriging and manifold mapping methods, deep neural networks (DNN) have emerged as a promising type of surrogates to capture nonlinear data pattern [11,12].

* Corresponding author.
E-mail address: xsdu@umich.edu (X. Du).

https://doi.org/10.1016/j.ast.2021.106701
1270-9638/© 2021 Elsevier Masson SAS. All rights reserved.

Feldstein et al. [13] fused multiple information sources to support decision making and risk identification in early-stage multidisciplinary design optimization. The fused multi-fidelity surrogate model used a fidelity-weighted combination of Gaussian process surrogates. The weighting factors took into account both the quality of Gaussian process approximation and the designers' confidence in the underlying information source. Liem and Martins [14] proposed a mixture of experts approach to combine gradient-enhanced kriging models based on the divide-and-conquer principle. In addition, they applied an adaptive sampling algorithm to span the input parameter space efficiently. The proposed approach was successfully demonstrated on conventional and unconventional aircraft configurations, where the surrogate model was used to represent the aircraft performance over a range of flight conditions. Rumpfkeil and Beran [15] applied multifidelity sparse polynomial chaos expansion models to represent the critical flutter dynamic pressures with respect to Mach number, angle of attack, and thickness-to-chord ratio. Compressed sensing was used for sparse representation to construct the proposed model via hybrid multiplicative and additive bridge functions. They managed to prove the predictive performance to be effective on analytic test functions and a complex aeroelastic model.

DNN surrogates show promising potential for regression tasks and handle large data sets via batch optimization [16,17]. In addition, DNN models are flexible at the dimension of the model response. Raissi et al. [18,19] developed the physics informed neural networks—neural networks trained for supervised learning while obeying physics laws described by general partial differential equations. They adopted DNN and backpropagation-based derivative information to infer the solution of general partial differential equations. The proposed approach successfully dealt with a series of benchmark problems, including incompressible flow and dynamic vortex shedding past a circular cylinder. Bhatnagar et al. [20] constructed convolutional neural networks (CNN) to predict the velocity and pressure fields of airfoils. They investigated the network architecture effectiveness in predicting flow fields, and concluded that shared encoding and decoding was computationally more efficient for flow field prediction compared to separated alternatives. Sekar et al. [21] approximated the pressure coefficient distribution (C_p) using deep CNN surrogate to realize airfoil inverse designs. In their latest work, they extracted airfoil geometric parameters through CNN model, fed the extracted parameters to multilayer perceptron (MLP) networks together with operating flight conditions and yielded satisfactory flow field prediction [22]. Chen and Fuge [23] proposed to parameterize airfoil shapes using a Bézier curve-based generative adversarial networks (BézierGAN) model, which intelligently reduced the design space by filtering out unrealistic airfoil shapes. The BézierGAN model combined with Gaussian process regression was demonstrated on airfoil design optimizations to maximize the lift-to-drag ratio and found better optimal designs using fewer optimization iterations than other parameterization methods (such as PARSEC and proper orthogonal decomposition) [24]. Azabi et al. [25] developed a multi-objective optimization framework utilizing a DNN model and exhibited the advantages on the aerodynamic design of an unmanned aerial vehicle. In particular, the DNN model was trained to classify the trial solutions suggested by the optimizer as valid or invalid, which accelerated the optimization by avoiding evaluating the invalid candidates. The efforts cited above were successful and promising but restricted to small ranges of flight conditions or limited geometric parameter space. Furthermore, none of these efforts focused on the lift-constrained minimum drag optimization, which is an important problem in aerodynamic design.

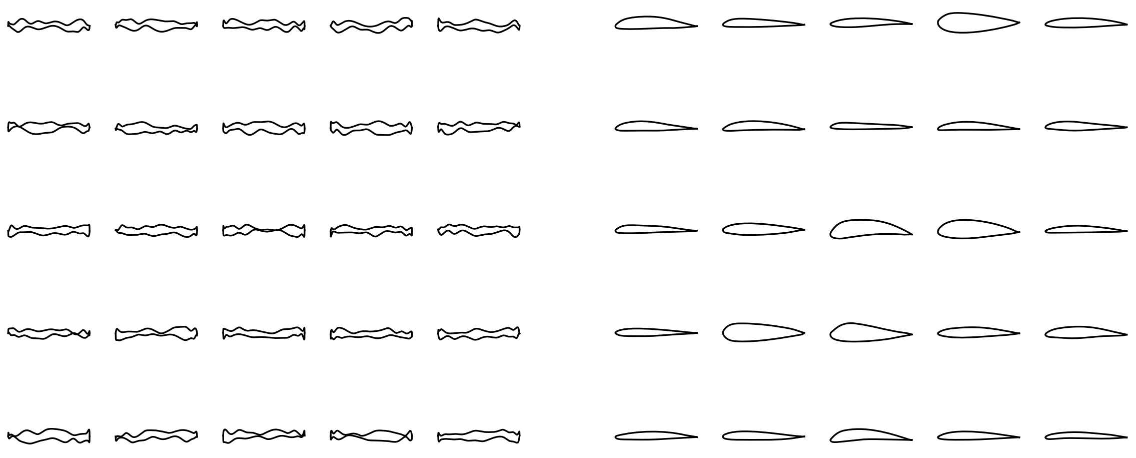
To fill the current gap in neural networks-based aerodynamic design optimization, we have been spending efforts on surrogate modeling for wide ranges of airfoil shapes and flight conditions. This enables fast interactive airfoil optimization through the Webfoil[1] web interface. In addition to the surrogate-model-based optimization, Webfoil also serves as an airfoil database. Li et al. [26] applied singular value decomposition (SVD) for airfoil parameterization. They combined the gradient-enhanced kriging with partial least squares (GEK-PLS) [27] and mixture of experts to predict aerodynamic drag coefficient (C_d), lift coefficient (C_l) and moment coefficient (C_m). The constructed GEK-PLS models yielded a good agreement for both subsonic and transonic regimes. Bouhlel et al. [11] proposed a gradient-enhanced DNN method for the surrogate modeling of C_d, C_l, and C_m. The latter effort improved on previous work in terms of accuracy using the same training and testing data sets. Both surrogate models realized fast, interactive aerodynamic optimization and have been integrated into Webfoil. However, these efforts only provided scalar aerodynamic quantities, and the flow conditions were restricted to discrete Reynolds numbers. Another shortcoming of these previous efforts is that the thickness constraints were not guaranteed to be satisfied because of the way the SVD was performed on the airfoil shapes. Finally, different numbers of SVD airfoil shape modes were used for the subsonic and transonic regimes, which prohibited cross-regime multipoint design optimizations.

To improve our previous efforts, we propose to extend the generative adversarial networks (GAN) technique to parameterize airfoils [28]. The GAN technique was originally developed as an intelligent generative framework to advance the generator models via an adversarial process where the generator and discriminator are trained simultaneously. The discriminator's training aims to differentiate the samples from an existing data set and the generated shapes by the generator. In contrast, the training of the generator aims to generate shapes that share similar patterns as the existing data set such that the discriminator cannot differentiate. This adversarial training setup facilitates the generator to produce shapes similar to the existing data set as much as possible. Information-based GAN [29] model introduced latent variables into the original GAN model to represent the structured data features (*e.g.*, airfoil thickness) by maximizing the mutual information between the latent variables and the observations. Latent variables in statistics mean hidden variables to be inferred through modeling (training of GAN model) of the measurable observations. Based on an existing data set of airfoil shapes, training should reveal thickness and camber as two of the latent variables that exert the most global control on the airfoil shape. The next section details the GAN-series model setup.

Bézier-based GAN (BézierGAN) extended the GAN-series models for airfoil parameterization by adding a Bézier curve as the last layer of the generator for smoother airfoil shapes [23,24]. The BézierGAN model has been shown to reduce the design space by filtering out unreasonable shapes, thus accelerating the design optimization process. Since the latent variables exert the most control on the generated airfoil shapes, the authors only considered the latent variables for surrogate modeling and airfoil design optimizations [23,24]. However, considering only latent variables negatively affects the parameterization fitting accuracy of airfoils and may prevent the optimization from reaching the actual optimal shapes.

The present paper builds on these previous efforts by developing an improved fast, interactive optimization framework with several new features. First, we develop a B-Spline-based generative adversarial network (BSplineGAN) model for intelligent parameterization. The BSplineGAN model is trained to generate reasonable airfoil shapes so that the original design space is automatically reduced while maintaining sufficient ranges in the design space. The generated airfoil shapes are expected to be smoother

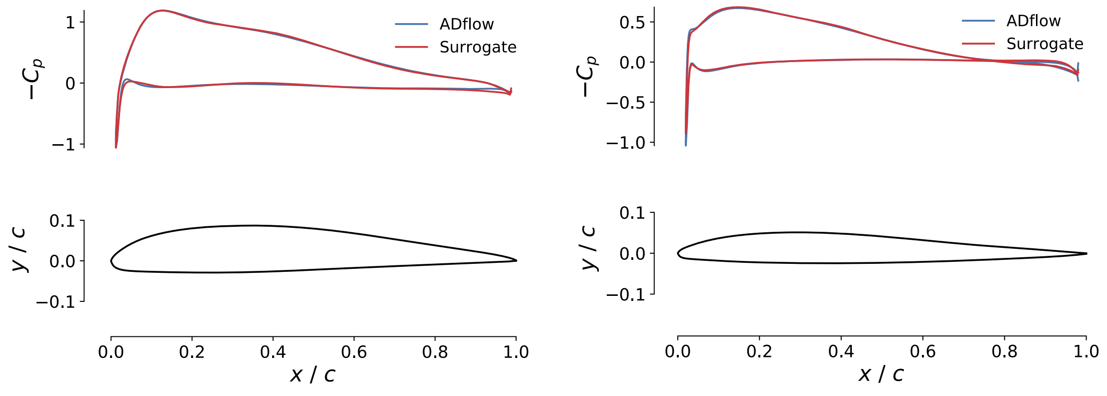
[1] http://webfoil.engin.umich.edu.



(a) Randomly generated airfoils by B-spline curves of 18-th order with 18 control points on each airfoil surface.

(b) Randomly generated airfoils by a BSplineGAN model. We use 18-th order B-spline curves (18 control points on each airfoil surface) as the last one layer of the generator networks.

Fig. 1. Comparison of randomly generated shapes between B-spline and BSplineGAN.



(a) Random airfoil 1

(b) Random airfoil 2

Fig. 2. C_p comparison between CFD results and surrogate predictions for random airfoils in testing data set.

and more practical using this approach (Fig. 1), and we use the same parameterization for both subsonic and transonic regimes. Also, the airfoil thicknesses are computed accurately and the corresponding gradients are computed via backpropagation within `Tensorflow` [30]. In addition, we consider a wider range of flow conditions—Mach number (M), Reynolds number (Re), and angle of attack (α). Typical aircraft flights exhibit a correlation between M and Re, so we exploit this correlation using Gaussian copula dependent sampling. In terms of surrogate modeling, we train MLP models to predict C_d and C_l whose gradients with respect to the geometric parameters (*i.e.*, BSplineGAN parameters) are computed via the backpropagation process. Moreover, we train recurrent neural networks (RNN) to predict the C_p distributions, which provide richer information to analyze and interpret the designs (Fig. 2) [31]. The proposed fast interactive framework performs an airfoil aerodynamic design within a few seconds and is verified against aerodynamic shape optimizations based on direct CFD evaluations.

The rest of the paper is organized as follows. We describe the key modules in the proposed framework in Section 2. Section 2.2 elaborates the GAN-series model setups. Section 2.3 introduces the Latin hypercube sampling (LHS) strategies for independent sampling and infill sampling and the Gaussian copula dependent sampling, followed by the open-source mesh generator and flow solver (Section 2.4). We also depict the surrogate modeling (Section 2.5) and verification (Section 2.6). The demonstration and verification of the surrogate-based design framework against the direct CFD-driven optimizations are in Section 3. The paper ends with the conclusions in Section 4.

2. Methodology

This section details the methods developed and implemented for this work. We first explain the general process of constructing the proposed surrogate-based design framework, followed by a detailed discussion of each key component.

2.1. Overview

The overall construction process of the proposed design framework is summarized in Fig. 3 using the extended design structure matrix (XDSM) standard [32]. The construction process consists of the following steps:

1) An airfoil database [26] (post-processed based on the UIUC database) is used to train the BSplineGAN model for intelligent parameterization and automatic design space reduction. And we assume the database contains sufficient variability of airfoil

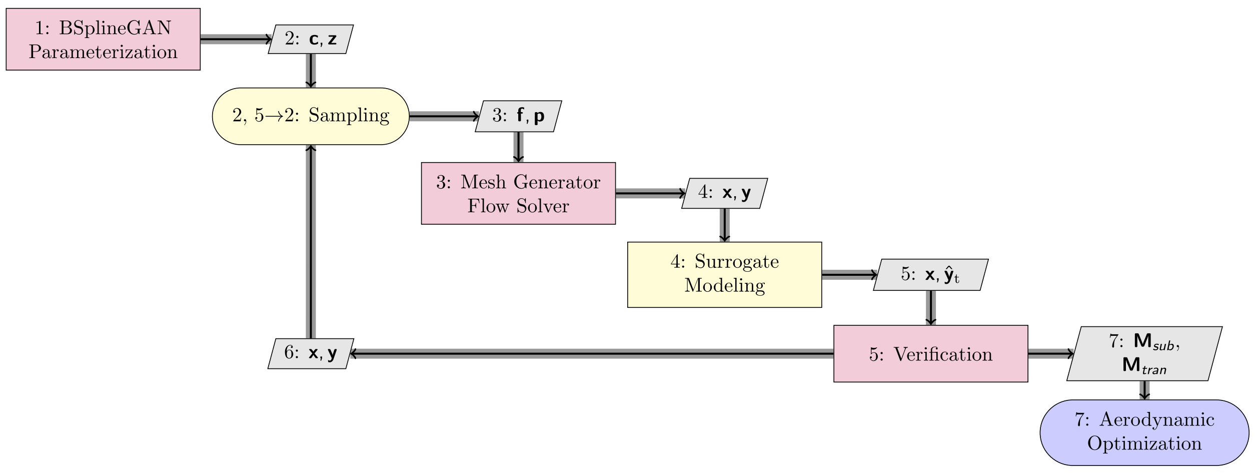


Fig. 3. Process and data dependencies for the proposed aerodynamic optimization framework illustrated using the XDSM [32]. The diagonal nodes are process components and the off-diagonal nodes are the data transferred between components. The thick gray lines represent the data flow, while the black lines represent the process flow.

shapes. In Fig. 3, $\mathbf{c}$ are the latent variables and $\mathbf{z}$ are the noise variables of BSplineGAN model. The latent variables target to expose inherent features in the data, such as airfoil thickness and exert the most global control over the airfoil shapes. In contrast, each of the noise variables adds slight changes to the airfoil thicknesses within a short range of chord-wise locations.

2) Once the BSplineGAN model is constructed, we sample the BSplineGAN variable space and the flight condition space. In Fig. 3, $\mathbf{f}$ are the generated airfoil coordinates and $\mathbf{p}$ are the sampled operating conditions.
3) The computational meshes are generated by pyHyp[2] [33] and evaluated under the sampled operating conditions using the ADflow[3] [34,35] CFD solver. The design variable vector, $\mathbf{x}$, includes the BSplineGAN variables and operating conditions. The model response vector, $\mathbf{y}$, includes C_d, C_l, and C_p.
4) We train neural network surrogates to predict the model response vector. Here, $\hat{\mathbf{y}}_t$ contains the surrogate predictions corresponding to the testing data points.
5) Infilled sampling is performed as needed based on the errors. The outputs of this process are the two sets of trained neural network models ($\mathbf{M}_{sub}$ and $\mathbf{M}_{tran}$) within the Mach number ranges of $M = [0.3, 0.6]$ and $M = (0.6, 0.7]$. Each set of trained models contains separate surrogates for C_d, C_l, and C_p.
6) The aerodynamic shape optimization is performed using the SNOPT gradient-based optimizer [36] through the pyOptSparse interface[4] [37]. The objective and constraint functions are predicted by the neural network surrogates. The BSplineGAN model evaluates the geometric constraints (airfoil thickness and area). Derivatives of the objective and constraint functions are computed by backpropagation.

2.2. *Airfoil parameterization using BSplineGAN model*

In this section, we introduce the original GAN model and its key variations that inspire the proposed BSplineGAN model. Then, we describe the detailed setup of the BSplineGAN intelligent airfoil parameterization method.

[2] https://github.com/mdolab/pyHyp.
[3] https://github.com/mdolab/adflow.
[4] https://github.com/mdolab/pyoptsparse.

2.2.1. *Generative adversarial networks*

GAN was originally developed by Goodfellow et al. [28] to generate shapes that share similar characteristics with existing data sets. The GAN modeling principle also applies to the airfoil parameterization, whose process is shown in Fig. 4 and listed as follows.

1) The generator takes the random variables ($\mathbf{z}$) following uniform distribution $\mathcal{U}(0, 1)$ as inputs and generates airfoil shapes ($\mathbf{g}$).
2) The discriminator takes $\mathbf{g}$ and an existing airfoil database ($\mathbf{e}$) as inputs and estimates the corresponding probabilities of being realistic. Here, $\mathbf{p}_\mathrm{g}$ and $\mathbf{p}_\mathrm{e}$ are the probabilities of generated airfoils and existing airfoils being realistic, respectively.
3) The training of the discriminator tunes the discriminator weights ($\mathbf{w}_d$) to distinguish between realistic and unrealistic airfoils. This training process drives $\mathbf{p}_\mathrm{g}$ and $\mathbf{p}_\mathrm{e}$ towards $\mathbf{0}$ and $\mathbf{1}$, respectively. The training of the generator tunes its weights ($\mathbf{w}_\mathrm{g}$) to generate shapes that are as realistic as possible. This training drives $\mathbf{p}_\mathrm{g}$ towards $\mathbf{1}$. Simultaneous training of the discriminator and generator results in the $\mathbf{p}_\mathrm{g}$ and $\mathbf{p}_\mathrm{e}$ values of $\mathbf{0.5}$ within the whole input space meaning that the discriminator cannot differentiate generated and existing shapes.

This type of model setup and training results in reasonable airfoil shapes within the whole GAN parametric space, representing the airfoil shape space well with sufficient variation. Thus, the random variables, $\mathbf{z}$, of a trained GAN model control the generated airfoil shapes and can serve as design variables for airfoil shape optimization.

This process is mathematically formulated as the minimax problem [28],

$$\min_{\mathbf{w}_\mathrm{g}} \max_{\mathbf{w}_d} V(\mathbf{w}_\mathrm{g}, \mathbf{w}_d) = \mathbb{E}_{\mathbf{z}\sim P_z}[\log(\mathbf{1} - \mathbf{p}_\mathrm{g})] + \mathbb{E}_{\mathbf{e}\sim P_\mathrm{data}}[\log(\mathbf{p}_\mathrm{e})], \tag{1}$$

where $\mathbf{e}$ is sampled from the existing data distribution P_data.

2.2.2. *B-spline parameterization*

A B-spline curve is commonly used to represent airfoils and has a mathematical expression of

$$B(u) = \sum_{i=0}^{n} N_{i,k}(u) p_i, \tag{2}$$

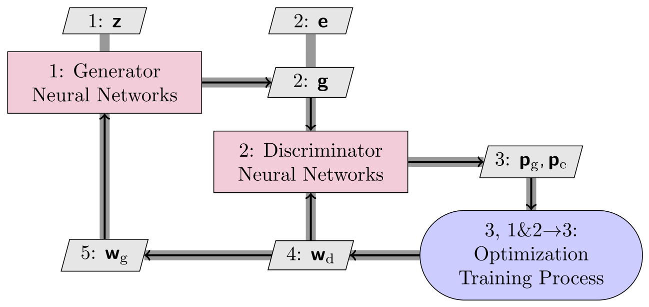

1: z
2: e
1: Generator Neural Networks
2: g
2: Discriminator Neural Networks
3: $\mathbf{p}_g, \mathbf{p}_e$
3, 1&2→3: Optimization Training Process
5: $\mathbf{w}_g$
4: $\mathbf{w}_d$

Fig. 4. Construction of GAN model consists of training the generator and the discriminator, which compete against each other.

where k is the order of B-spline curve, u is a knot within the range $[0, 1]$, $N_{i,k}$ are the basis functions, and p_i contains the (x, y) coordinates of the ith control point in the global coordinate system. We keep x coordinates of control points fixed and vary y coordinates to change airfoil shapes. The total number of control points is $n+1$. The basis functions are defined as

$$N_{i,1} = \begin{cases} 1 & u_i \le u \le u_{i+1}, \\ 0 & \text{otherwise}, \end{cases} \tag{3}$$

$$N_{i,k} = \frac{u - u_i}{u_{i+k-1} - u_i} N_{i,k-1}(u) + \frac{u_{i+k} - u}{u_{i+k} - u_{i+1}} N_{i+1,k-1}(u), \tag{4}$$

where we use the increasing knot vector of the length $n+1+k$, $[u_0, \dots, u_{n+k}]$ and $u_0 = 0, u_{n+k} = 1$. Each knot is generated using the cosine function

$$u_j = 1 - \cos\left(\frac{\pi}{1000} j\right), j = 0, 1, \dots, 500. \tag{5}$$

We construct two distinct B-splines: one for the upper airfoil surface and the other for the lower one. Each B-spline curve has two end control points fixed at the leading edge $p_{\mathrm{LE}} = (0, 0)$ and the trailing edge $p_{\mathrm{TE}} = (1, 0)$. The remaining control points for each surface are distributed using a half-cosine distribution between 0 and 1 along the chordwise direction, and they are only allowed to move in the vertical direction. The half-cosine distribution is given as

$$p_{i,x} = \frac{1}{2}\left[1 - \cos\left(\pi \frac{(i-1)}{n+1}\right)\right]. \tag{6}$$

The independence between the B-spline curve degree and the number of control points allows flexible and fine shape control [38]. In this work, we interpolate the generated curves at the following x coordinates to be consistent with the post-processed airfoil data set, such that [26]

$$x_i = 1 - \cos\left(\frac{\pi}{250} i\right), i = 0, 1, \dots, 125. \tag{7}$$

2.2.3. B-spline-based generative adversarial networks

When developing the BSplineGAN model (Fig. 5) for airfoil parameterization, we consider the following improvements for a better control over the generated shape. The generator (Step 1) takes not only the noise variables ($\mathbf{z}$) following $\mathcal{N}(0, 0.5^2)$ but also the latent variables ($\mathbf{c}$) following $\mathcal{U}(0, 1)$ as inputs (off-diagonal element 1) [29]. Figs. 6(a) to 6(c) show the most global control exerted by $\mathbf{c}$, while Fig. 6(d) shows the slight changes on the airfoil shapes added by $\mathbf{z}$. $\mathbf{c}$ represent the structured data set features, such as thickness and camber, leading to major control over the generated airfoil shapes by maximizing the mutual information between latent variables and existing data distribution. The mutual

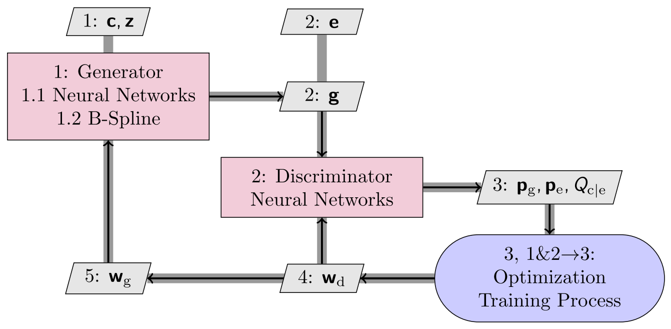

1: c, z
2: e
1: Generator
1.1 Neural Networks
1.2 B-Spline
2: g
2: Discriminator Neural Networks
3: $\mathbf{p}_g, \mathbf{p}_e, Q_{\mathrm{c|e}}$
3, 1&2→3: Optimization Training Process
5: $\mathbf{w}_g$
4: $\mathbf{w}_d$

Fig. 5. The BSplineGAN model captures salient structured semantic features by maximizing mutual information between existing data and latent variables ($\mathbf{c}$). The B-spline layer within the generator guarantees smooth airfoils.

information is used to quantify how much one random variable is related to another one with the formal definition

$$I(X_1, X_2) = \sum_{x_1 \in X_1} \sum_{x_2 \in X_2} P(x_1, x_2) \log \frac{P(x_1, x_2)}{P(x_1) P(x_2)}, \tag{8}$$

where $P(x_1)$ and $P(x_2)$ are the marginal distributions of the two random variables X_1 and X_2. The B-spline layer (Step 1.2) of the generator makes the generated airfoil shapes smoother, sharing a similar principle as the BézierGAN model [23,24]. We represent airfoil upper and lower surfaces using two separate B-spline curves and force the leading and trailing edges at the points of $(0, 0)$ and $(1, 0)$, respectively. The discriminator (Step 2) still reads $\mathbf{e}$ (off-diagonal element 0) and $\mathbf{g}$ (off-diagonal element 2), but produces one more output, $Q(\mathbf{c}|\mathbf{e})$ (off-diagonal element 3), an auxiliary distribution for approximating the real distribution, $P(\mathbf{c}|\mathbf{e})$.

To complete the setup of the optimization training process (Step 3), we formulate the lower bound of the mutual information (Eqn. (8)) with $Q(\mathbf{c}|\mathbf{e})$, which yields

$$L_1(\mathbf{w}_g, \mathbf{w}_d) = \mathbb{E}_{\mathbf{g} \sim P_{G(\mathbf{c},\mathbf{z})}} [\mathbb{E}_{\mathbf{c}' \sim P(\mathbf{c}|\mathbf{e})} [\log Q(\mathbf{c}'|\mathbf{e})]] + \text{const}, \tag{9}$$

where $P_{G(\mathbf{c},\mathbf{z})}$ is the distribution of generated airfoil shapes. Chen et al. [29] provided more details on this formulation. We maximize the lower bound of mutual information in addition to the original GAN minimax objective function with regularization terms on:
(a) adjacent B-spline control points to keep them close via the average Euclidean distance

$$R_1(G) = \frac{1}{Nn} \sum_{j=1}^{N} \sum_{i=1}^{n+1} \| p_i^{(j)} - p_{i-1}^{(j)} \|_2, \tag{10}$$

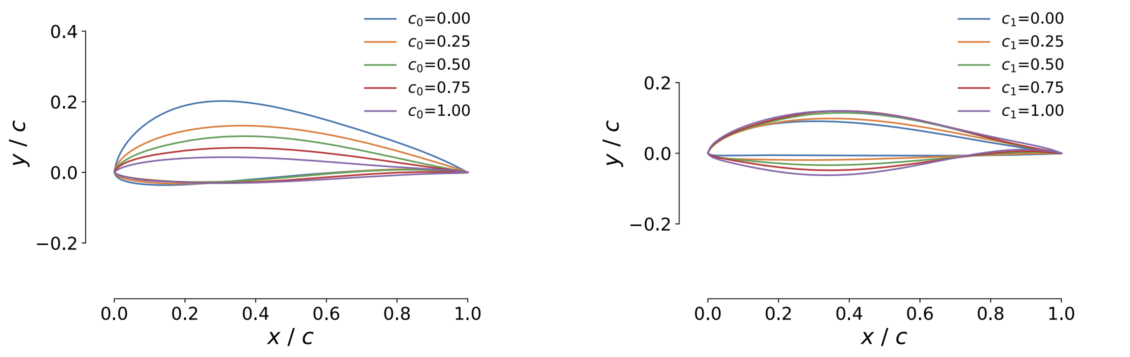
where N is the training batch size, n is the number of control points, p_i is the i^{th} control points.
(b) in addition to the original setup in Chen and Fuge [23], we consider the difference between upper and lower surface control points of the same x coordinates to avoid intersected airfoil shapes

$$R_2(G) = \frac{1}{Nn_s} \sum_{j=1}^{N} \sum_{i=1}^{n_s} \max(0, p_{l,i}^{(j)} - p_{u,i}^{(j)}), \tag{11}$$

where n_s is the number of control points on each surface.

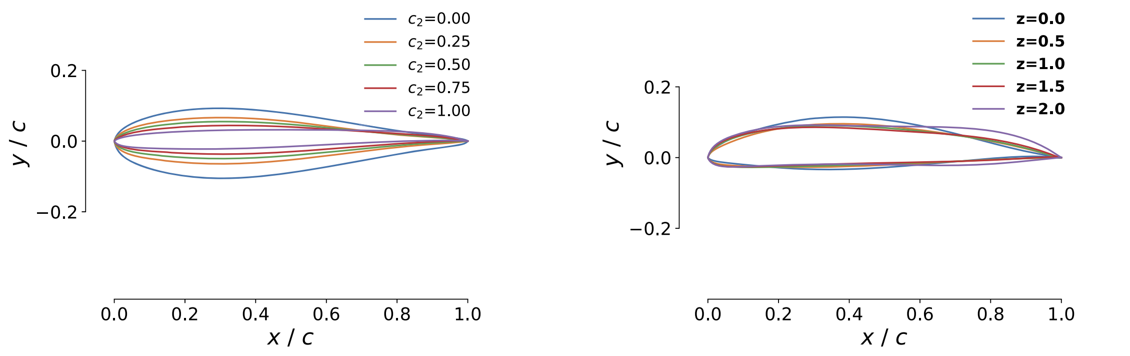
The mutual information lower bound and the regularization terms are added to the original GAN objective function (Eqn. (1)) with no change in the training process. Therefore, the objective function becomes

$$\min_{\mathbf{w}_g, Q} \max_{\mathbf{w}_d} V(\mathbf{w}_g, \mathbf{w}_d) - \lambda_0 L_1(\mathbf{w}_g, \mathbf{w}_d) + \sum_{i=1}^{2} \lambda_i R_i(\mathbf{w}_g). \tag{12}$$



(a) The latent variable c_0 mainly controls the airfoil camber through the model training process (c_1=c_2=0.75, $\mathbf{z}$ = $\mathbf{0}$)

(b) The latent variable c_1 mainly changes the airfoil leading-edge angles (c_0=c_2=0.5, $\mathbf{z}$ = $\mathbf{0}$)



(c) The latent variable c_2 captures the airfoil thickness variable (c_0=c_1=1.0, $\mathbf{z}$ = $\mathbf{0}$)

(d) Noise variables slightly change the airfoil shapes, but when all noise variables are greater than 1, the geometric effects are still obvious ($\mathbf{c}$ = $\mathbf{0.5}$)

Fig. 6. We train an example BSplineGAN model with 10 noise variables ($\mathbf{z}$) and 3 latent variables ($\mathbf{c}$) and investigate the effects on the generated airfoil shapes.

We set λ_i as 1 in this work. The construction and training process of the BSplineGAN are completed within Tensorflow [30].

2.3. Sampling strategies

In this work, we sample the independent variables using the Latin hypercube sampling (LHS) [39] and sample the dependent variables using the Gaussian copula [40]. If the verification accuracy is not sufficient, we use the augmented LHS for infill sampling.

2.3.1. Latin hypercube sampling

LHS [39] is an improved version of Monte Carlo sampling. Fig. 7(a) shows an example where four LHS points are generated in two-dimensional space. Instead of randomly sampling pre-set distributions, we first divide the space of $[0, 1]^2$ into evenly distributed four sub-intervals in each dimension, constructing a 4×4 matrix. We sweep through the columns (sub-intervals), searching for empty rows (sub-intervals), among which we randomly select one row to generate a value for every column. The generated values are used as cumulative-distribution-function (CDF) values of corresponding pre-assigned distributions, such as uniform or Gaussian distributions. The corresponding inverse CDF values are the sampled parameters.

Infill sampling using augmented LHS extends the small data set while keeping the characteristics of LHS (Fig. 7(b)). Thus, we can accumulate existing small data sets with newly generated ones to achieve a higher level of accuracy. We use the function augmentLHS within the R package lhs for infill sampling.

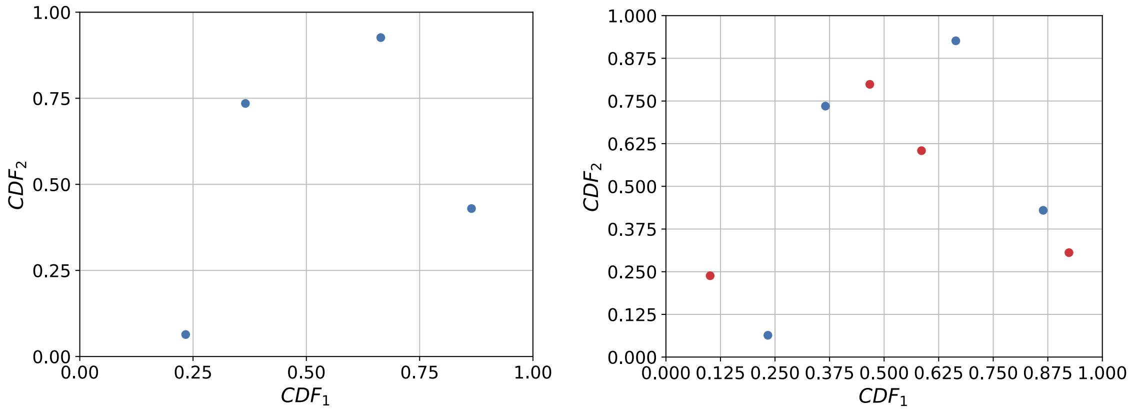
2.3.2. Copula sampling

A K-dimensional copula is a multivariate distribution over $[0, 1]^K$ with uniform marginals along each dimension [41]. A selected copula provides a correlation structure between random variables provided with marginal univariate distributions. The Gaussian copula is a widely used copula type that can handle both independent and dependent samplings, as shown in Fig. 8. We use Gaussian copula for dependent sampling to avoid unrealistic or impractical Mach–Reynolds pairs. The Gaussian copula is formulated as

$$GC(u_1, \ldots, u_K; \mathbf{R}) = \Phi_K\left(\Phi^{-1}(u_1), \ldots, \Phi^{-1}(u_K); \mathbf{R}\right), \tag{13}$$

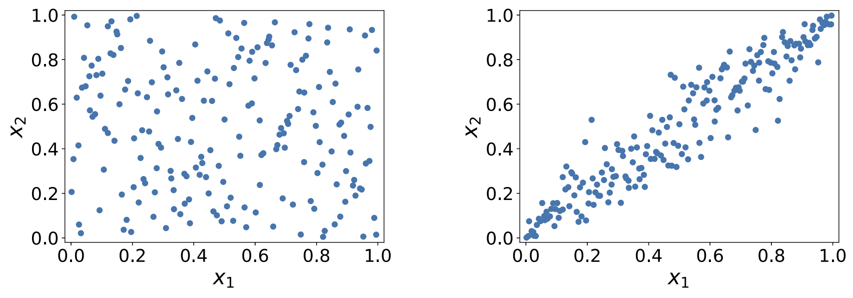
where $\mathbf{u}$ contains K elements, each of which belongs to a standard uniform space, $\mathcal{U}(0, 1)$; Φ_K is the K-variate Gaussian distribution with a mean vector $\mathbf{0}$ corresponding to the K random variables and correlation matrix $\mathbf{R}$; and Φ^{-1} is the inverse CDF of the standard Gaussian distribution.

Setting the target joint probability density function as a Gaussian copula, we achieve dependent sampling via an inverse Nataf transformation [41]. An inverse Nataf transformation maps standard uniform space into probability space identified by a pre-assigned CDF $\mathbf{F_X}$. Thus, we can sample from $\mathbf{F_X}$, as shown in Fig. 9. First, we start from K-dimensional uniform distributions, generating random independent uniform parameters $\mathbf{Z}$ using LHS. Then, the inverse standard Gaussian distribution produces random independent Gaussian parameters, $\mathbf{W}$, using $\mathbf{Z}$ as CDF values. We convert $\mathbf{W}$ to random dependent parameters $\mathbf{V}$ by $\mathbf{V} = \Gamma\mathbf{W}$, where Γ is Cholesky factor of $\mathbf{R}$, satisfying the matrix equation $\Gamma\Gamma^{\mathrm{T}} = \mathbf{R}$.



(a) Four initial LHS sample points (b) Four augmented LHS sample points added to the initial samples

Fig. 7. Augmented LHS adds four new LHS points (red) to initial LHS points (blue) by sweeping through columns of the $(4+4) \times (4+4)$ matrix and by generating a random value within an arbitrary empty row at each column sweep as normal LHS method. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)



(a) ρ_{x_1,x_2}=0, representing an independent sampling (b) ρ_{x_1,x_2}=0.95, making a highly dependent sampling

Fig. 8. Gaussian copula for random parameters, $(x_1, x_2) \sim \mathcal{U}(0,1)$, following different correlations (ρ_{x_1,x_2}).

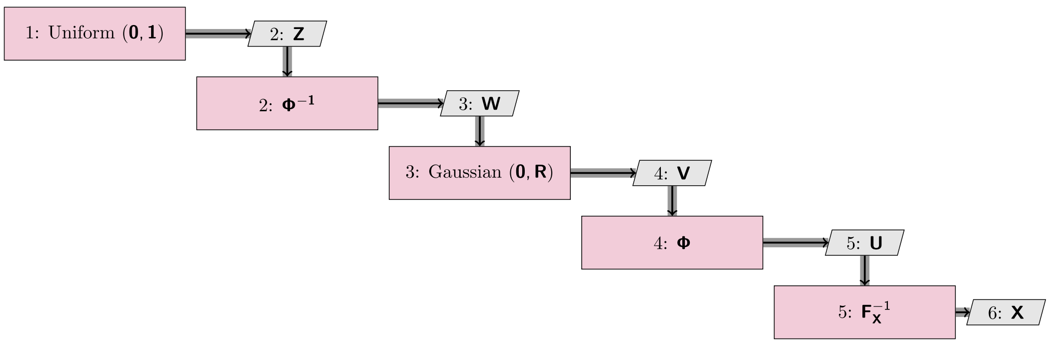


Fig. 9. Gaussian copula dependence sampling via inverse Nataf transformation.

We feed $\mathbf{V}$ into a standard Gaussian distribution to get the corresponding CDF values, $\mathbf{U}$. In the end, we calculate the sampled $\mathbf{X}$ using $\mathbf{U}$ as CDF values of the inverse of $\mathbf{F_X}$ ($\mathbf{F_X^{-1}}$). The samples generated in this way follow the pre-assigned distributions ($\mathbf{F_X}$) and maintain the pre-assumed correlations. `augmentLHS` can be added to the first step for infilling samples.

2.4. Mesh generation and flow simulation

We use pyHyp as the mesh generator and ADflow as the flow solver. pyHyp uses a hyperbolic volume mesh marching scheme to extrude structured surface meshes into volume meshes [33]. Hyperbolic mesh generation is formulated to achieve mesh orthogonality and cell volume specification. pyHyp enhances hyperbolic mesh generation for high-quality meshes by adding spatially-variable smoothing coefficient, metric correction procedures, local treatment of severe convex corners, and new extrapolation treatments of floating and axis boundaries [33].

ADflow is a finite-volume structured multiblock and overset mesh solver distributed under an open-source license [34]. ADflow includes a discrete adjoint to compute derivatives for gradient-based aerodynamic shape optimization [5,42] and features a Python API that provides a convenient interface for CFD analysis and design optimization [34]. The inviscid fluxes are discretized using three numerical schemes: the scalar Jameson–Schmidt–Turkel artificial dissipation scheme, a matrix dissipation scheme, and a monotone upstream-centered scheme for conservation laws. The viscous flux gradients are computed using a Green–Gauss approach. ADflow solves the compressible flow equations; the mean equations for the conservation of mass, momentum, and energy are solved simultaneously. Several turbulence models, including the one-equation Spalart–Allmaras model and the two-

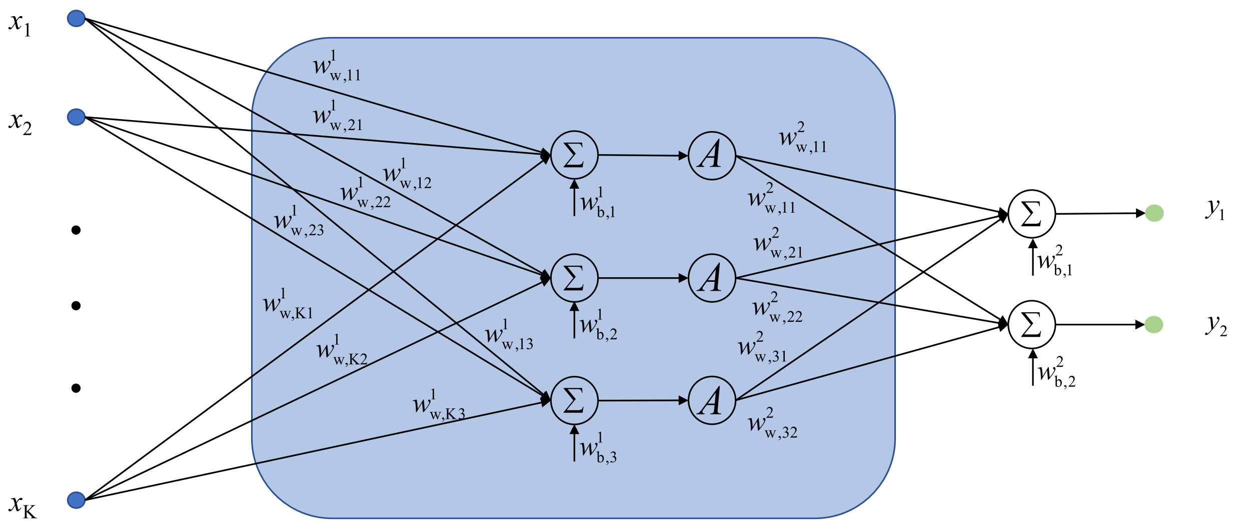


Fig. 10. A two-layer MLP takes inputs ($\mathbf{x}$), multiplies them by weighting parameters ($\mathbf{w}_w$) and adds bias parameters ($\mathbf{w}_b$). Then, the output of the operation Σ is sent to the activation function A. Superscripts represent the layer index, and subscripts represents the connection between specific neurons.

equation shear stress transport model, are implemented. ADflow can solve the turbulence models either as a segregated system or as a fully-coupled system with the mean flow equations. In terms of converging the residual equations, ADflow implements the approximate Newton–Krylov (ANK) algorithm, which is suitable for both multiblock and overset meshes and is sufficiently robust as a globalization scheme for the full Newton–Krylov (NK) solver [35]. The full NK solver has fast terminal convergence within the Newton basin of attraction. We start with ANK until a relative l^2 convergence residual of 10^{-10} followed by the NK to reach 10^{-12}.

2.5. Neural network surrogates with mixture of experts

Neural network surrogates effectively capture nonlinear characteristics and patterns in target data sets and scale well with the number of input dimensions. Batch optimization enables neural networks to deal with large data sets effectively. We construct neural network surrogates to predict C_d, C_l, and C_p. As mentioned above, we use MLP for C_d and C_l predictions, and RNN for C_p predictions. We construct different surrogates for subsonic and transonic regimes and make predictions under the guidance of a mixture of experts (MOE) approach [14]. Based on our investigation and previous experience, several separate surrogates guided by MOE outperform a single global one.

2.5.1. Multilayer perceptron

MLP is a type of feed-forward network and refers to multiple layers of perceptrons added with activation functions (Fig. 10). We extend this type of construction to a multiple-layer architecture and with the following steps.

1) Preprocess the input parameters with `MinMaxScaler` within the `SKLearn` toolbox [43] to normalize the input parameters to be within the range of $[0, 1]$ based on the corresponding minimum and maximum values.
2) Construct MLP networks, each layer of which ends with an activation function, such as scaled exponential linear unit (`selu`) and leaky rectified linear unit (`leaky_relu`) within `Tensorflow` [30].
3) Set the cost function as the root mean squared error (RMSE) between model observations and surrogate predictions of the training data.
4) Train the MLP surrogates using the Adam optimizer [44] via batch optimization strategy.
5) Monitor the RMSE of training and validation data sets for the convergence of MLP model training.
6) Investigate the predictive performance in the testing data set and decide whether to enlarge the training data set.

2.5.2. Recurrent neural networks

In addition to predicting scalar quantities, we also predict vector model response, *i.e.*, C_p distribution, using RNN models [31]. The RNN model's output is a function of the input parameters and the outputs of the previous time step. This characteristic gives the RNN memory of the previous outputs, which is not available in MLP. However, the RNN model suffers from extremely large or small gradients due to the accumulated process of long-time sequences. Long short-term memory (LSTM) is designed to avoid the long-term dependency issue by adding forget, update, and output layers (Fig. 11) [45] to the original RNN model.

The forget layer is controlled by a `sigmoid` activation function, which takes the input parameters ($\mathbf{x}$) and outputs from previous time steps ($\mathbf{y}_0$), and generates numbers ($\mathbf{g}_f$) between 0 and 1 representing how much information in state variables ($\mathbf{C}_0$) is kept. Here, 0 represents "forget" all the information while 1 represents "remember" all the information in $\mathbf{C}_0$.

The update layer consists of a `sigmoid` and a `tanh` activation function. The `sigmoid` layer outputs the update gate ($\mathbf{g}_u$) and decides which information we need to update, while the `tanh` layer creates candidate state variables ($\mathbf{C}'$). The Hadamard product between $\mathbf{g}_u$ and $\mathbf{C}'$ decides what to add to the filtered $\mathbf{C}_0$.

The output layer includes an output gate of `sigmoid` activation function and a `tanh` activation function which scales the filtered $\mathbf{C}_0$ to the range of $[-1, 1]$. The Hadamard product between $\mathbf{g}_o$ and scaled $\mathbf{C}_0$ works as the output layer of the LSTM cell.

The key variable calculations in Fig. 11 are summarized as follows:

$$\begin{aligned}
\mathbf{g}_f &= \text{sigmoid}\left(\mathbf{w}_f \cdot [\mathbf{y}_0, \mathbf{x}] + \mathbf{b}_f\right), \\
\mathbf{g}_u &= \text{sigmoid}\left(\mathbf{w}_u \cdot [\mathbf{y}_0, \mathbf{x}] + \mathbf{b}_u\right), \\
\mathbf{C}' &= \tanh\left(\mathbf{w}_C \cdot [\mathbf{y}_0, \mathbf{x}] + \mathbf{b}_C\right), \\
\mathbf{g}_o &= \text{sigmoid}\left(\mathbf{w}_o \cdot [\mathbf{y}_0, \mathbf{x}] + \mathbf{b}_o\right),
\end{aligned} \tag{14}$$

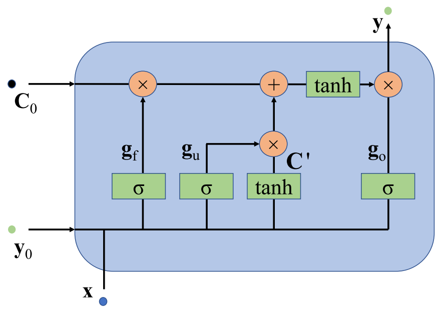


Fig. 11. An LSTM cell consists of the forget gate variable ($\mathbf{g}_f$), update gate variable ($\mathbf{g}_u$), and output gate variable ($\mathbf{g}_o$).

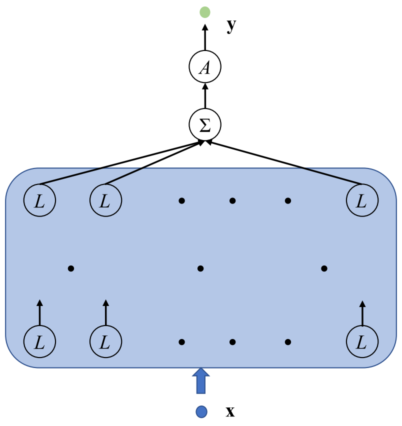


Fig. 12. We stack LSTM cells (L) as MLSTM, followed by a MLP layer to match model observations. Each LSTM cell takes the output of the corresponding previous layer cell as its inputs.

where $\mathbf{w}$ contains the weighting parameters and the subscripts represent the corresponding variables. The initialized output and state variables are $\mathbf{y}_0$ and $\mathbf{C}_0$, respectively.

Our testing shows that LSTM works not only for time-dependent problems but also for steady regression problems. The geometric parameters and flight conditions in this work are concatenated as the single-time-step inputs in the LSTM cell. The C_p distribution is the quantity of interest to be predicted of this single-time-step LSTM cell. We stack multi-layer LSTM (MLSTM) surrogates to predict the C_p distributions (Fig. 12) since deep architectures often have an advantage over shallow architectures at dealing with complex problems [46]. Specifically, a hierarchical RNN model can be exponentially more efficient at representing complex functions than a shallow one [47]. Each layer of the model solves a part of the task before passing it on to the next until the last layer provides the final output. In this work, the input parameters are combined as a pack and fed to each LSTM cell in the same layer. Then, each layer's outputs are packed again and fed to a higher hidden layer, and so on. The LSTM cells and MLSTM are constructed within `TensorFlow` [30]. This is the first time that MLSTM gets introduced to static aerodynamic problems.

2.5.3. *Mixture of experts*

The drag coefficient is highly nonlinear as the Mach number approaches 1, so we construct separate surrogate models for subsonic and transonic regimes. Mathematically, we formulate the prediction as

$$\mathbf{M} = l_1 \cdot \mathbf{M}_1(\mathbf{x}) + l_2 \cdot \mathbf{M}_2(\mathbf{x}), \tag{15}$$

where $\mathbf{M}_1$ and $\mathbf{M}_2$ are the subsonic and transonic neural network surrogates, respectively, and the gate parameters l_1 and l_2 are computed as

$$\begin{cases} l_1 = 1, \quad l_2 = 0, \quad \text{if} \quad M \in [0.3, 0.6], \\ l_1 = 0, \quad l_2 = 1, \quad \text{if} \quad M \in (0.6, 0.7]. \end{cases} \tag{16}$$

2.5.4. *Adam optimizer*

Adam is an efficient stochastic optimization algorithm that requires only first-order gradients with little memory requirement [44]. This algorithm computes individual adaptive learning rates for different parameters by estimating the first and second moments of the gradients. Adam can be seen as a combination of gradient descent with momentum [48] and root-mean-square propagation (RMSP) [49] algorithms.

The moment algorithm accelerates the gradient descent algorithm considering the exponentially weighted average of the gradients [48]. The updates are given by

$$w_{t+1} = w_t - \alpha m_t, \tag{17}$$

where

$$m_t = \beta m_{t-1} + (1-\beta)\left[\frac{\partial L}{\partial w_t}\right]. \tag{18}$$

The subscripts t, $t-1$, and $t+1$ are the current, previous, and next optimization step, respectively, w are weights to be determined, α is learning rate, $\frac{\partial L}{\partial w_t}$ is the derivative of loss function with respect to weights at current optimization step, and β is a constant moving average parameter.

RMSP [49] is an adaptive learning algorithm by taking the exponential moving average which is formulated as

$$w_{t+1} = w_t - \frac{\alpha}{(v_t + \epsilon)^{1/2}} * \left[\frac{\partial L}{\partial w_t}\right]^2, \tag{19}$$

where

$$v_t = \beta v_{t-1} + (1-\beta)\left[\frac{\partial L}{\partial w_t}\right], \tag{20}$$

and we use a small positive constant ($\epsilon = 10^{-8}$). The other variables are the same as in Eqn. (18).

As stated above, the Adam optimizer combines moment and RMSP algorithms through the formulation of

$$w_{t+1} = w_t - \hat{m}_t\left(\frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}}\right), \tag{21}$$

where

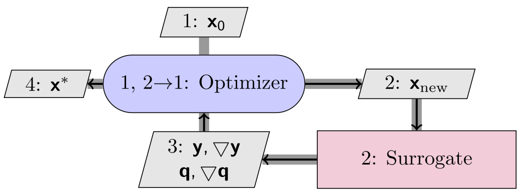
$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}, \end{aligned} \tag{22}$$

$\hat{m}_t$ and $\hat{v}_t$ follow the aforementioned updating process. In this work, we use the Adam optimizer within `Tensorflow` for all neural network training. We provide training details in Section 3.

2.6. *Verification metrics*

We use RMSE, relative l^2 error (ϵ) for C_d and C_l, mean relative l^1 error ($\bar{\epsilon}$) for C_p as the verification metrics for the neural network surrogates in this work. We also use the absolute error to analyze the response and surrogate prediction and corresponding statistics for deeper insights. The RMSE, ϵ, and $\bar{\epsilon}$ are defined as

$$\text{RMSE} = \sqrt{\frac{1}{N_{\text{test}}}\sum_{i=1}^{N_{\text{test}}}\left(S_{i,\text{pred}} - S_{i,\text{test}}\right)^2}, \tag{23}$$



1: $\mathbf{x}_0$
4: $\mathbf{x}^*$
1, 2→1: Optimizer
2: $\mathbf{x}_{\text{new}}$
3: $\mathbf{y}, \triangledown\mathbf{y}$ $\mathbf{q}, \triangledown\mathbf{q}$
2: Surrogate

Fig. 13. The surrogate-based optimization simplifies and accelerates airfoil optimization by using a surrogate model for the objective and constraint functions.

$$\epsilon = \frac{||\mathbf{S}_{\text{pred}} - \mathbf{S}_{\text{test}}||_2}{||\mathbf{S}_{\text{test}}||_2}, \tag{24}$$

$$\bar{\epsilon} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{|\mathbf{V}_{i,\text{pred}} - \mathbf{V}_{i,\text{test}}|}{|\mathbf{V}_{i,\text{test}}|}, \tag{25}$$

where N_{test} is the number of testing data points, $S_{i,\text{pred}}$ and $S_{i,\text{test}}$ are the scalar surrogate prediction and real observation of testing data, respectively. The $\mathbf{V}_{i,\text{pred}}$ and $\mathbf{V}_{i,\text{test}}$ vectors contain the surrogate prediction and real observation of testing data, respectively.

2.7. Airfoil aerodynamic optimization

Once surrogate models are trained and verified to be accurate enough, we perform the surrogate-based airfoil aerodynamic optimization process shown in Fig. 13. The SNOPT optimizer drives the optimization through the pyOptSparse interface. Once the optimizer provides updated designs within optimization steps, the surrogates directly provide objective functions, constraints, and corresponding gradients and send them back to the optimizer for the next iteration until the convergence criteria is fulfilled.

We verify the surrogate-based design framework by comparing results with CFD-based airfoil shape optimizations using the MACH-Aero framework,[5] which integrated the following key modules (see Fig. 14). First, we construct a free-form deformation box for morphing the airfoil shapes and generate a volume mesh of baseline geometry ($\mathbf{x}_0$) using pyHyp in Step 1 (Preprocessing). Then, the optimizer (again SNOPT through pyOptSparse) updates the design variables to $\mathbf{x}_{\text{new}}$ in Step 2. The geometry parameterization module (pyGeo[6]) performs the geometry deformation (Step 3) and computes the values of the geometric constraints and the corresponding gradients. The pyGeo framework applies changes to the geometry using a free-form deformation volume approach [50], which also changes the surface mesh and computes analytic derivatives for gradient-based optimization [50]. Based on the updated design surface ($\mathbf{f}$), the volume mesh deformation module (IDWarp[7]) generates a new volume mesh ($\mathbf{m}$) for Step 4. In the end, the flow and adjoint solvers (ADflow) run simulations for the deformed mesh and pass the quantities of interest ($\mathbf{y}$), state variables ($\mathbf{q}$), and gradients ($\triangledown\mathbf{y}$ and $\triangledown\mathbf{q}$) back to the optimizer (Step 2). This iterative loop continues until the optimal design ($\mathbf{x}^*$) is obtained.

3. Results and discussion

This section details the neural network model setups and demonstrates the proposed airfoil design framework under subsonic and transonic flight conditions. Parametric studies guide the key variable setup (geometric variables and flow conditions), and mesh convergence studies guarantee the CFD model accuracy.

[5] https://github.com/mdolab/MACH-Aero.
[6] https://github.com/mdolab/pygeo.
[7] https://github.com/mdolab/idwarp.

We also elaborate on the architecture of neural network surrogates, followed by the optimization verification against direct CFD simulation-based optimizations.

3.1. Key variables

We use the simplified architecture (Table 1) first proposed by Chen et al. [29], who introduced the latent variables to represent structured features of typical image data sets. Table 1 shows the BSplineGAN setup consisting of a discriminator network model and a generator network model with a B-spline layer on top. An existing data set containing 1,552 airfoil shapes [26] is used as the training data of BSplineGAN. We improve the idea proposed in Chen and Fuge [23], who extended the work of Chen et al. [29] by implementing Bézier curves as the last layer of the generator for smooth airfoil shapes. In particular, we set the last layer of the generator using two separate B-spline curves, which better controls the airfoil shapes. We also determine the number of latent variables by fitting accuracy to the existing airfoil data set according to a parametric study (Fig. 15). In the end, all noise and latent variables are considered for surrogate modeling and design optimizations to avoid losses of accuracy or flexibility.

Each B-spline curve is of 18-th order; it has 16 control points and two end control points fixed at the leading and trailing edges. The $\bar{\epsilon}$ fitting error (Eqn. (25)) to the airfoil database using such B-spline curves is below 0.3% of existing airfoil coordinates. We complete the training of BSplineGAN models using the Adam optimizer within `Tensorflow` with the momentum terms $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The generator and discriminator networks' learning rates are both set to 10^{-4}. The batch size is 32, and the total number of training steps is 13,200. A BSplineGAN model uses 16 latent variables and 10 noise variables is selected because it achieves a $\bar{\epsilon}$ within 1% of existing airfoils thicknesses (Fig. 15). Thus, all the BSplineGAN variables are considered into the input space of surrogate models and used as geometric parameters of airfoil shape designs.

The advantages that BSplineGAN has over the BézierGAN are due to the fact that B-splines are a generalization of Bézier curves [38]. First, using B-spline curves allows the curve order to be independent with the number of control points. Second, B-spline curves have finer shape control because of the strong convex hull property [38]. Third, advanced techniques, such as changing knots, can be flexibly implemented into B-spline parameterization for the convenience of editing and designing shapes. Piegl and Tiller [38] had detailed demonstrations for the readers who have interests. Besides, we use two distinct B-spline curves, complete fitting-accuracy studies, and consider all BSplineGAN variables for modeling and optimization to improve the prior BézierGAN work from the configuration perspective.

We investigate typical Mach and Reynolds number pairs, as shown in Table 2. The Mach numbers correspond to the component perpendicular to the wing leading edge and are given by

$$M = M_\infty \cos\Lambda, \tag{26}$$

where M_∞ is the freestream Mach number, and Λ is the leading-edge sweep angle. Our tests indicate that the Gaussian copula with a correlation of 0.85 adequately covers the typical pairs (Fig. 16).

We summarize the input variables for the neural network surrogates in Table 3. We assign $\mathcal{U}(0,1)$ to the 16 latent variables and $\mathcal{N}(0, 0.5^2)$ to the 10 noise variables. The BSplineGAN noise variables follow a normal distribution when constructing the BSplineGAN, but we sample noise variables as $\mathcal{U}(-2, 2)$ to cover the design space sufficiently. We set M to have a distribution $\mathcal{U}(0.3, 0.7)$ and α to $\mathcal{U}(0,3)$ deg. Here, we sample $\log(Re)$ uniformly instead of Re to avoid ignoring the low Reynolds number range.

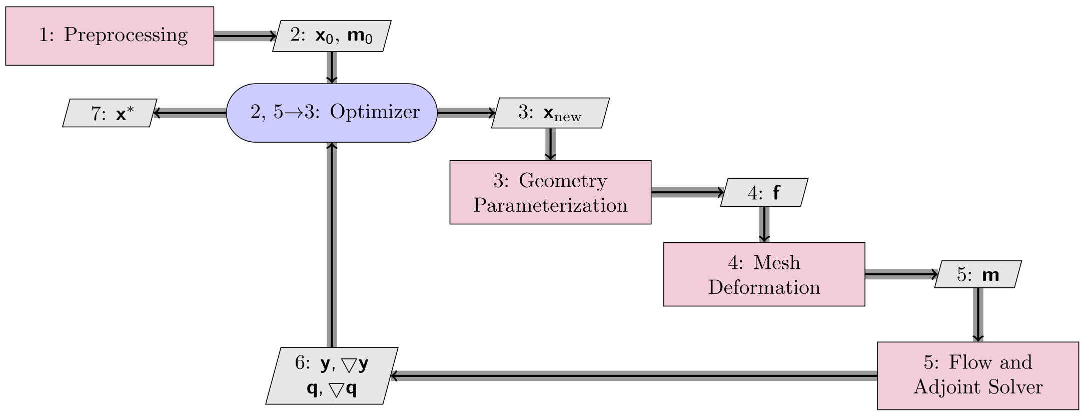

1: Preprocessing
2: $\mathbf{x}_0$, $\mathbf{m}_0$
7: $\mathbf{x}^*$
2, 5→3: Optimizer
3: $\mathbf{x}_{\text{new}}$
3: Geometry Parameterization
4: $\mathbf{f}$
4: Mesh Deformation
5: $\mathbf{m}$
5: Flow and Adjoint Solver
6: $\mathbf{y}$, $\nabla\mathbf{y}$ $\mathbf{q}$, $\nabla\mathbf{q}$

Fig. 14. MACH-Aero performs aerodynamic optimization using direct CFD simulations to compute the objective and constraint functions.

Table 1
The BSplineGAN model architecture includes a generator and a discriminator. The layers can be fully connected (FC), convolutional (Conv), or deconvolutional (Deconv). Kernel size of all Deconv layers are [4, 3]. All strides for Conv and Deconv are [2, 1]. Selected activation functions include rectified linear unit (`relu`) and hyperbolic tangent (`tanh`) functions. All dropouts have a probability of 0.9 to keep current element.

Layers	Generator	Discriminator
L1	FC (1024 neurons), `relu`, batch norm	Conv (64 filters, kernel size [4, 2]), `relu`, batch norm
L2	FC ($4 \times 3 \times 256$ neurons), `relu`, batch norm	Conv (128 filters, kernel size [4, 2]), `relu`, batch norm
L3	Reshape to a [1, 4, 3, 256] tensor	Conv (256 filters, kernel size [4, 2]), `relu`, batch norm
L4	Deconv (128 filters), `relu`, batch norm	FC (16384 neurons), `relu`, batch norm
L5	Deconv (64 filters), `relu`, batch norm	FC (1024 neurons), no activation, no norm
L6	Deconv (32 filters), `relu`, batch norm	
L7	Conv (1 filter, kernel size [1, 2]), `tanh`, no norm	
L8	B-spline layer	

Table 2
Flight condition (M, Re) pairs for a wide range of aircraft.

Aircraft	M	Re	Aircraft	M	Re
Boeing 747	0.67	2×10^9	Cirrus SR22	0.285	4.7×10^6
Airbus A350	0.72	9×10^6	Model airplane	0.105	2.5×10^5
Airbus A320	0.71	2.7×10^6	Jonker JS-3 Rapture	0.087	1.6×10^6
Embraer E190	0.74	2.8×10^6	Paper airplane	0.001	4.7×10^4
Cessna 425	0.426	6.3×10^6			

Table 3
Distributions for the input parameters.

16 latent variables	10 noise variables	M	$\log(Re)$	α (deg)
$\mathcal{U}(0, 1)$	$\mathcal{N}(0, 0.5^2)$	$\mathcal{U}(0, 0.9)$	$\mathcal{U}(4, 10)$	$\mathcal{U}(0, 3)$

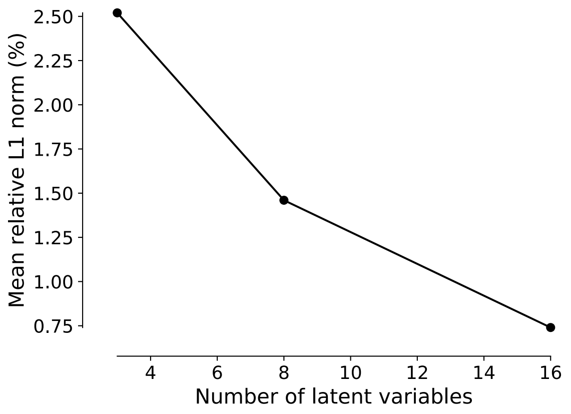


Fig. 15. Parametric study with respect to BSplineGAN latent variables. We select 16 latent variables, which achieves a $\bar{\epsilon}$ within 1%.

3.2. Mesh convergence study

We set up a general set of mesh for all data and complete the two mesh convergence study cases (subsonic and transonic) shown in Tables 4 and 5 and Fig. 17. We start with the finest mesh (M0), coarsen the mesh by half in both the streamwise and normal-wall directions to get M1, and then croasen again to get M2. The Richardson extrapolation values are 83.04 and 98.63 drag counts for the subsonic and transonic test cases, respectively. The difference in C_d between the M1 mesh and the Richardson extrapolation value is 0.4 drag counts for the subsonic case, achieving a mesh convergence order of 1.554. The transonic case's corresponding difference is around 1.3 drag counts, achieving a mesh convergence order of 1.235. Given this accuracy, we select M1 for our studies.

3.3. Surrogate model constructions and verification

We construct separate surrogate models for the Mach number ranges $M = [0.3, 0.6]$ and $M = (0.6, 0.7]$. The input parameters

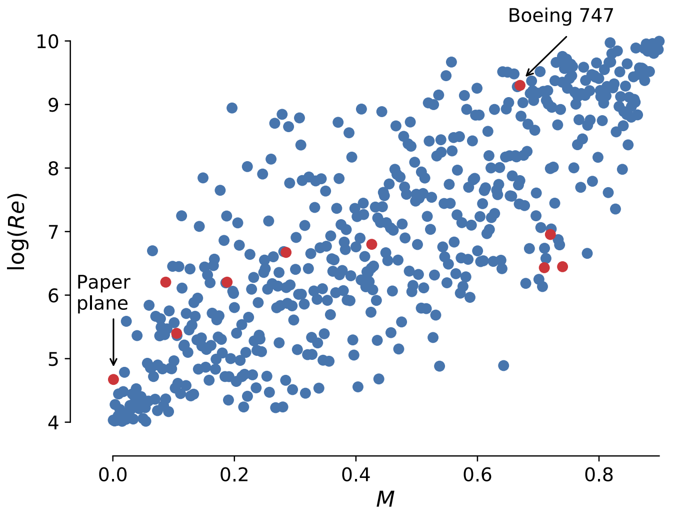


Fig. 16. Gaussian copula-generated samples (blue) with a pre-assigned correlation of 0.85 covers typical (M, Re) pairs (red) adequately.

Table 4
Subsonic case, $M = 0.45$, $Re = 6.0 \times 10^6$, target $C_l = 0.20$.

	Mesh size	α	C_l	C_d (10^{-4})
M0	687,616	1.6003	0.2000	83.391
M1	171,904	1.5949	0.2000	84.071
M2	42,976	1.5901	0.2000	86.067

Table 5
Transonic case, $M = 0.70$, $Re = 6.5 \times 10^6$, target $C_l = 0.72$.

	Mesh size	α	C_l	C_d (10^{-4})
M0	687,616	2.6109	0.7200	99.187
M1	171,904	2.5750	0.7200	99.942
M2	42,976	2.5266	0.7200	101.719

(BSplineGAN variables and flight conditions) remain the same for all surrogates, so we could perform cross-regime multipoint optimizations in future work. In this section, we detail the setup and verification.

3.3.1. *Subsonic surrogate model*

We run 45,696 training points, 330 validation points, and 331 testing points within the subsonic regime (M within $[0.3, 0.6]$). We construct MLP surrogates for C_d and C_l, and a MLSTM surrogate for C_p. The architectures for these surrogates are listed in Table 6. All the models are trained using the Adam optimizer within `Tensorflow` with the momentum terms $\beta_1 = 0.99$ and $\beta_2 = 0.999$, and the learning rate of 0.01. The C_d, C_l, and C_p surrogates have the batch sizes of 3,000, 2,500, and 2,500, respectively, and the total numbers of training steps of 37,980, 11,430, and 26,964, respectively.

The neural networks achieve the best predictive performance when the C_d surrogate has eight layers of neurons, the C_l surrogate has six layers, and the C_p surrogate has six layers. A decreasing number of neurons on each layer ([200, 180, 160, 160, 160, 140, 140, 140]) achieves the best C_d predictive accuracy, while an increasing number of neurons on each layer ([120, 140, 160, 180, 200, 200]) achieves the best C_l predictive accuracy. A decreasing number of LSTM cells ([180, 160, 140, 120, 80, 40]) has the best predictive accuracy and smoothest C_p distributions.

We verify the trained neural network surrogates using RMSE, ϵ, and $\bar{\epsilon}$, as shown in Table 7. Fig. 18 provides a closer look into the predictions versus model observations on the testing data set. One count is 10^{-4} for C_d and 10^{-3} for C_l. The RMSEs of the MLP surrogates of C_d and C_l are both around 5% of standard deviation for the testing data set (σ_{test}), and the ϵ is around 2.3%. The MLSTM surrogate decreases the RMSE of C_p to 0.0372, which corresponds to around 8% σ_{test}, and $\bar{\epsilon}$ is 6.13%. In general, the accuracy of C_p prediction is worse than that of C_d and C_l. Accurate surrogate prediction for C_p distributions is challenging near the trailing edge because of the trailing-edge-pressure-spike issue commonly found in CFD simulations (Fig. 2(b)). These metrics represent a satisfactory level of predictive accuracy.

We also compute the absolute error and check the corresponding key statistics for more insights (Fig. 19). The mean absolute error of the C_d and C_l predictions are 1.41 counts and 9.61 counts, respectively. Most of the absolute errors in the C_d prediction (85%) are within two counts, while 85% of the errors in C_l are within 18.4 counts.

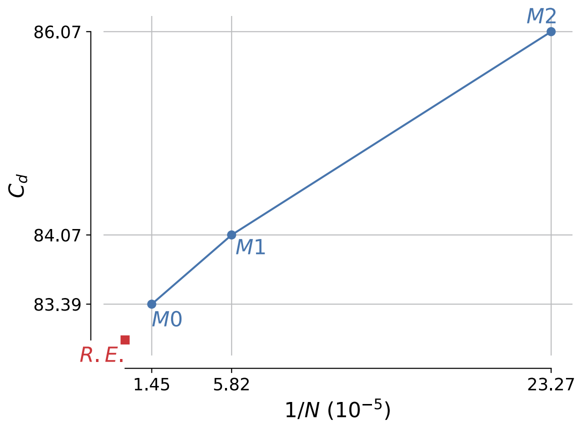
We show sample C_p distribution comparisons between the CFD results and surrogate predictions in Fig. 2. The airfoil shapes are randomly selected from the testing data set. The MLSTM surrogate exhibits good predictive performance compared with the CFD results.

3.3.2. *Transonic surrogate model*

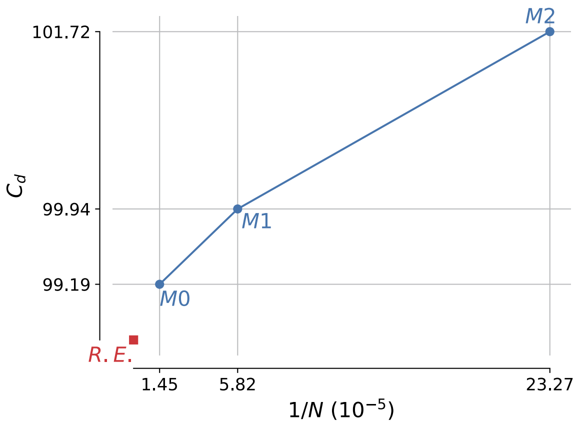
The transonic surrogate modeling uses 39,505 training points, 94 validation points, and 99 testing points for Mach numbers in the range of $M = (0.6, 0.7]$. The training settings for all the transonic surrogates are the same as the subsonic ones, namely, the Adam optimizer within `Tensorflow` with the momentum terms $\beta_1 = 0.99$ and $\beta_2 = 0.999$ and a learning rate of 0.01. The C_d, C_l, and C_p surrogates have the batch sizes of 1,800, 2,800, and 3,800, respectively, and the total numbers of training steps of 19,646, 5,910, and 6,556, respectively. The transonic surrogates tend to reach the training convergence earlier than the subsonic ones. We use six hidden layers with various activation functions for all neural network surrogates (see Table 8), while we use eight layers for the C_d surrogate in the subsonic regime. Testing shows that a decreasing number of neurons on each layer, *i.e.*, (180, 160, 140, 120, 100, 80), has the best predictive performance for C_d, and (180, 160, 140, 120, 80, 40) works the best for C_p, while an increasing number of neurons, *i.e.*, (100, 120, 140, 160, 180, 200), performs the best for C_l. This setup of neuron numbers on each layer follows a similar trend as the subsonic surrogate modeling. Specifically, the architecture of the transonic C_p surrogate is the same as the subsonic one. In contrast, the transonic C_d and C_l surrogates have around 20 fewer neurons on each layer than the subsonic ones. In addition, when selecting the types of activation functions, we follow a setup similar to the subsonic surrogate modeling and investigate the verification metrics for predictive performance.

Table 9 shows that the surrogate RMSEs for C_d, C_l and C_p are 8.76 counts, 19.67 counts and 0.084, respectively. The RMSEs for C_d and C_l are both around 7% of σ_{testing}, and the RMSE for C_p is around 16% σ_{testing}. The relative errors of C_d, C_l and C_p are 4.65%, 2.87%, and 11.72%, respectively. Comparing the verification metrics with Table 7 reveals that the surrogates have better predictive accuracy in the subsonic regime, which agrees with the previous experience that the aerodynamic coefficients are more nonlinear for Mach numbers above about 0.6 [14]. The C_p prediction is not as accurate as C_d and C_l due to the trailing-edge-pressure-spike issue in CFD mentioned in the subsonic cases. In addition, the nonlinearity due to shocks is even more challenging to capture using surrogates.

We compare the surrogate and CFD results in Figs. 20 and 21. The mean absolute error of the C_d and C_l predictions are 5.04 counts and 12.00 counts, respectively. We find that 85% of the absolute error in C_d prediction are within 12.5 counts, while 85% of the errors in C_l prediction are within 24.4 counts. Comparing the statistics of absolute error with those from the subsonic regime



(a) The subsonic case achieves a mesh convergence order of 1.554

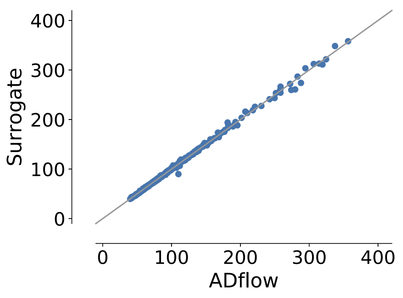


(b) The transonic case achieves a mesh convergence order of 1.235

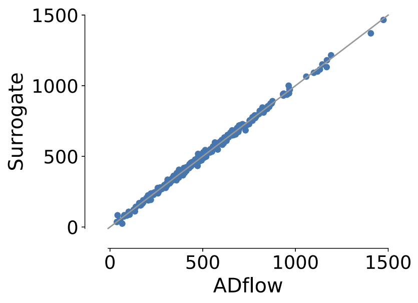
Fig. 17. Mesh convergence study showing the Richardson extrapolation (R.E.) value.

Table 6
Eight-layer MLP, six-layer MLP and six-layer MLSTM surrogates are constructed for the predictions of C_d, C_l and C_p within the subsonic regime. The input parameters are listed in Table 3.

Layers	C_d surrogate	C_l surrogate	C_p surrogate
L1	200 neurons, `tanh`	120 neurons, `tanh`	180 cells, `tanh`
L2	180 neurons, `sigmoid`	140 neurons, `sigmoid`	160 cells, `sigmoid`
L3	160 neurons, `selu`	160 neurons, `selu`	140 cells, `selu`
L4	160 neurons, `selu`	180 neurons, `selu`	120 cells, `selu`
L5	160 neurons, `selu`	200 neurons, `leaky_relu`	80 cells, `relu`
L6	140 neurons, `leaky_relu`	200 neurons, `leaky_relu`	40 cells, `relu`
L7	140 neurons, `leaky_relu`		
L8	140 neurons, `leaky_relu`		

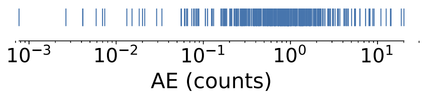


(a) C_d comparison (counts)

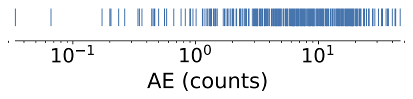


(b) C_l comparison (counts)

Fig. 18. The subsonic neural network surrogates match well with the model observations. The grey line represents a perfect match between predictions and observations.

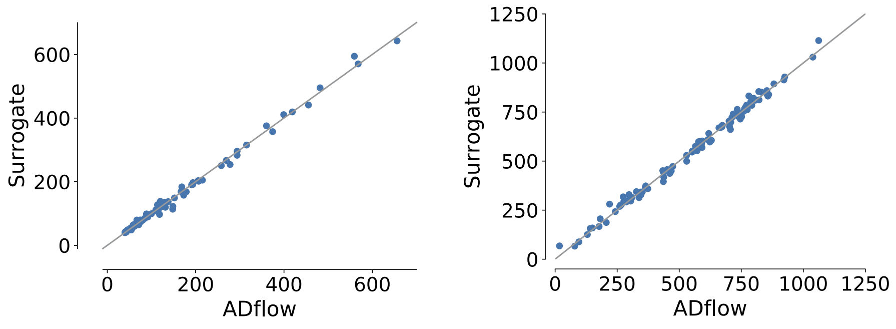


(a) C_d prediction, mean absolute error = 1.41 counts and 85% absolute error ≤ 2 counts



(b) C_l prediction, mean absolute error = 9.61 counts and 85% absolute error ≤ 18.4 counts

Fig. 19. The absolute error is calculated for C_d and C_l within subsonic regime.



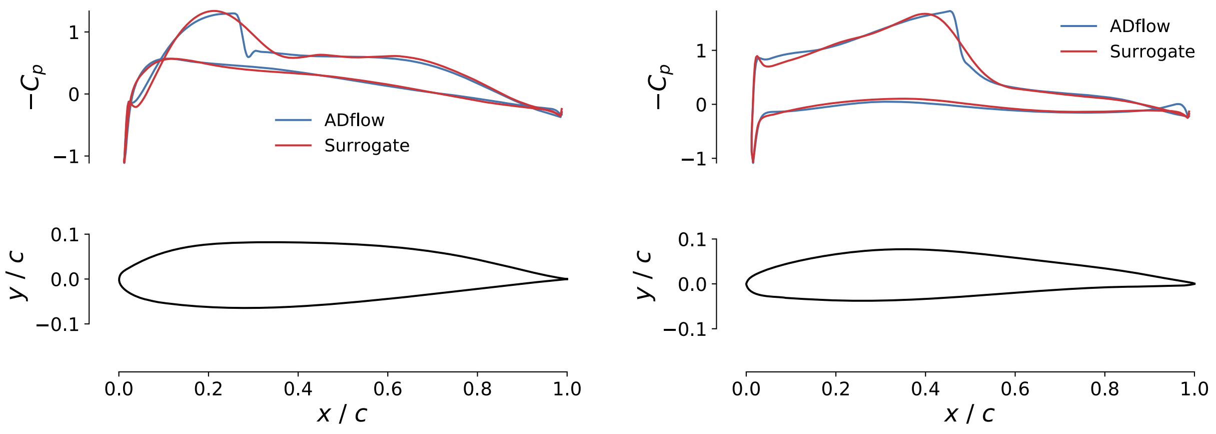
(a) C_d comparison (counts) (b) C_l comparison (counts)

Fig. 20. The transonic neural network surrogate modes match well with the model observations. The grey line represents a perfect match between predictions and observations.

10^{-1}
10^0
10^1
AE (counts)

(a) C_d prediction, mean absolute error = 5.04 counts and 85% absolute error $\leq$ 12.5 counts (b) C_l prediction, mean absolute error = 12.00 counts and 85% absolute error $\leq$ 24.4 counts

Fig. 21. Absolute errors of transonic neural network surrogates.



(a) Random airfoil 1 (b) Random airfoil 2

Fig. 22. Transonic regime C_p comparison between CFD results and surrogate predictions for random airfoils in testing data set.

Table 7
Errors for the subsonic surrogate model.

	C_d	C_l	C_p
σ_{test}	56.91	236.00	0.44
RMSE	2.77	12.90	0.0372
ϵ / $\bar{\epsilon}$ (%)	2.26	2.34	6.13

shows again that surrogate models' predictive performance is more challenging for the higher Mach numbers.

We show a comparison of C_p distributions between CFD results and surrogate predictions in Fig. 22. The airfoil shapes and flight conditions are randomly selected from the transonic testing data set. The MLSTM surrogate predictions agree well with the CFD results, although the predicted C_p slightly smooths out the shock wave.

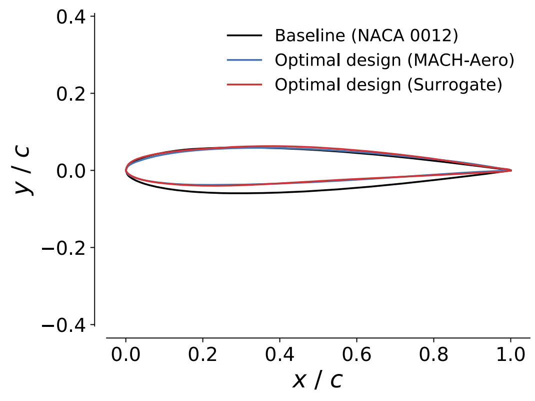
3.4. Airfoil optimization

We demonstrate the proposed framework on airfoil shape optimization problems for both the subsonic and transonic flight regimes. The optimization problem formulation is detailed in Table 10. The objective is to minimize C_d with respect to the BSpline-GAN variables and α. The flight conditions are set to $Re = 6.5 \times 10^6$ and a constant M.

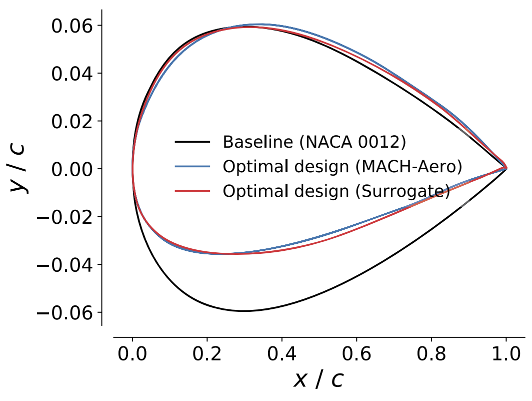
The constraints include a constant target C_l and thickness constraints within [0.8, 3.0] of the baseline thickness. These thickness constraints are enforced at six x-coordinate stations evenly distributed within $[0.01, 0.99]$ of the chord for the subsonic case. We set 100 x-coordinate stations for the transonic case because C_d is more sensitive to local airfoil shape, making it susceptible to oscillations between x stations. For the subsonic surrogate-based optimization, we consider the thickness constraints at the six locations that are closest to the evenly distributed coordinates. For the transonic surrogate-based optimization, we consider thickness constraints at 1,000 x-coordinates by interpolating through the BSplineGAN-generated airfoil coordinates. We limit the thickness-constraint upper bounds to be $0.1\mathbf{t}_{\text{base}}$ higher than lower bounds ($[0.8, 0.9]\mathbf{t}_{\text{base}}$ in this transonic surrogate-based optimization). The unique setup of the transonic surrogate-based optimizations leads to good agreement with CFD-based optimizations.

Table 8
All transonic neural network surrogates have six layers; MLP and MLSTM surrogates are constructed for scalar and vector quantities, respectively. The input parameters are listed in Table 3.

Layers	C_d surrogate	C_l surrogate	C_p surrogate
L1	180 neurons, `tanh`	100 neurons, `tanh`	180 cells, `tanh`
L2	160 neurons, `sigmoid`	120 neurons, `sigmoid`	160 cells, `sigmoid`
L3	140 neurons, `selu`	140 neurons, `selu`	140 cells, `selu`
L4	120 neurons, `selu`	160 neurons, `selu`	120 cells, `selu`
L5	100 neurons, `leaky_relu`	180 neurons, `leaky_relu`	80 cells, `relu`
L6	80 neurons, `leaky_relu`	200 neurons, `leaky_relu`	40 cells, `relu`



(a) Equal axis



(b) Normal axis

Fig. 23. Comparison of optimal airfoils for the surrogate-based and CFD-based optimizations starting from a NACA 0012 baseline (subsonic case).

Table 9
Errors for the transonic surrogate model.

	C_d	C_l	C_p
σ_{test}	126.67	238.74	0.51
RMSE	8.76	16.13	0.084
ϵ / $\bar{\epsilon}$ (%)	4.65	2.87	11.72

C_d and C_l are predicted by the trained MLP surrogates, which also provide the corresponding gradient information via the back-propagation within `Tensorflow`. Since we use the BSplineGAN to generate y coordinates of upper and lower airfoil surfaces at the same x coordinates, the thickness and corresponding gradient with respect to design variables are also available via backpropagation. We verify the surrogate-based optimal designs against the CFD-based optimal and the predicted C_d, C_l, and C_p against the CFD simulation results.

3.4.1. Subsonic airfoil optimization

We set up the subsonic case following Table 10 with $C_l^* = 0.4$, $M = 0.6$, and a baseline design of NACA 0012. The surrogate-based and CFD-based optimizations converge to almost the same optimal design, as shown in Fig. 23. We verify the results by running CFD simulations on baseline and surrogate-based optimal designs (Table 11).

The results show that the predicted C_d of the baseline design at $C_l = 0.4$ is 87.98 drag counts, while the CFD result is 88.34, making absolute error 0.36 drag counts. The predicted C_d of the optimal design at $C_l = 0.4$ is 79.59 drag counts, while the CFD result is 81.75; an absolute error of 2.16 drag counts. These results agree with the verification on the testing data set and prove the surrogate-based optimizations being effective.

We predict C_p distributions for both baseline and surrogate-based optimal design and compare with CFD results as shown in Fig. 24. The results show that the MLSTM surrogate captures the characteristics of C_p distributions on the baseline and optimal designs well. The RMSEs on the C_p distributions of the baseline and optimal designs are 0.024 and 0.018, respectively. The RMSE values are consistent with the RMSE (0.037) of the whole testing data set.

The surrogate-based optimization and C_p prediction process take around three seconds using one-processor on a personal desktop computer. In contrast, the simulation-driven optimization integrated with an adjoint solver for gradient information requires around one hour using 48 processors on a high-performance computing (HPC) cluster.

3.5. Transonic airfoil optimization

We also demonstrate the surrogate-based airfoil shape optimization for a transonic flight condition. We tested a series of transonic design cases, all of which agree well with direct CFD-based optimizations. Here shows a case starting with a baseline of a typical airfoil, RAE 2822. The details are similar to the subsonic case (see Table 10) except $C_l^* = 0.6$ and $M = 0.7$. The surrogate-based optimization and CFD-based optimization reach almost the same optimal super-critical airfoil design, as shown in Fig. 25. We verify the results by running CFD on the baseline and surrogate-based optimal designs (see Table 12).

The results show that the predicted C_d of baseline design at $C_l = 0.6$ counts is 95.78 drag counts, while the CFD result is 95.28, corresponding to an absolute error of 0.5 drag counts. Similarly, the predicted C_d of surrogate-based optimal design at $C_l = 0.6$ counts is 88.17 drag counts, while the CFD result is 88.98 drag counts, corresponding to an absolute error of 0.81 drag counts. Both absolute errors are within one drag count representing a good surrogate predictive performance.

We also compare the C_p distributions of the baseline and surrogate-based optimal designs in Fig. 26. The predicted C_p distributions match the CFD results well. There exist minor differences on the upper airfoil surface near the fore section because shock normally happens in that region, which is challenging for the surrogate model to capture. The C_p distribution RMSEs of the predicted baseline and surrogate-based optimal using the CFD re-

Table 10
Airfoil optimization problem formulation.

	Function or variable	Description	Quantity
minimize	C_d	Drag coefficient	
w.r.t.	$\mathbf{x}$	BSplineGAN variables	26
	α	Angle of attack	1
		Total design variables	**27**
subject to	$C_l = C_l^*$	Lift-coefficient constraint	1
	$0.8\mathbf{t}_{\text{base}} \leq \mathbf{t} \leq 3.0\mathbf{t}_{\text{base}}$	Thickness constraints	6 or 100
		Total constraints	**7** or **101**
Conditions	M	Mach number	
	$Re = 6.5 \times 10^6$	Reynolds number	

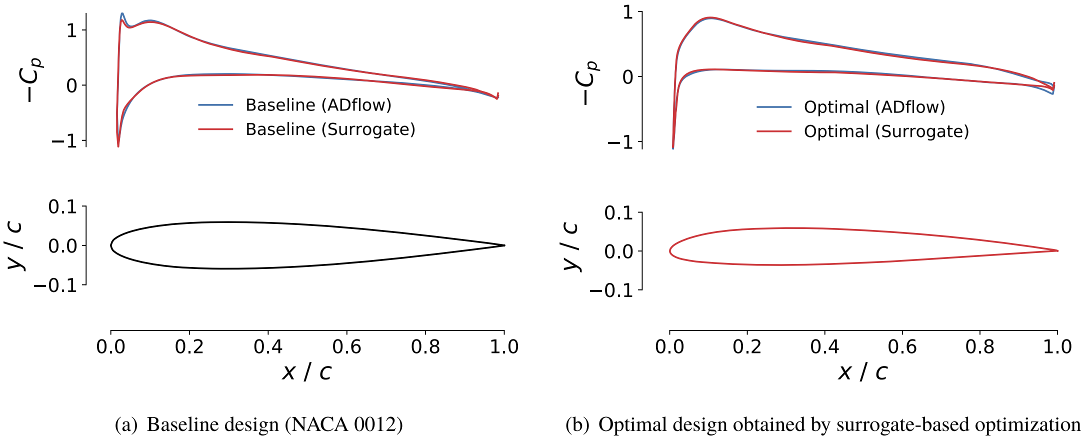


Fig. 24. Comparison of C_p distributions for CFD (blue) and MLSTM surrogate (red) for this subsonic case.

Table 11
Comparison of baseline and optimized C_d and C_l computed by MLP surrogates and CFD (subsonic case).

	C_d (counts)	C_l
Baseline (Surrogate)	87.98	0.4
Optimized (Surrogate)	79.59	0.4
Baseline (CFD)	88.34	0.4
Optimized (CFD)	81.75	0.4

Table 12
Comparison of baseline and optimized C_d and C_l computed by MLP and CFD (transonic case).

	C_d (counts)	C_l
Baseline (Surrogate)	117.66	0.6
Optimized (Surrogate)	85.55	0.6
Baseline (CFD)	113.11	0.6
Optimized (CFD)	86.55	0.6

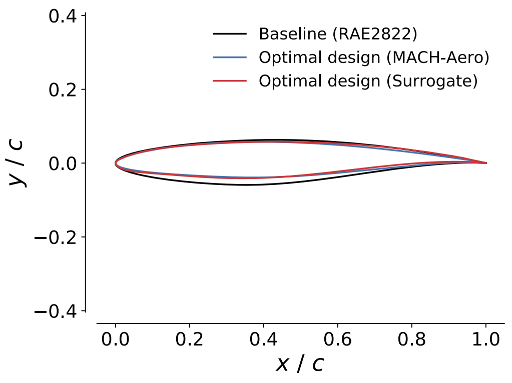


Fig. 25. Optimal airfoils for the surrogate-based and CFD-based optimizations both starting from a RAE 2822 baseline are compared (transonic case).

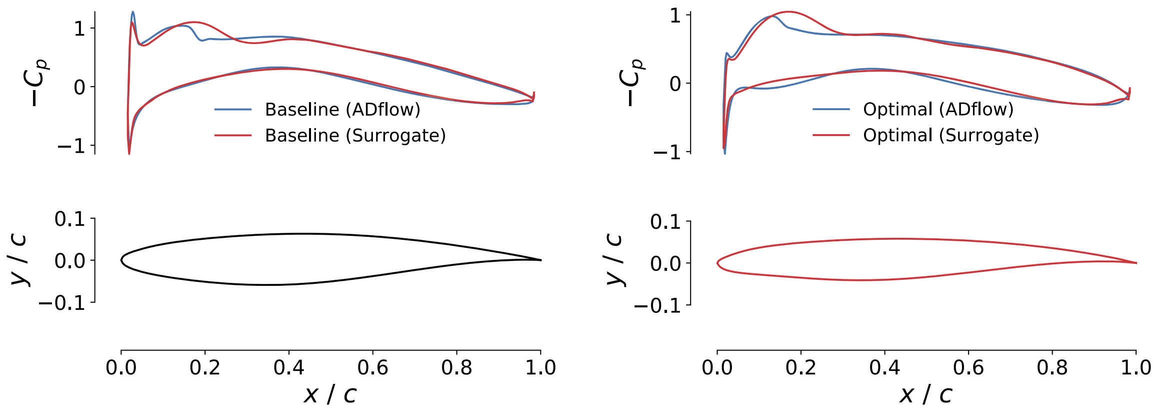
sults as the reference are 0.061 and 0.071, respectively. As for the subsonic results, the RMSE values are consistent with the RMSE (0.084) of the whole transonic testing data set.

Similarly to the subsonic case, the surrogate-based optimization and C_p prediction require only three seconds on a personal desktop computer. Still, the simulation-driven optimization requires twice as much time as the subsonic case (around two hours using 48 processors on an HPC cluster).

4. Conclusions

In this work, we propose a generalized airfoil aerodynamic design framework that performs airfoil shape optimization in a fast, interactive manner. The proposed framework is based on data-driven neural network surrogates. The surrogate inputs are the airfoil geometric parameters and the flight conditions (Mach number, Reynolds number, and angle of attack). The Mach number ranges from 0.3 to 0.7, enabling the framework to perform airfoil aerodynamic optimization in both the subsonic and transonic regimes.

To improve the predictive performance and reduce the required number of training points, we develop and implement the following techniques. The B-spline-based generative adversarial networks (BSplineGAN), trained on a post-processed airfoil database, efficiently generates realistic airfoils and automatically reduces the design space. The Gaussian copula dependent sampling covers Mach and Reynolds pairs of typical aircraft to reduce the input space by avoiding unpractical ones. We divide the Mach range into $[0.3, 0.6]$ and $(0.6, 0.7]$ intervals for better predictive performance. We apply data-driven neural network surrogates to capture complex mapping patterns and handle the large data set. Multilayer perceptron



(a) Baseline design (RAE 2822)

(b) Optimal design obtained from surrogate-based optimization

Fig. 26. Comparison of C_p distributions from CFD and MLSTM surrogate model for this transonic case.

surrogates are used to predict the drag and lift coefficients. Long short-term memory models are used to predict the pressure distributions for richer design information.

The surrogate models are verified to have good accuracy in testing data sets consisting of BSplineGAN-generated airfoil shapes for the sampled flight conditions. The proposed surrogate-based framework is demonstrated for subsonic and transonic flight conditions and compared with results based on direct simulation evaluations. The optimal airfoil shapes match the simulation-based results, and the predictive performance on aerodynamic quantities achieves good accuracy. The surrogate models have better accuracy in the subsonic regime than in the transonic regime, which is not surprising given the nonlinearity of transonic flow.

We achieve fast, interactive airfoil aerodynamic design completed with the resulting C_p distributions using the proposed framework. The whole process takes a few seconds using one processor on a personal desktop computer. In contrast, a simulation-driven gradient-based airfoil optimization takes one to two hours using 48 processors on an high-performance computing cluster.

The accuracy and speed achieved with the design framework allow us to integrate the framework into Webfoil and perform interactive airfoil aerodynamic optimization on any modern computing device.

Despite the satisfactory performance, the C_p prediction in both subsonic and transonic regions still has room for improvement. We can do better post-processing on the simulation-based C_p distributions to filter out the trailing-edge-pressure-spike issue for both subsonic and transonic regimes. Moreover, we would also consider combining neural networks with reduced-order modeling to capture the nonlinearity on transonic C_p distributions.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was partially supported by the Air Force Office of Scientific Research (AFOSR) MURI on "Managing multiple information sources of multi-physics systems", Program Officer Jean-Luc Cambier, Award Number FA9550-15-1-0038.

References

[1] A. Jameson, J. Vassberg, Computational fluid dynamics for aerodynamic design—its current and future impact, in: 39th Aerospace Sciences Meeting and Exhibit, American Institute of Aeronautics and Astronautics, Reno, NV, 2001.

[2] J.R.R.A. Martins, Perspectives on aerodynamic design optimization, in: AIAA SciTech Forum, AIAA, Orlando, FL, 2020.

[3] Y. Yu, Z. Lyu, Z. Xu, J.R.R.A. Martins, On the influence of optimization algorithm and starting design on wing aerodynamic shape optimization, Aerosp. Sci. Technol. 75 (2018) 183–199, https://doi.org/10.1016/j.ast.2018.01.016.

[4] A. Jameson, Aerodynamic Shape Optimization Using the Adjoint Method, Lecture Series, Von Karman Institute for Fluid Dynamics, Rode Saint Genèse, Belgium, 2003.

[5] G.K.W. Kenway, C.A. Mader, P. He, J.R.R.A. Martins, Effective adjoint approaches for computational fluid dynamics, Prog. Aerosp. Sci. 110 (2019) 100542, https://doi.org/10.1016/j.paerosci.2019.05.002.

[6] P. He, C.A. Mader, J.R.R.A. Martins, K.J. Maki, DAFoam: an open-source adjoint framework for multidisciplinary design optimization with OpenFOAM, AIAA J. 58 (3) (2020), https://doi.org/10.2514/1.J058853.

[7] B. Peherstorfer, P.S. Beran, K.E. Willcox, Multifidelity Monte Carlo estimation for large-scale uncertainty propagation, in: 2018 AIAA Non-Deterministic Approaches Conference, American Institute of Aeronautics and Astronautics, 2018.

[8] S. Koziel, L. Leifsson (Eds.), Surrogate-Based Modeling and Optimization, Springer, New York, 2013.

[9] M.A. Bouhlel, J.T. Hwang, N. Bartoli, R. Lafage, J. Morlier, J.R.R.A. Martins, A Python surrogate modeling framework with derivatives, Adv. Eng. Softw. 135 (2019) 102662, https://doi.org/10.1016/j.advengsoft.2019.03.005.

[10] A.I. Forrester, A.J. Keane, Recent advances in surrogate-based optimization, Prog. Aerosp. Sci. 45 (1) (2009) 50–79, https://doi.org/10.1016/j.paerosci.2008.11.001.

[11] M.A. Bouhlel, S. He, J.R.R.A. Martins, Scalable gradient-enhanced artificial neural networks for airfoil shape design in the subsonic and transonic regimes, Struct. Multidiscip. Optim. 61 (2020) 1363–1376, https://doi.org/10.1007/s00158-020-02488-5.

[12] W. Liu, S. Batill, Gradient-enhanced neural network response surface approximations, in: 8th Symposium on Multidisciplinary Analysis and Optimization, Multidisciplinary Analysis Optimization Conferences, AIAA, 2019.

[13] A. Feldstein, D. Lazzara, N. Princen, K. Willcox, Multifidelity data fusion: application to blended-wing-body multidisciplinary analysis under uncertainty, AIAA J. 58 (2) (2020) 889–906.

[14] R.P. Liem, J.R.R.A. Martins, Surrogate models and mixtures of experts in aerodynamic performance prediction for mission analysis, in: 15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Atlanta, GA, 2014, AIAA-2014-2301.

[15] M. Rumpfkeil, P. Beran, Multifidelity sparse polynomial chaos surrogate models applied to flutter databases, AIAA J. 58 (3) (2020) 1292–1303.

[16] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (2015) 436–444.

[17] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.

[18] M. Raissi, P. Perdikaris, G. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (1) (2018) 686–707.

[19] M. Raissi, Deep hidden physics models: deep learning of nonlinear partial differential equations, J. Mach. Learn. Res. 19 (1) (2018) 932–955.

[20] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, S. Kaushik, Prediction of aerodynamic flow fields using convolutional neural networks, Comput. Mech. 64 (2) (2019) 525–545.

[21] V. Sekar, M. Zhang, C. Shu, B. Khoo, Inverse design of airfoil using a deep convolutional neural network, AIAA J. 57 (3) (2019) 993–1003.

[22] V. Sekar, B. Khoo, Fast flow field prediction over airfoils using deep learning approach, Phys. Fluids 31 (5) (2019) 057103(1)–057103(14).

[23] W. Chen, M. Fuge, BezierGAN: automatic generation of smooth curves from interpretable low-dimensional parameters, arXiv:1808.08871, 2018.

[24] W. Chen, K. Chiu, M. Fuge, Aerodynamic design optimization and shape exploration using generative adversarial networks, in: AIAA SciTech Forum, San Diego, USA, 2019.

[25] Y. Azabi, A. Savvaris, T. Kipouros, Artificial intelligence to enhance aerodynamic shape optimisation of the Aegis UAV, Mach. Learn. Knowl. Extract. 1 (2019) 552–574.

[26] J. Li, M.A. Bouhlel, J.R.R.A. Martins, Data-based approach for fast airfoil analysis and optimization, AIAA J. 57 (2) (2019) 581–596, https://doi.org/10.2514/1.J057129.

[27] M.A. Bouhlel, J.R.R.A. Martins, Gradient-enhanced kriging for high-dimensional problems, Eng. Comput. 1 (35) (2019) 157–173, https://doi.org/10.1007/s00366-018-0590-x.

[28] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, in: Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger (Eds.), Generative Adversarial Nets, in: Advances in Neural Information Processing Systems, vol. 27, Curran Associates, Inc., 2014, pp. 2672–2680.

[29] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, P. Abbeel, Infogan: interpretable representation learning by information maximizing generative adversarial nets, arXiv:1606.03657, 2016.

[30] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: a system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283, https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[31] A. Sherstinsky, Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network, Phys. D: Nonlinear Phenom. 404 (2020) 132306, https://doi.org/10.1016/j.physd.2019.132306.

[32] A.B. Lambe, J.R.R.A. Martins, Extensions to the design structure matrix for the description of multidisciplinary design, analysis, and optimization processes, Struct. Multidiscip. Optim. 46 (2012) 273–284, https://doi.org/10.1007/s00158-012-0763-y.

[33] N. Secco, G.K.W. Kenway, P. He, C.A. Mader, J.R.R.A. Martins, Efficient mesh generation and deformation for aerodynamic shape optimization, AIAA J. 59 (4) (2021), https://doi.org/10.2514/1.J059491.

[34] C.A. Mader, G.K.W. Kenway, A. Yildirim, J.R.R.A. Martins, ADflow: an open-source computational fluid dynamics solver for aerodynamic and multidisciplinary optimization, J. Aerosp. Inform. Syst. 17 (9) (2020), https://doi.org/10.2514/1.I010796.

[35] A. Yildirim, G.K.W. Kenway, C.A. Mader, J.R.R.A. Martins, A Jacobian-free approximate Newton–Krylov startup strategy for RANS simulations, J. Comput. Phys. 397 (2019) 108741, https://doi.org/10.1016/j.jcp.2019.06.018.

[36] P.E. Gill, W. Murray, M.A. Saunders, SNOPT: an SQP algorithm for large-scale constrained optimization, SIAM Rev. 47 (1) (2005) 99–131, https://doi.org/10.1137/S0036144504446096.

[37] N. Wu, G. Kenway, C.A. Mader, J. Jasa, J.R.R.A. Martins, pyOptSparse: a Python framework for large-scale constrained nonlinear optimization of sparse systems, J. Open Sour. Softw. 5 (54) (2020) 2564, https://doi.org/10.21105/joss.02564.

[38] L. Piegl, W. Tiller, The NURBS Book, Springer Science & Business Media, 2012.

[39] M.D. McKay, R.J. Beckman, W.J. Conover, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 21 (2) (1979) 239–245.

[40] R.B. Nelsen, An Introduction to Copulas, 2nd ed., Springer-Verlag, New York, 2006.

[41] R. Lebrun, A. Dutfoy, A generalization of the Nataf transformation to distributions with elliptical copula, Probab. Eng. Mech. 24 (2) (2009) 172–178, https://doi.org/10.1016/j.probengmech.2008.05.001.

[42] G.K.W. Kenway, C.A. Mader, P. He, J.R.R.A. Martins, CFD discrete adjoint benchmarks, Mendeley Data, https://doi.org/10.17632/w4hsj8wzm8, 2019.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Rettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[44] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, preprint, arXiv:1412.6980, 2014.

[45] R.C. Staudemeyer, E.R. Morris, Understanding LSTM—A Tutorial into Long Short-Term Memory Recurrent Neural Networks, 2019.

[46] A. Khan, A. Sohail, U. Zahoora, A.S. Qureshi, A survey of the recent architectures of deep convolutional neural networks, Artif. Intell. Rev. (2020) 1–62.

[47] R. Pascanu, C. Gulcehre, K. Cho, Y. Bengio, How to construct deep recurrent neural networks, in: Proceedings of the Second International Conference on Learning Representations, 2014.

[48] N. Qian, On the momentum term in gradient descent learning algorithms, Neural Netw. 12 (1) (1999) 145–151, https://doi.org/10.1016/S0893-6080(98)00116-6.

[49] S. Tieleman, G. Hinton, Lecture 6.5—RMSProp: Neural Networks for Machine Learning, COURSERA Technical Report, 2012.

[50] G.K. Kenway, G.J. Kennedy, J.R.R.A. Martins, A CAD-free approach to high-fidelity aerostructural optimization, in: Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference, Fort Worth, TX, 2010, https://doi.org/10.2514/6.2010-9231.