NeuralFoil: An Airfoil Aerodynamics Analysis Tool Using Physics-Informed Machine Learning

Peter Sharpe* and R. John Hansman†
Massachusetts Institute of Technology, Cambridge, MA

NeuralFoil **is an open-source Python-based tool for rapid aerodynamics analysis of airfoils, similar in purpose to XFoil. Speedups ranging from 8x to 1,000x over XFoil are demonstrated, after controlling for equivalent accuracy. NeuralFoil computes both global and local quantities (lift, drag, velocity distribution, etc.) over a broad input space, including: an 18-dimensional space of airfoil shapes, possibly including control deflections; a 360° range of angles of attack; Reynolds numbers from 10^2 to 10^{10}; subsonic flows up to the transonic drag rise; and with varying turbulence parameters. Results match those of XFoil closely: the mean relative error of drag is 0.37% on simple cases, and remains as low as 2.0% on a test dataset with numerous post-stall and transitional cases. NeuralFoil facilitates gradient-based design optimization, due to its C^∞-continuous solutions, automatic-differentiation-compatibility, and bounded computational cost without non-convergence issues.**

NeuralFoil is a hybrid of physics-informed machine learning techniques and analytical models. Here, physics information includes symmetries that are structurally embedded into the model architecture, feature engineering using domain knowledge, and guaranteed extrapolation to known limit cases. This work also introduces a new approach to surrogate model uncertainty quantification that supports robust design optimization.

This work discusses the methodology and performance of NeuralFoil with several case studies, including a practical airfoil design optimization study including both aerodynamic and non-aerodynamic constraints. Here, NeuralFoil optimization is able to produce airfoils nearly identical in performance and shape to expert-designed airfoils within seconds; these computationally-optimized airfoils provide a useful starting point for further expert refinement.

I. Nomenclature

β = Prandtl-Glauert compressibility correction factor, defined as $\sqrt{1 - M_\infty^2}$

*PhD Candidate, AIAA Student Member
†T. Wilson Professor in Aeronautics, AIAA Fellow

BL	=	boundary layer
CFD	=	computational fluid dynamics
C_D	=	drag coefficient
C_L	=	lift coefficient
C_M	=	moment coefficient
C_p	=	local pressure coefficient (in the real compressible flow)
C_{p_0}	=	local pressure coefficient in the equivalent incompressible flow
$C_{p_{\text{min}}}$	=	minimum local pressure coefficient (in the real compressible flow)
$C_{p_{0,\text{min}}}$	=	minimum local pressure coefficient in the equivalent incompressible flow
c	=	airfoil chord
$c_{\mathcal{D}}$	=	dissipation coefficient
c_f	=	skin friction coefficient
c_{lower}	=	Kulfan (CST) shape parameters associated with the airfoil's lower surface
c_{upper}	=	Kulfan (CST) shape parameters associated with the airfoil's upper surface
γ	=	ratio of specific heats of the working fluid; 1.4 for air near standard conditions
H	=	BL shape parameter, defined as δ^*/θ
H^*	=	BL kinetic energy shape parameter, defined as θ^*/θ
LE	=	airfoil leading edge
M	=	local Mach number
M_∞	=	freestream Mach number
M_{crit}	=	critical Mach number, defined as the lowest freestream Mach number where any point in the flow is supersonic
M_{dd}	=	drag-divergent Mach number, defined as the freestream Mach number above which drag begins to rise rapidly ($\partial(C_D)/\partial M_\infty \geq 0.1$)
ν	=	kinematic viscosity
N_{crit}	=	BL critical amplification factor (affected by surface roughness, freestream turbulence)
RANS	=	Reynolds-averaged Navier-Stokes equations
Re_c	=	chord Reynolds number, defined as $u_\infty c/\nu$
Re_θ	=	momentum-thickness Reynolds number, a local BL quantity defined as $u_e\theta/\nu$
s	=	coordinate along the airfoil surface, starting at the TE and proceeding over the top surface
θ	=	BL momentum thickness
θ_{TE}	=	Trailing-edge angle

TE	=	airfoil trailing edge
u_e	=	BL edge velocity
u_∞	=	freestream velocity magnitude
$u_{\max}$	=	maximum velocity magnitude at any point in the flow field
x_{tr}	=	actual laminar-turbulent transition location, relative to the leading edge; appropriate suffixes denote top- and bottom-surface measures
$x_{\text{tr,forced}}$	=	location of any forced trips for bypass transition (e.g., turbulators, rivets), if present; $x_{\text{tr,forced}}/c = 1$ if absent
x/c	=	nondimensional distance along the airfoil chord (LE $\rightarrow$ 0, TE $\rightarrow$ 1)
y/c	=	nondimensional distance normal to the airfoil chord (LE and TE at 0, barring any control surface deflections)
z_{in}	=	input latent space vector
z_{out}	=	output latent space vector

II. Introduction

In conceptual aircraft design, the problem of shaping a typical wing is usually decomposed into two parts: planform design and airfoil design. The latter, which is the focus of this work, is a multidisciplinary design problem that requires consideration of a variety of aerodynamic, structural, and manufacturing objectives and constraints. A non-exhaustive list of major considerations could include:

- Profile drag across the expected operating range of the airfoil (spanning lift coefficients, Reynolds numbers, and Mach numbers), including adequate off-design performance [**?**];
- Pitching moment and aft-camber coefficients, which can drive tail sizing (modifying trim drag), affect divergence speed;
- Hinge moments and control effectiveness of any control surfaces, which drive actuator design and weight;
- Stall behavior, which can affect handling qualities and safety;
- Thickness at various points, in order to accommodate fuel volume and required structural members to resist failure (e.g., by bending, buckling, divergence, flutter, or control reversal);[**?**]
- Sensitivity to boundary layer performance, freestream turbulence, and trips, all of which impose constraints on surface finish, cleanliness, and manufacturing tolerances [1**?** , 2];
- Peak suction pressures, which affect the critical Mach number in transonic applications or cavitation in hydrodynamic applications;
- Shock stability and buffet considerations in transonic applications;

- Manufacturability, which might include flat-bottom airfoil sections, strictly-convex airfoil shapes (e.g., to accommodate shrink-coverings, which are common in ultra-lightweight applications [**?**]), or restrictions on trailing-edge angle.

These airfoil design drivers often differ considerably at different locations along the span of the wing, which often leads to a family of airfoils being used in the design of a given wing. To fulfill such design requirements, a designer will typically either find an existing airfoil or design a new airfoil. Given the specificity of the requirements illustrated in the list above, designing bespoke airfoils can yield considerable performance improvement.

Currently, three approaches are commonly used to computationally design new airfoils: inverse design methods, direct manual methods, and optimization methods. In the inverse design approach, popularized by Drela's XFoil code [**?**] and Eppler's Profil code [3, 4], conformal mapping methods are used to reconstruct a new airfoil shape from a user-specified pressure distribution. This has the benefit of allowing the engineer to directly operate on the most relevant aerodynamic quantities, and it produces considerable design insight. However, it can be time-consuming to produce airfoils that satisfy non-aerodynamic constraints (e.g., manufacturability, spar thickness) due to the lack of direct control here. Conformal mapping methods are also only strictly applicable to potential-flow-governed regions, so the specified pressure distribution is often only the inviscid, rather than the viscous (true) pressure distribution*.

In the direct manual method (or "geometric design" method, using parlance from XFoil), an engineer formulates an airfoil shape directly and modifies this iteratively by hand. Aerodynamic analysis is performed by any code that will perform the forward problem (geometry $\rightarrow$ aerodynamics), such as XFoil, MSES [5], or any RANS-based CFD code [**?**]. This allows for easier satisfaction of non-aerodynamic constraints. However, it is predictably more difficult to directly target aerodynamic quantities. Significant user expertise is also required to identify the most relevant geometric parameters and to make effective changes to the airfoil shape.

In the optimization approach, a parameterized airfoil shape is optimized to minimize a cost function and satisfy specified constraints (which may be both aerodynamic and non-aerodynamic). At first glance, this appears to be an automation of the direct manual method. However, Drela and others note the surprising difficulty of posing the correct optimization problem [**? ?**], so this approach often requires just as much (if not more) human expertise than the direct manual method. As stated by Drela [**?**], optimization-based airfoil design "is still an iterative cut-and-try undertaking. But compared to [direct] techniques, the cutting-and-trying is not on the geometry, but rather on the precise formulation of the optimization problem." To support this, Drela gives compelling case studies of how airfoil design optimization can go awry in the absence of user review and care.† However, despite these cautionary notes, the optimization-based airfoil design approach also offers compelling benefits: resulting airfoil performance can equal that of airfoils by an expert [**?**], particularly on problems with unique or otherwise non-intuitive constraints. This optimization process can

*This can be alleviated by using nonlinear optimization to target the viscous pressure distribution, albeit with reduced numerical robustness.

†Some codes, like LINDOP [5], alleviate this somewhat by using a hybrid of the direct and optimization approaches: update directions are computed by an optimizer, but the actual changes are reviewed and implemented by a human between iterations.

also require orders of magnitude less engineering time, and it provides a systematic and disciplined approach that is especially suited to the most challenging design problems [6].

In all these methods, some form of a computational tool for airfoil aerodynamics analysis is required. For subsonic airfoils, the gold standard of such tools is XFoil [**?**]. Morgado et al. find that XFoil is more accurate than RANS-CFD-based tools in this regime [7], yet it has a computational cost that is roughly 1,000x lower than RANS approaches – a testament to the power of its modeling approach, which strongly-couples integral boundary layer and potential flow methods. A complete description of this modeling approach is available in Drela's *Aerodynamics of Viscous Fluids* [**?**], and in recent state-of-the-art work by Zhang [**? ?**]. However, despite XFoil's many strengths, it has several attributes that make it less-than-ideal for directly driving numerical optimization studies [**?**]. Among these:

- XFoil is not guaranteed to produce a solution. When an "ambitious" calculation is attempted, XFoil often fails to provide a converged solution; the unconverged result often has wildly-diverging values and is effectively unusable. In some cases, calculations can lead to infinite loops or process crashes due to unhandled exceptions. While this is acceptable in certain applications (e.g., manual direct analysis), it is generally unacceptable for use in numerical optimization. Instead, optimization strongly benefits from a robust analysis tool that always produces a result, even if that result has degraded accuracy; this allows the analysis to steer the optimizer back towards the design space of reasonable airfoils [6]. A particularly useful attribute is when the model is deliberately made to be slightly pessimistic (e.g., overestimate drag) in regions of the design space with high uncertainty, adding further optimization pressure towards reasonable designs with low performance uncertainty.
    - More generally, design optimization is not the only application that strongly benefits from an aerodynamics analysis tool that always produces an answer. Other examples where a non-answer, infinite loop, or crashed process are unacceptable include real-time control (e.g., as an aerodynamic model for a model-predictive controller onboard an aircraft) and flight simulation.
- XFoil solutions are not necessarily unique, and slightly different solutions can be obtained for the same analysis problem (airfoil shape, angle of attack, and Reynolds and Mach numbers). In practice, this manifests as an effective hysteresis depending on whether the angle of attack is swept up or down. This flow non-uniqueness is in fact a real physical[‡] effect [6, 8**?**]. However, this non-uniqueness can be exceptionally problematic for numerical optimization, as an infinitesimal change in an input parameter can result in the solution jumping to a different Newton basis of attraction. Therefore, there is no limit to how sensitive performance can be to input parameters, which hampers techniques like finite-differencing for gradient-based optimization.
- XFoil solutions are non-smooth[§] with respect to input parameters, which makes them fundamentally incompatible with gradient-based optimization. (Any attempt to directly optimize XFoil results with gradient-based methods

[‡]For example, flow over an airfoil may separate as its angle of attack increases past 12°, but it may not fully reattach until the angle of attack descends back to below 11°

[§]precisely, they are C^0 continuous but not C^1 continuous

invariably results in premature stopping at a local minimum.) Interestingly, this is a consequence of how laminar-turbulent transition is handled by XFoil's integral boundary layer solver. This solve requires the use of laminar and turbulent boundary layer *closure models*, which are curve-fitted functions that yield various necessary quantities (H^*, c_f, $c_{\mathcal{D}}$, etc.) of the von Karman integral momentum and kinetic energy equations as a function of the two values that parameterize the boundary layer (H, Re_θ). The laminar and turbulent versions of these functions differ. XFoil implements a cut-cell approach on the transitioning interval, which restores C^0-continuity (i.e., transition won't truly "jump" from one node to another discretely); however, a sharp change in gradient occurs whenever an individual node switches its equation from laminar to turbulent. Adler et al. provide a graphical depiction of this phenomenon [**?**].

- Most interfaces between an optimizer and XFoil communicate through a series of text files (i.e., hard disk), rather than by sharing data in memory (i.e., RAM). Given the quick speed of an individual XFoil run, this input-output overhead imposes a non-negligible performance penalty. While this programming-language-agnostic interface is arguably one of the reasons for XFoil's long-enduring popularity, it comes at the cost of runtime performance if the tool is to be used in a high-throughput setting (e.g., for design optimization or aerodynamic database construction).

This motivates the development of a new airfoil aerodynamics analysis tool that captures the advantages of XFoil (accuracy, speed) while mitigating these drawbacks (i.e., incompatibility with gradient-based optimization, non-convergence challenges). In recent years, many fields have benefited from a hybrid data-and-theory approach [**?**], where data-driven models are used to augment traditional physics-based models with learned closures. This work presents a similar physics-informed approach as applied to analyzing airfoil aerodynamics.

III. NeuralFoil Tool Description and Methodology

A. Overview

NeuralFoil is a tool for rapid aerodynamics analysis of airfoils, similar in purpose to XFoil [**?**]. A precise list of inputs and outputs to NeuralFoil is given in Figure 1.

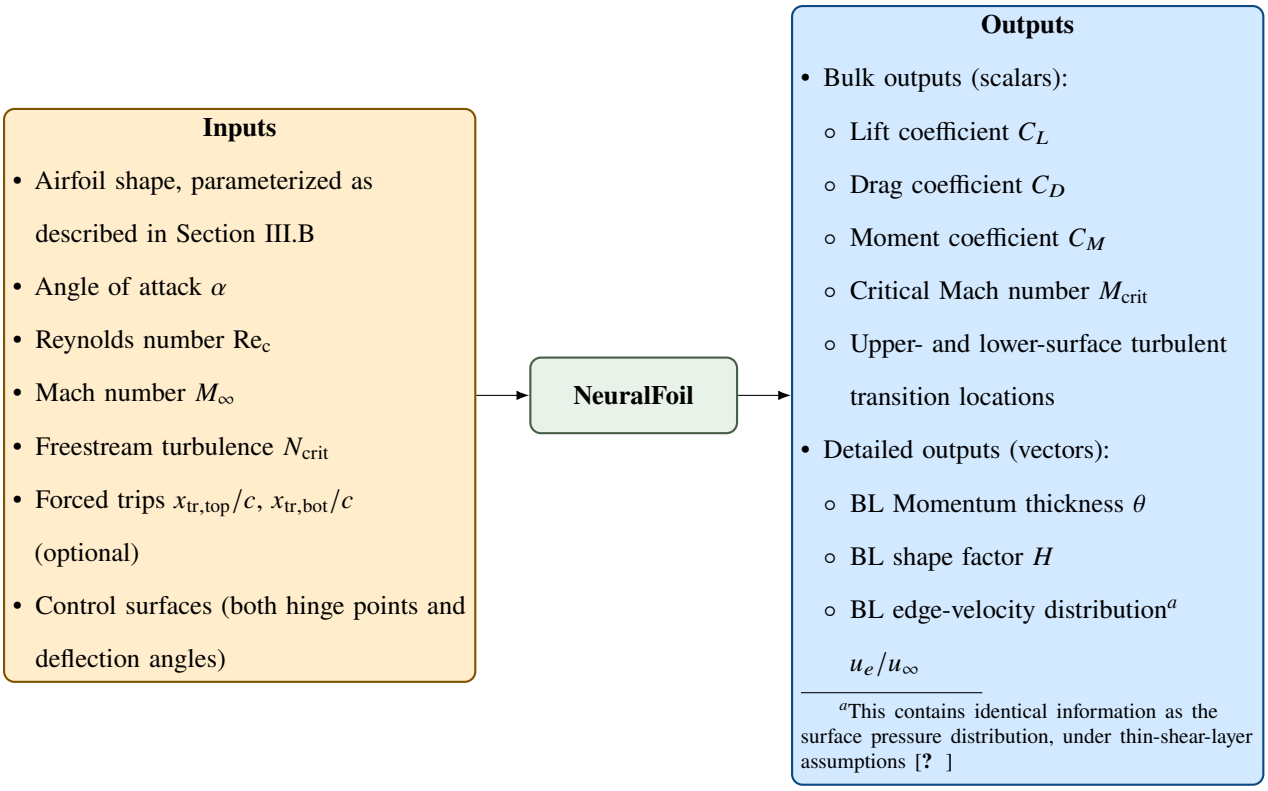


Fig. 1 User-facing inputs and outputs of the NeuralFoil model.

Although NeuralFoil's results will be most accurate in "well-behaved" flow regimes (e.g., attached flow), reasonable aerodynamics estimates can be expected:

- For nearly all practical single-element airfoil shapes that can be analyzed with XFoil, including modifications for trailing-edge control surface deflections up to substantial deflections (roughly ±40°)
- Across the 360° angle of attack range, by leveraging analytical post-stall models regressed from high-α wind tunnel data by Truong [**?**]
- Across a large range of Reynolds numbers (roughly 10^2 to 10^{10}; described in Table 3), with physically-sensible extrapolation even beyond this range (e.g., Stokes flow limit)
- At Mach numbers from zero to the transonic drag rise

NeuralFoil has a mathematical form that is fully explicit (i.e., no iterative solvers are used; there is no value-dependent code execution). This guarantees that a deterministic result is returned in bounded computational time and without any need for initial guesses. This also makes compatability with automatic differentiation frameworks much more straightforward, as the computational graph is static for any input. At a high level, the mathematical model within NeuralFoil consists of the following main steps:

1) **Pre-solve**: Converts the user-supplied airfoil shape into the required parameterization (including any control surfaces, which are made part of the airfoil geometry as described in Section III.B.2)

2) **Encoding**: Transforms all inputs (except Mach number, which is treated later) into an appropriate input vector space for the neural network. This is performed using prescribed functions based on domain knowledge.
3) **Learned Model**: A learned neural network maps from this latent vector space to another latent vector space.
4) **Decoding**: Transforms the outputs of this neural network back into the space of user-facing outputs. This is also performed using prescribed functions based on domain knowledge.
5) **Uncertainty quantification, model fusion, and extrapolation**: Based on the neural network's self-reported trustworthiness (via a process described in Section III.D), the network's results are merged with analytical models for airfoil behavior in massively-separated flow conditions.
6) **Compressibility correction**: The solution is corrected for non-zero Mach numbers using analytical methods, as described in Section III.D.1.

In the following sections, we will describe this model architecture and theoretical basis in more detail. Readers primarily interested in general performance metrics and validation studies are invited to skip ahead to Section IV. Readers primarily interested in a practical aerodynamic shape optimization example are directed to Section V.

B. Pre-Solve

1. Airfoil Geometry Parameterization

NeuralFoil accepts user-specified airfoil shapes in a variety of common formats – for example, as an array of (x, y) coordinates, as a standard coordinate-array `*.dat` file, as a series of CST parameters, or as an Airfoil-class object within the AeroSandbox aircraft design optimization framework [**?**].

Underneath this interface layer, NeuralFoil converts this specified airfoil geometry to an 8-parameter-per-side CST (Kulfan) parameterization, including Kulfan's added leading-edge-modification (LEM) and trailing-edge thickness parameter [**?** **?**]. This gives a total of 18 parameters to describe a given airfoil shape, which are illustrated in Figure 2. This parameterization family was chosen due to work by Masters [**?**] and others, which shows that this is one of the most parameter-efficient representations of airfoil shape. Kulfan's parameterization is strongly related to an orthogonal polynomial decomposition using Bernstein polynomials. Because of this, the format is interpretable: it is a linear combination of mode shapes, each of which is roughly locally-supported[¶]. Another benefit of the CST parameterization is interoperability, as it is commonly implemented in existing aerospace tools such as OpenVSP [**?**].

The CST parameterization allows for varying numbers of degrees of freedom. An 18-parameter representation was chosen based on the work of Masters [**?**], which shows that error in aerodynamic force prediction decreases substantially near this threshold. Thus, this parameterization strikes an acceptable balance between parameterization error and dimensionality. The 18-parameter representation also corresponds to one of the initially-proposed discretization levels proposed by Kulfan [**?**] (labeled in this work as "BPO8"), so this parameterization is a natural choice for compatibility

[¶]this contrasts with approaches such as taking an SVD over a standard airfoil database, which creates less-interpretable mode shapes

with existing airfoil design tools.

Conversion of user-specified airfoils to this format for NeuralFoil to use is automatically and efficiently handled as a least-squares fitting problem.

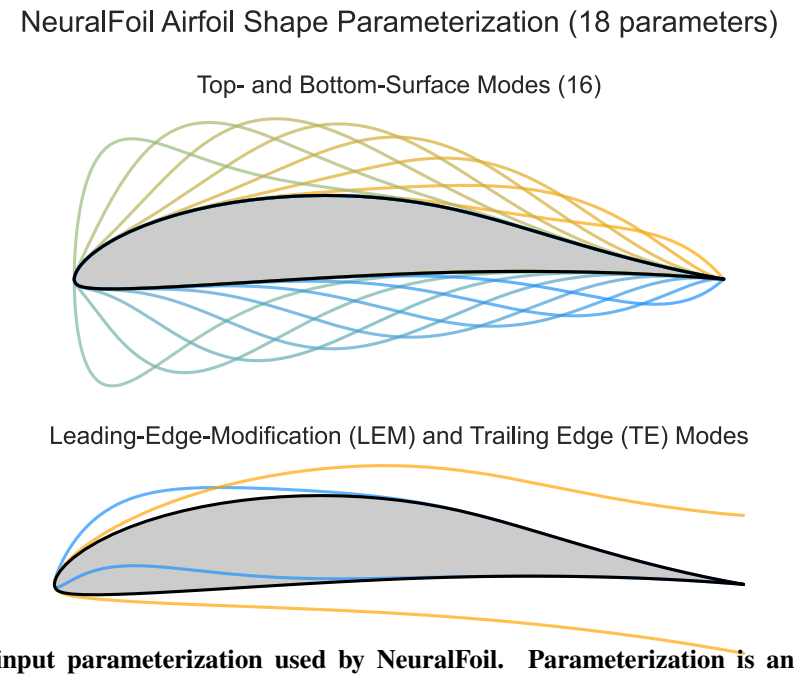


Fig. 2 Geometry input parameterization used by NeuralFoil. Parameterization is an 18-parameter CST (Kulfan) parameterization [? ? ?]. Each colored line in the figure represents a mode shape associated with one of these parameters; modes are linearly combined to form the airfoil shape.

2. Control Surfaces

Control surfaces are handled as a degenerate problem by re-normalizing the deflected airfoil shape. This is illustrated in Figure 3. In step 1, an example airfoil is given. In step 2, a user-specified control surface deflection is applied. In step 3, the airfoil is re-normalized by applying the necessary similarity transformation (rotation, translation, and scaling) such that the leading-edge and trailing-edge locations are placed at their standard $(0, 0)$ and $(1, 0)$ locations in chord-normalized airfoil coordinates. The geometric rotation required for this re-normalization is later applied as a change in the effective angle of attack, $\Delta\alpha$. For consistency, the scaling factor required for this operation is also later used to scale the input Reynolds number appropriately, though the effect of this Reynolds scaling is typically minor. Finally, the resulting pitching moment must also be adjusted to account for the shifting of the force center due to translation during re-normalization.

Also visible in Figure 3 is the geometric effect of restricting the airfoil shape to the space of CST-parameterized airfoils, which is performed along with the re-normalization step between steps 2 and 3. The effect of this is that the sharp corners associated with the control surface deflection are smoothed out, which is a consequence of the smooth mode shapes associated with the CST parameterization. The rationale behind accepting this loss of fidelity here is

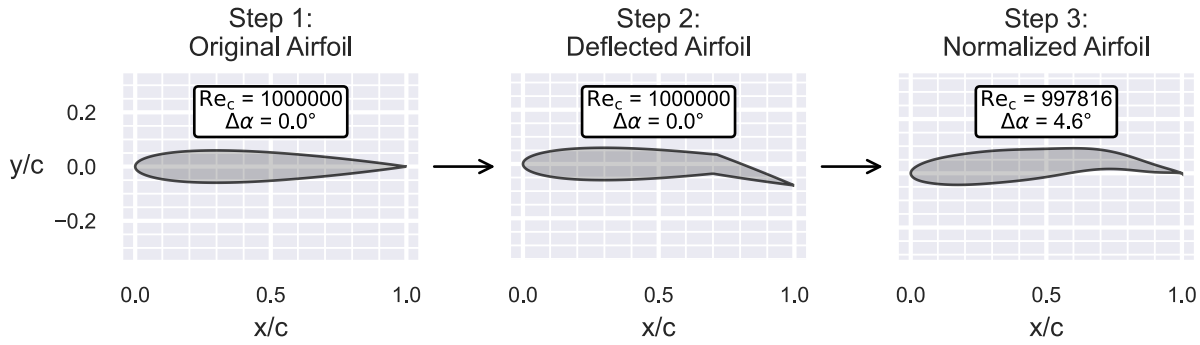


Fig. 3 Illustration of the automatic procedure for handling control surface deflections in NeuralFoil.

that the control surface deflection invariably causes a turbulent transition at the deflection point on the suction side, regardless of whether the sharp or smoothed geometry is used. Because the turbulent boundary layer that follows the hinge is less sensitive to the pressure distribution (and hence, airfoil shape), the loss of geometric accuracy is less significant. Nevertheless, some loss of aerodynamic accuracy is to be expected.

To quantify this loss of accuracy, Figure 4 shows airfoil aerodynamics results obtained by both NeuralFoil and XFoil for various control surface deflections. Notably, reasonably close agreement is seen even for control surface deflections as aggressive as ±40°.

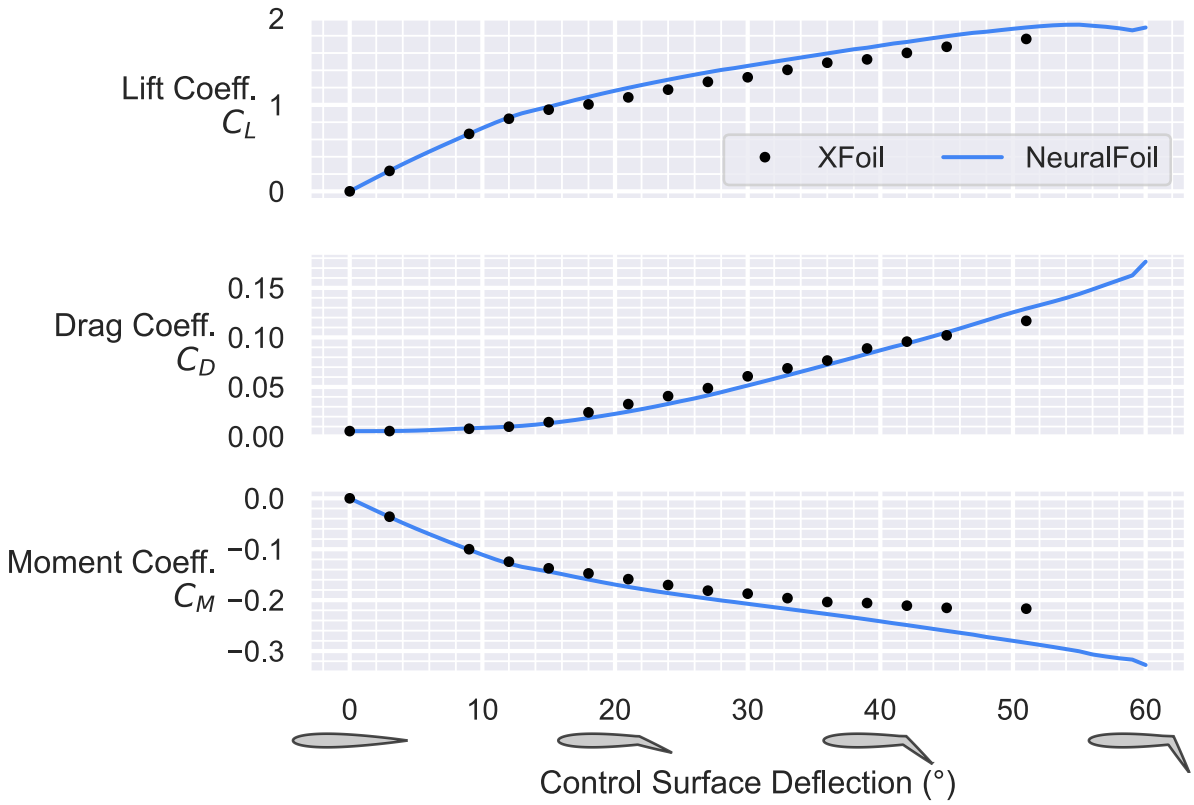


Fig. 4 Accuracy of NeuralFoil on an airfoil with large control surface deflections. Here, we show aerodynamics results from both NeuralFoil and XFoil. All runs are on a NACA0012 airfoil at $\mathrm{Re_c} = 10^6$ and $\alpha = 0°$, with varying control surface deflections on a trailing-edge flap hinged at $x/c = 0.70$.

C. Encoding, Learned Model Architecture, and Decoding

After geometry parameterization, NeuralFoil transforms the user-facing inputs and outputs shown in Figure 1 into intermediate vector spaces (latent spaces) that are more amenable to learning by a neural network. These latent spaces are carefully parameterized such that the learned model has a close-to-affine mapping of inputs to outputs. Effectively, this is feature engineering using domain-specific knowledge.

This encoding/decoding scheme has two major effects. First, it substantially increases the model's parameter efficiency[‖], since the training data has fewer nonlinearities in this latent space. Secondly, the combination of encoding functions, model architecture, and decoding functions can be used to guarantee physically-sound extrapolation beyond the dataset [**?**]. An example that illustrates both effects is the encoding of both the $\mathrm{Re_c}$ input and the C_D output into logspace. This means that an affine model in the latent space corresponds to a power-law $C_D(\mathrm{Re_c})$ model in user-facing space, which is a relationship supported by physical theory for both laminar and turbulent flows (e.g., Falkner-Skan and Schlichting boundary layer models [**?**]).

[‖] i.e., test-set accuracy, relative to the number of parameters (which represent model complexity and computational cost)

1. Encoding

The user-facing input space (shown in Figure 1) is transformed into the following input latent space, which is effectively what is seen by the neural network:

$$z_{\text{in}} = \text{Affine}\left(\begin{bmatrix} \text{Airfoil shape (18 parameters)} \\ \sin(2\alpha) \\ \sin^2(\alpha) \\ \cos(\alpha) \\ \ln(\text{Re}_c) \\ N_{\text{crit}} \\ x_{\text{tr,top,forced}} \\ x_{\text{tr,bot,forced}} \end{bmatrix}\right) \tag{1}$$

The resulting z_{in} is 25-dimensional. Note that the inputs described in Equation 1 also undergo a simple elementwise affine transformation. This transformation is such that the distribution of typical inputs in the latent space has a mean of roughly zero and a standard deviation of roughly one**. Exact scaling and shift factors are available in the open-source codebase described in Section VI. The purpose of this transformation is to improve the stability of the neural network training process, as well as to improve the performance of the weight-decay-based regularization strategy that is later used during training to improve generalization (described in Section III.F).

In addition to the previously-discussed log-space transformation of the Reynolds number, some nonlinear transformations on the angle of attack α are present and merit discussion as well. Encoding of the angle of attack into a purely-trigonometric representation embeds the periodic nature of the problem into the model architecture. For example, model evaluation at $\alpha = 0°$ and $\alpha = 360°$ are structurally identical, as these are encoded to the same location in the input latent space. The $\sin(2\alpha)$ and $\sin^2\alpha$ terms are directly proportional to common post-stall analytical models of lift and drag, respectively (such as those by Hoerner and Truong [**? ?**]). The goal of this representation is to improve generalization performance in massively-separated flow conditions, where training data is scarce.

The freestream Mach number is a notable omission from the input latent space, and the learned model is both trained and evaluated on the equivalent incompressible flow. A compressibility correction is then performed after the neural network evaluation, as described in Section III.D.1. This dimensionality reduction was performed to keep the required amount of training data more manageable, and because analytical models can accurately and quickly perform this compressibility correction (up to the critical Mach number).

**This normalization is performed on the basis of the training data, which is described later in Section III.E.

2. Learned Model Architecture

After encoding inputs into an input latent space, NeuralFoil processes these inputs through a feedforward neural network (i.e., a multilayer perceptron). NeuralFoil offers eight different deep neural network models, which offer a tradeoff between accuracy and computational cost. This tradeoff is implemented as differences in the number and size of hidden layers, which are detailed in Table 2 for each model.

Table 2 Neural network model sizes offered in NeuralFoil, offering a trade between accuracy and speed.

Model	Hidden Layers	Neurons per Hidden Layer (width)
"xxsmall"	1	48
"xsmall"	2	48
"small"	2	64
"medium"	3	64
"large"	3	128
"xlarge"	4	128
"xxlarge"	4	256
"xxxlarge"	5	512

Each layer is a fully-connected linear layer. The activation function applied between each layer is the Swish function[††], defined as $\sigma(x) = x/(1 + e^{-x})$. Compared to typical machine learning scenarios, the choice of activation function here has surprising importance, due to the desired properties of the resulting model. The Swish activation function is smooth (C^∞-continuous), which makes the resulting neural network a smooth function as well. By preserving this continuity, NeuralFoil is made much more amenable to later gradient-based design optimization.

Likewise, the asymptotic behavior of the Swish function has key implications. Networks with activation functions that asymptote to piecewise-linear functions with differing positive and negative slopes (e.g., Swish, ReLU, LeakyReLU, Softplus) have fundamentally different extrapolation properties than networks with activation functions that asymptote to a constant (e.g., Sigmoid, Tanh). The former creates networks that asymptote to locally-affine functions, while the latter creates networks that asymptote to locally-constant functions. This observation about network extrapolation properties, discussed in greater detail by Xu et al. [**?**], forms part of the justification for the latent space encodings described in the previous section where affine extrapolation is physically-consistent. Other activation functions were also considered. For example, the Softplus activation function meets many of the aforementioned criteria, but here it

[††]sometimes written with an optional parameter β, which we set as $\beta = 1$

was abandoned in favor of Swish because networks with the latter achieved slightly better generalization performance.

Considerations during training of this network are discussed in Section III.F.

3. Embedding of Angle of Attack Symmetry

A key step during neural network evaluation (applied both during training and inference) is the embedding of physics symmetry with respect to angle of attack. The rationale behind this symmetry can be intuitively explained with the aid of the illustrations in Figure 5. Here, we consider a generic cambered airfoil at some angle of attack, as well as its "image scenario" consisting of the flipped airfoil at a flipped angle of attack. The flow physics in these two scenarios should be exactly equal and opposite, which will not generally be the case unless this is enforced. To enforce this symmetry, NeuralFoil computes the neural network output for both the original and flipped airfoil, and then merges the results. Outputs that are physically-guaranteed to exhibit even symmetry (e.g., C_D) are averaged, while outputs that exhibit odd symmetry (e.g., C_L, C_M) are subtracted. More details about this approach to embedding symmetries and invariants into neural networks are discussed in recent work by Zhang [**?**].

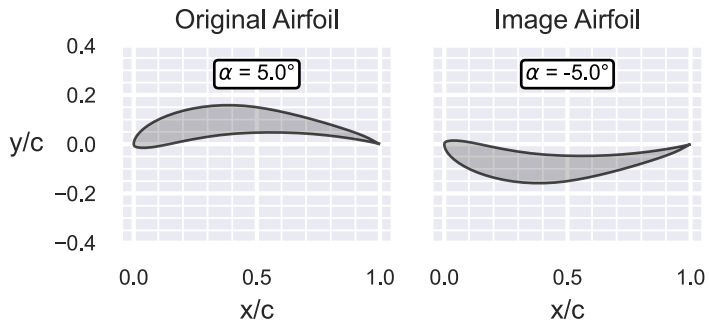


Fig. 5 Illustration of the "image" approach used to structurally embed symmetry with respect to angle of attack in NeuralFoil.

This symmetry embedding has important impacts for end-use cases of NeuralFoil. For example, when NeuralFoil analyzes a symmetric airfoil at $\alpha = 0°$, it will always yield a lift coefficient and pitching moment of *exactly* zero[‡‡], and upper- and lower-surface transition locations will also be exactly equal. This exactness would not be the case if other common methods, such as data duplication to learn physics symmetries, were used. A practical case where error in this symmetry would be particularly noticeable is on an aircraft's vertical stabilizer. Here, flows around symmetric airfoils at zero local incidence are common, and any error in enforcing this symmetry would lead to a noticeable nonzero net yaw moment on the aircraft.

[‡‡]to within machine precision, with differences only due to compensated summation algorithms during matrix multiplication

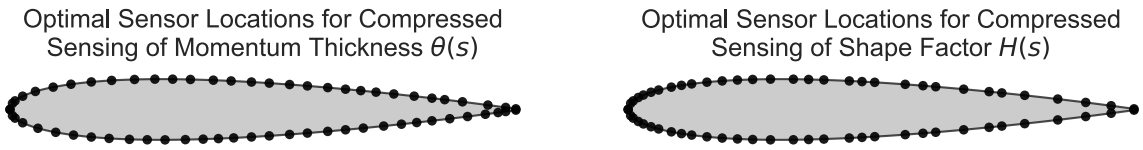
4. Decoding

The raw result of the learned model is another latent-space vector, which is then transformed back into the user-facing output space. This decoding process is broadly motivated by the same considerations as the encoding process, where the goal is to embed physical intuition into the architecture. The output latent space consists of the following vector, shown here as a function of the user-facing outputs:

$$z_{\text{out}} = \text{Affine}\left(\begin{bmatrix} \text{Analysis Confidence} \\ C_L \\ \ln(C_D) \\ C_M \\ x_{\text{tr,top}} \\ x_{\text{tr,bot}} \\ \ln(\text{Re}_\theta) \text{ at 64 locations} \\ \ln(H) \text{ at 64 locations} \\ u_e/u_\infty \text{ at 64 locations} \end{bmatrix}\right) \tag{2}$$

The output latent space has 195 dimensions, and, similar to the input latent space, undergoes an elementwise affine transformation to normalize the distribution of typical outputs. The "analysis confidence" output described in Equation 2 deserves special attention and is discussed in Section III.D.

The boundary layer outputs in Equation 2 are computed at 64 locations along the airfoil surface, which are evenly spaced in the normalized x-coordinate on the top and bottom surfaces. These data points are conceptually similar to physical sensors, as they are effectively a discrete projection of the actual boundary layer flow, which itself is a continuous function of the surface coordinate s. The spacing of these discrete "sensors" was chosen on the basis of a compressed sensing study, the results of which are shown in Figure 6. In short, this study aimed to determine "optimal sensor placement" for airfoil boundary layer data: where should one place 64 discrete sensors on an airfoil to minimize the error when reconstructing the boundary layer data? Data for this study was generated from XFoil via the same process detailed in Section III.E. Compressed sensor placement was performed using approximate QR-factorization-based methods described by de Silva et al. [9]



(a) Optimal sensor placement for minimum-error $\theta(s)$ reconstruction.

(b) Optimal sensor placement for minimum-error $H(s)$ reconstruction.

Fig. 6 Results from a compressed sensing study to determine optimal discrete locations to track boundary layer data for minimal reconstruction error.

The results in Figure 6 show that accurate reconstruction of the momentum thickness $\theta(s)$ is best achieved by roughly-uniform sensor placement. Interestingly, the optimal placement for the shape factor $H(s)$ is not uniform, but rather is concentrated near the leading edge. Since H closely relates to laminar/turbulent behavior and transition location, this result is physically intuitive: small changes in H near the leading edge have a disproportionate impact on the downstream flow.

D. Uncertainty Quantification, Model Fusion, and Extrapolation

The "analysis confidence" output detailed in Equation 2 can be loosely interpreted as a representation of NeuralFoil's self-reported model uncertainty. Quantitatively, this output is trained on binary features corresponding to whether an XFoil analysis with these input conditions converged. In effect, this is a classification problem based on whether the surrogate's underlying model returns trustworthy answers near a given point in the input space. The raw network output is a logit, which is then transformed into a confidence score in the range $(0, 1)$ using a logistic function. Example results of this "analysis confidence" metric are shown and discussed in Section IV.B.

Presenting the user with a measure of analysis confidence has several real-world benefits. First, it gives the user direct feedback on when the result of a requested analysis should be trusted. Surrogate models, especially those created with machine learning, invariably lose accuracy when extrapolating beyond the training data. This can be mitigated somewhat with embedded physics constraints, but ultimately NeuralFoil is no exception here. Furthermore, most neural surrogates give no obvious information about when this spurious extrapolation is occurring. By providing a confidence score, NeuralFoil can flag these scenarios; this may encourage a user to cross-check with a higher-fidelity model or pursue a less-uncertain design. Second, the confidence score can be used as a direct constraint during aerodynamic shape optimization driven by NeuralFoil. This enables a form of robust optimization where results are guaranteed to be within the region of the input space where the surrogate model is trustworthy.

Learning this "analysis confidence" binary classifier with no priors or physics knowledge presents a problem, however. Without such modifications, there is no guarantee that the model will extrapolate towards "untrustworthiness" (i.e., low analysis confidence) when far from the training data distribution. To enforce this behavior, the analysis

confidence logit (i.e., logarithm of odds-ratio) is modified during both training and inference. The added term is the negative squared Mahalanobis distance[§§] of the query point with respect to the training data distribution:

$$\text{Analysis Confidence} = \sigma\left(\text{(Raw Logit)} - (\vec{z}_{\text{in}} - \vec{\mu})^T S^{-1} (\vec{z}_{\text{in}} - \vec{\mu})\right) \tag{3}$$

where:
$$\begin{aligned}
\text{Analysis Confidence} &= \text{The final output of the analysis confidence model, in the range } (0, 1) \\
\sigma &= \text{The logistic function, defined as } \sigma(x) = 1/(1 + e^{-x}) \\
\vec{\mu} &= \text{The mean of the training data distribution in the input latent space} \\
\text{Raw Logit} &= \text{The raw output of the neural network, before adding the Mahalanobis distance term} \\
S &= \text{The covariance matrix of the training data distribution in the input latent space} \\
\vec{z}_{\text{in}} &= \text{The query point in the input latent space}
\end{aligned}$$

Because the covariance matrix of the training data is positive-definite, the correction term is guaranteed to asymptote to $-\infty$ as the query point moves away from the training data in any direction. By contrast, the raw logit is structurally guaranteed to extrapolate to a locally-affine function, as previous discussed in Section III.C.2. Therefore, this modification guarantees that the analysis confidence tends to zero far from the training data distribution. Because this modification is included at both train and inference time, this modification has minimal effect on the model's performance *within* the training data distribution, and only serves to embed desirable extrapolation properties when *outside* the distribution.

Another benefit of this modification is that it tends to make level sets of the analysis confidence more convex, since the modification is a quadratic form with a positive-definite Hessian[¶¶]. By regularizing the analysis confidence to be "more convex", this modification ultimately leads to better convergence in downstream gradient-based optimization problems that use the analysis confidence as a constraint.

The weighting of this Mahalanobis distance term could be varied, depending on how willing the developer of a neural surrogate is to allow the surrogate to extrapolate beyond the training distribution. In the case of NeuralFoil, a unit weighting (as shown in Equation 3) was chosen and appears to strike a good balance between extrapolation correctness and the amount of added nonlinearity. However, more broadly, this tradeoff forms an interesting area of further research in neural surrogates with self-reported trust metrics.

Based on the analysis confidence of this learned model, NeuralFoil fuses its attached-flow results with empirical models for massively-separated (post-stall) flow conditions. Specifically, NeuralFoil uses the analytical post-stall models regressed by Truong [**?**] to provide a more accurate prediction in these conditions. Example results showing this post-stall model fusion are given in Figure 7.

[§§] essentially, the Euclidian norm of the multidimensional generalization of the z-score

[¶¶] directly proportional to the inverse of the covariance matrix

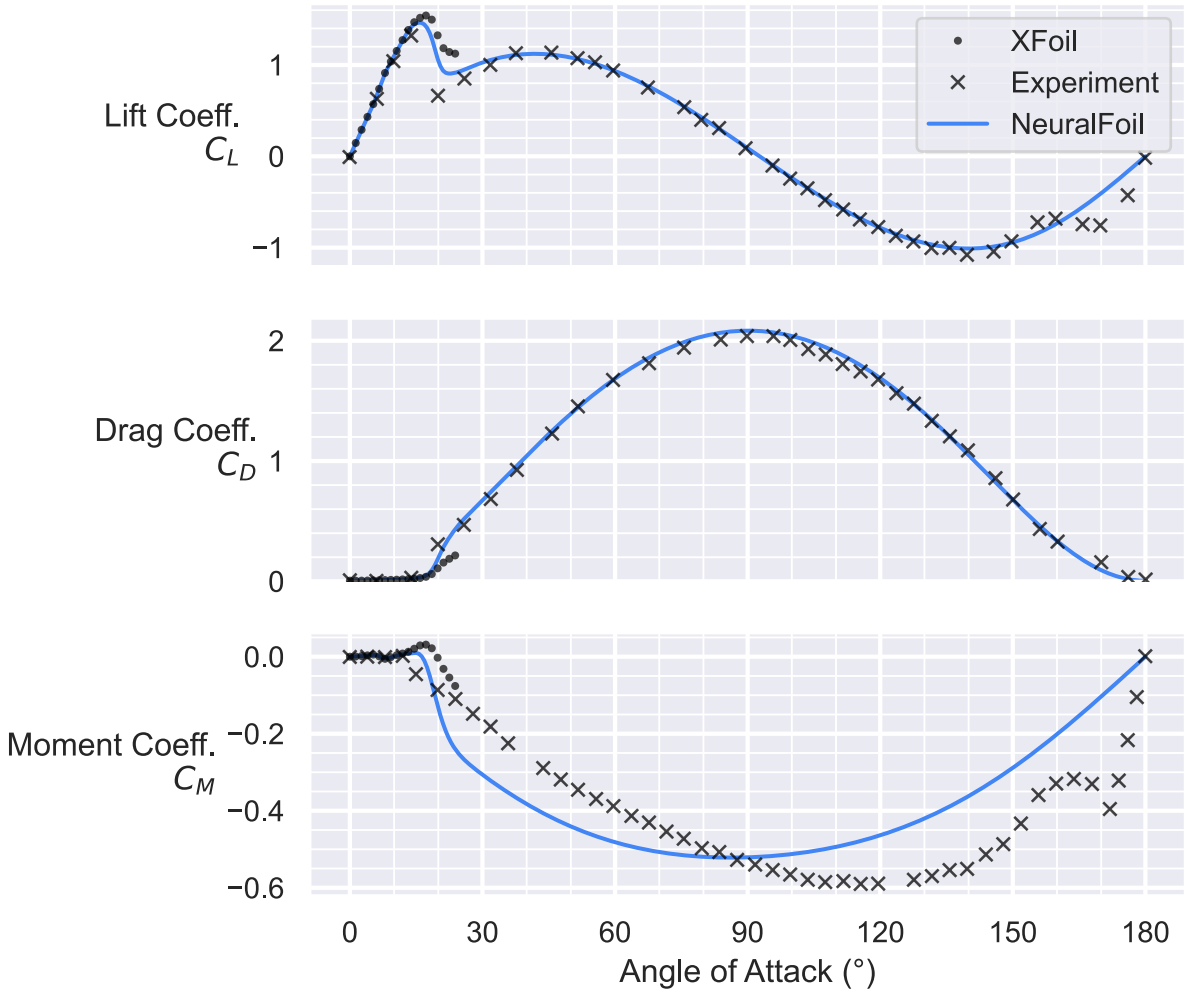


Fig. 7 Illustration of the performance of NeuralFoil in massively-separated flow conditions. NACA0012 airfoil at $\mathrm{Re_c} = 1.8 \times 10^6$. Experimental data reproduced from Langley wind tunnel data in NACA TN 3361 [?]. NeuralFoil computes post-stall lift and drag quite accurately; moment computation is somewhat less accurate.

The NeuralFoil analysis results have some notable deviations from experiment. First, moment computation is somewhat less accurate than lift and drag computation. Secondly, NeuralFoil does not capture reversed-flow reattachment near $\alpha = 180°$. Nevertheless, lift and drag results are relatively accurate, and results follow physically-sensible trends. In cases where post-stall data would be used (e.g., flight simulation, real-time control, helicopter blade stall, or recovering from a bad initial guess during design optimization), a less-accurate-but-still-reasonable answer may be useful. The model fusion shown in Figure 7 is also smooth, facilitating later gradient-based optimization.

1. Handling of Compressibility Effects

At a high level, compressibility is handled by applying a correction factor to the incompressible results. This correction factor is computed using Laitone's rule [?], which is a higher-order variant of the well-known Prandtl-

Glauert and Karman-Tsien compressibility corrections. All of these compressibility corrections take two inputs: the pressure coefficient in the equivalent incompressible flow C_{p_0} and the Mach number M; they return the pressure coefficient in the actual compressible flow C_p. For comparison, all three corrections are given here, which conveniently show the higher-order terms retained in each successive relation:

$$\text{Prandtl-Glauert:} \qquad C_p = \frac{C_{p_0}}{\beta} \tag{4}$$

$$\text{Karman-Tsien:} \qquad C_p = \frac{C_{p_0}}{\beta + M_\infty^2/(1+\beta) \cdot C_{p_0}/2} \tag{5}$$

$$\text{Laitone [?]:} \qquad C_p = \frac{C_{p_0}}{\beta + M_\infty^2/\beta \cdot C_{p_0}/2 \cdot \left(1 + \frac{\gamma-1}{2} M_\infty^2\right)} \tag{6}$$

where $\beta = \sqrt{1 - M_\infty^2}$, and γ is the ratio of specific heats (1.4 for air near standard conditions). In theory, these compressibility corrections should apply only to the pressure-derived forces on the airfoil, while the shear forces are relatively unaffected. To approximate this, NeuralFoil applies the compressibility correction to the lift force and pitching moment (which are pressure-dominated) but does not apply the correction to the drag force (which is often shear-dominated). This assumption, while relatively simple, proves to match compressible airfoil drag data computed with other methods quite closely, as shown in Section IV.E.

Another useful definition is that of the sonic pressure coefficient $C_{p_{\text{crit}}}$, which is the pressure coefficient below which flow goes supersonic. This is derived from the isentropic relations, yielding:

$$C_{p_{\text{crit}}} = \frac{2}{\gamma M_\infty^2} \left(\left(\frac{1 + \frac{\gamma-1}{2} M_\infty^2}{1 + \frac{\gamma-1}{2}} \right)^{\frac{\gamma}{\gamma-1}} - 1 \right) \tag{7}$$

At the location where supersonic flow first begins at M_{crit}, Equations 6 and 7 can be set equal. This yields a relation that maps the minimum value of the incompressible pressure coefficient $C_{p_{\text{min}}}$ to the critical Mach number M_{crit}. This relation does not admit a closed-form solution, but it can be solved numerically; this is shown in Figure 8.

This implicit mapping via a nonlinear solve is less desirable at runtime, for two main reasons. First, this method is iterative and does not have a static computational graph, complicating compatibility with automatic differentiation tools. Secondly, it is not guaranteed to converge, depending on the initial guess. Instead, this relation can be replaced with an explicit surrogate model, which was obtained using symbolic regression (via PySR [?]):

$$M_{\text{crit}} = \left(1.0083619 - C_{p_{0,\text{min}}} + \left(-0.51058894 \cdot C_{p_{0,\text{min}}}\right)^{0.6553655}\right)^{-0.5536965} \tag{8}$$

Replacing the implicit mapping of Figure 8 with the surrogate model of Equation 8 introduces negligible error,

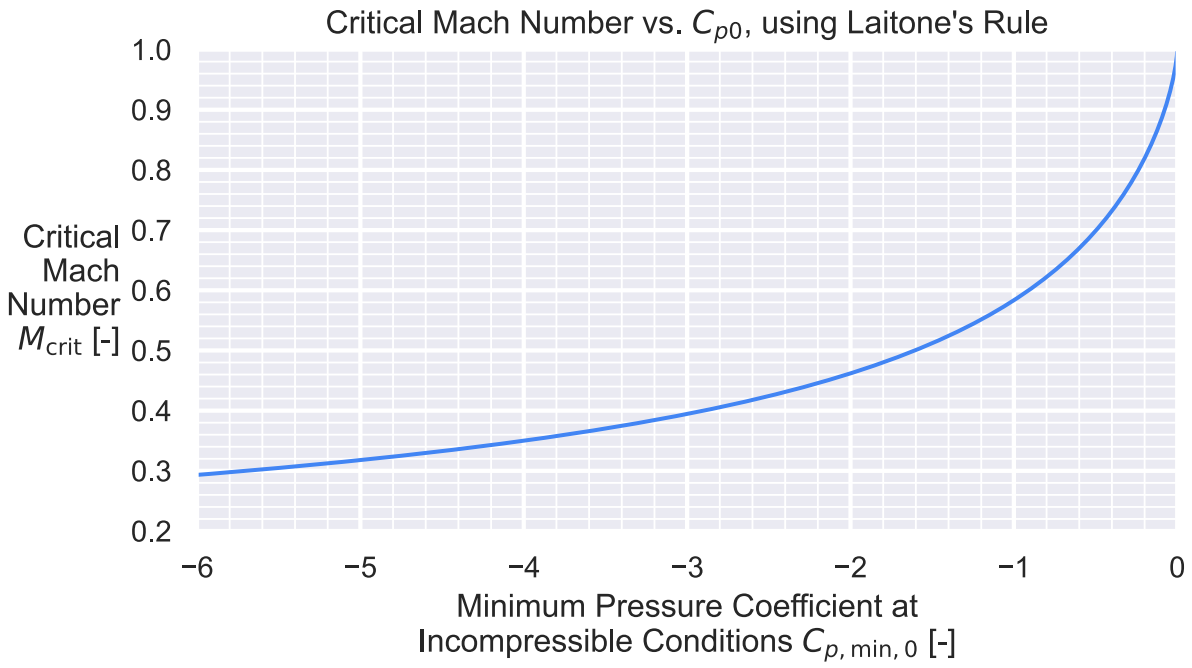


Fig. 8 Laitone's rule allows a mapping from the minimum incompressible pressure coefficient $C_{p_{0,\min}}$ to the critical Mach number M_{crit}.

with a corresponding mean RMS error in M_{crit} of 0.0014 over a representative range of $C_{p_{0,\min}}$ values[***]. This relation allows NeuralFoil to estimate the critical Mach number M_{crit} relatively accurately using only incompressible quantities, as shown later in Section IV.E and Table 5 relative to full-potential and RANS solutions. For the purposes of Equation 8, the incompressible $C_{p_{0,\min}}$ is computed definitionally from surface values of velocity magnitude, as reported by the learned model:

$$C_{p_{0,\min}} = 1 - \left(\frac{u_{\max}}{u_\infty}\right)^2 \qquad \text{at } M_\infty = 0 \tag{9}$$

Following a derivation from Mason [**?**], an empirical relation for the shape of the drag rise beyond M_{crit} is included. Drag in this transonic regime, especially beyond the drag-divergent Mach number M_{dd}, is relatively simple and typically errs on the side of over-estimating wave drag. Thus, NeuralFoil's predictions of *when* transonic flow will occur are reasonably trustworthy, but wave drag results deep within this transonic regime are not. However, given that a primary goal of the NeuralFoil tool is to drive design optimization, this simple empirical model achieves its purpose of steering an optimizer away from thick transonic airfoils with strong shocks.

[***]More precisely: finding the mean error with respect to Mach in the interval $M \in [0.001, 0.999]$.

E. Training Data Generation

Synthetic training data was generated by running XFoil on randomly-generated airfoil shapes and flow conditions. All training data is analyzed without compressibility in XFoil (i.e., $M_\infty = 0$); compressible effects were handled outside of the neural network training process, as described in Section III.D.1. The stochastic procedure used to generate the airfoil shapes used in training can be described as follows:

1) First, three parent airfoils are randomly selected from an airfoil database[†††], and converted to the Kulfan geometry parameterization. Three random weights are drawn, and the airfoils are merged into one based on these weights. This procedure effectively ensures that each training airfoil is derived from a unique combination of three parent airfoils.
2) To increase the diversity of training data, several further modifications are applied:
    - The thickness is randomly scaled by a factor drawn from Lognormal($\mu = 0,\ \sigma = 0.15$).
    - The airfoil's Kulfan parameters are perturbed by a random vector drawn from a multivariate normal distribution. The mean and covariance of this distribution are taken from the sample statistics of the airfoil database.
3) Flow conditions are randomly selected. Angle of attack α is drawn from the sum of a uniform and normal distribution, and Reynolds number $\mathrm{Re_c}$ is drawn from a log-normal distribution. Distributional statistics for these variables and transition criteria are given in Table 3.

Table 3 Summary statistics of flow conditions in the overall dataset, which is later partitioned into separate training and test datasets. Percentages refer to percentiles of the distribution. Strictly speaking, this table gives summary statistics for the subset of generated cases that were later associated with a successfully-converged XFoil run, so the true distribution of the generator is slightly wider. Suffixes 'k', 'M', and 'T' denote 10^3, 10^6, and 10^{12}, respectively.

	Minimum	**2.5%**	**25%**	**50% (Median)**	**75%**	**97.5%**	**Maximum**
Angle of attack α	−27.9°	−17.3°	−7.5°	+0.9°	+8.7°	+17.6°	+28.6°
Reynolds number $\mathrm{Re_c}$	0.916	1.87k	31.1k	296k	3.47M	262M	2.92T
Critical amplification factor N_{crit}	Uniformly distributed in $[0, 18]$.						
Transition locations $x_{\mathrm{tr,forced}}/c$	With 80% probability, natural transition. Otherwise, uniform in $[0, 1]$.						

In total, 7,913,292 data points (i.e., airfoils and flow conditions) were generated with this procedure and analyzed with XFoil. Cases that did not result in converged XFoil solutions were not used to train physics outputs; however, this non-convergence was still recorded as a binary feature to train the analysis confidence output described in Section III.D. Overall, 56% of generated cases resulted in converged XFoil solutions, which suggests that this data-generating

[†††]This database consists of 2,174 airfoils, which are drawn from a variety of sources (most notably, the UIUC airfoil database [10] and TraCFoil database [**?**]) and manually cleaned. The database is publicly available via AeroSandbox [**?**].

process does a reasonable job of spanning the space of inputs where XFoil convergence is likely. XFoil results that were purportedly converged but violated physical constraints (e.g., $\theta < 0$, $H < 1$, non-physical u_e values) were also automatically filtered and treated as unconverged, though this was rare and affected only 0.03% of cases. The distribution of training data points was sufficiently diverse that converged XFoil results were obtained with lift coefficients ranging from -2.67 to $+3.44$.

It is not currently known whether the number of points in this dataset is too little, too much, or appropriate. As a point of comparison, the number of data points used to train NeuralFoil is roughly two orders of magnitude larger than that of similar studies (see Table 6), although NeuralFoil's data distribution is substantially wider, and hence this may be warranted. In total, data generation required roughly 24 hours on MIT Supercloud, a computing cluster operated by the Lincoln Laboratory Supercomputing Center. It is possible either that similar performance could be achievable with much less data, or that more data could further improve performance. This is left as an area of future research.

F. Training Process

The neural networks at the heart of this approach were trained with MIT Supercloud's GPU computing capabilities. The training process was implemented using the PyTorch [**?**] framework. The RAdam optimizer [**?**] was used with a decaying learning rate scheduled based on the plateau of the training loss.

Synthetic data generated using the procedure described in Section III.E was split into a training dataset (95%) and a test dataset (5%), with the latter used only to evaluate generalization accuracy. Critically, a small amount of weight decay (effectively, a L^2-norm penalty on all weight and bias parameters) was added to the loss function, which improves generalization performance and causes the network training to asymptote without over-fitting. Other techniques to improve generalization, such as batch normalization and dropout, were tested; however, it was found that weight decay alone consistently produced the models with the most accurate generalization to the test dataset. Various hyperparameters associated with training the model were optimized via a parallelized grid search. Because the weight decay regularization effectively limits the amount of overfitting that is possible, all models were trained until the test-set loss reached an asymptote.

The loss function used when training the network is a Huber loss metric on each of the latent-space outputs shown in Equation 2. For this Huber loss, the L^1-to-L^2 transition point is at $\delta = 0.05$ (recalling that one unit in the output latent space corresponds roughly to the training data's output standard deviation). The exception to this Huber loss metric is the analysis confidence output, which is instead converted to a binary cross-entropy loss. Exact training details are publicly-available as described in Section VI.

Individual components of the loss function are weighted differently, to reflect differing importance during typical analysis and optimization. The highest weight is applied to drag, followed in descending order by lift, moment, and transition locations. The relative weightings of such parameters were optimized as a hyperparameter, with the goal

of minimizing generalization error (i.e., test-set performance) of drag estimation. Interestingly, this error appears minimized not by tilting the weighting entirely towards drag, but by also adding small but significant weights to other outputs, such as the detailed boundary layer. Stated more informally, this suggests that the best way to compute drag is to learn a general model that estimates all physical outputs, rather than a model focused purely on drag. This suggests that the model learns deeper physics relationships than simple memorization, as the presence of intermediate representations results in improved generalization performance on bulk quantities.

IV. Results and Discussion

A. Point Validation of NeuralFoil Accuracy with respect to XFoil

Qualitatively, NeuralFoil tracks XFoil very closely across a wide range of angles of attack and Reynolds numbers. In Figure 9, we compare the performance of NeuralFoil to XFoil on aerodynamic polar prediction. This study aims to evaluate the generalization performance: to what extent is the model learning physics vs. merely memorizing its training data? This generalization performance forms an important metric of usefulness for real-world design problems.

The airfoil analyzed here was developed separately for a real-world aircraft development program (*Dawn One*, a high-altitude long-endurance aircraft [**?** **?**]), using an inverse design process. This airfoil was not included in the training data, either directly or within the "parent airfoil" database described in Section III.E. Because of this, this represents a previously-unseen airfoil for NeuralFoil, so no unfair advantage is gained by memorization. The airfoil geometry itself is also shown in Figure 9.

In this study, aerodynamic performance is assessed across a range of Reynolds numbers spanning four orders of magnitude. The amplification factor is set as $N_{\text{crit}} = 9$ and natural transition is used. All cases are run in the incompressible limit ($M_\infty = 0$).

Figure 9 shows that excellent agreement is achieved between NeuralFoil and XFoil. The results are nearly exact for the "xxxlarge" model, which is the most accurate and computationally expensive model included with NeuralFoil. The "medium" model achieves slightly reduced accuracy while offering considerable speed improvements. Table 4.

The Reynolds numbers shown in Figure 9 were chosen to illustrate accuracy in a challenging flow condition that occurs near $\text{Re} = 80 \times 10^3$ for this airfoil. Here, the upper-surface boundary layer becomes extremely delicate, as illustrated by the sudden, discontinuous jump near $C_L = 1.0$. At angles of attack below this jump, the boundary layer simply undergoes laminar separation and never reattaches. At angles of attack above this jump, a laminar separation bubble (LSB) is formed, which undergoes turbulent reattachment before eventually separating again farther downstream. This specific effect only occurs in a narrow window of airfoil shapes, Re_c, and N_{crit} near the values shown in Figure 9, which forces the network to rely on learned physics. Nevertheless, this phenomenon is well-captured by the "xxxlarge" NeuralFoil model, demonstrating its generalization performance. Another benefit of NeuralFoil shown in Figure 9 is

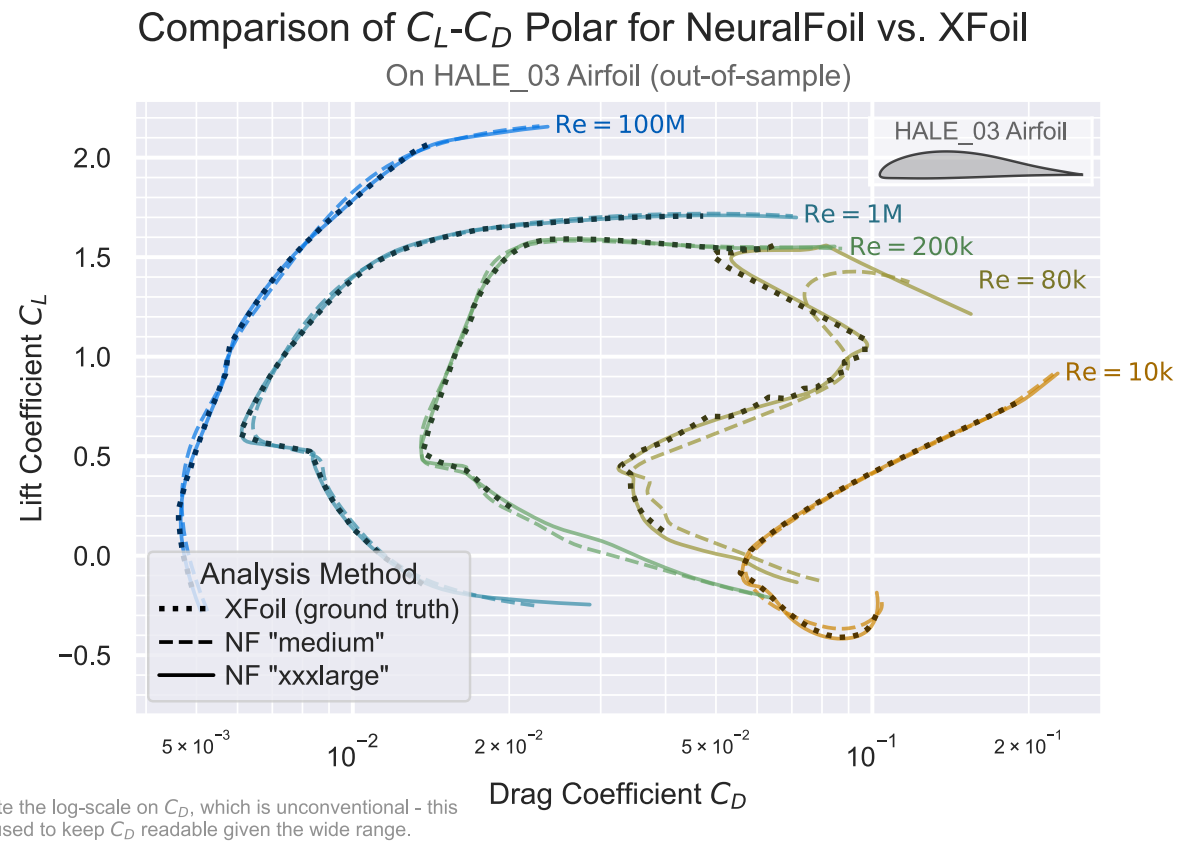


Fig. 9 Generalization performance of NeuralFoil, assessed by sample validation with repsect to XFoil on an out-of-distribution airfoil. Each colored line represents an analysis at a different Reynolds number. Results are for incompressible, viscous flow with $N_{\text{crit}} = 9$ and natural transition. Solid lines represent NeuralFoil ("NF") analyses, which are given for two models with different accuracy-speed tradeoffs. The dotted line represents XFoil results at the respective Reynolds number. Agreement with XFoil is almost exact for the "xxxlarge" model, across a broad range of angles of attack and Reynolds numbers.

the inherent C^∞-continuity of its solutions, which smooths out XFoil's "jagged" predictions in regions such as the aforementioned discontinuity.

B. Self-Reported Uncertainty Quantification

As discussed in Section III.D, NeuralFoil self-reports an analysis confidence metric that is intended to aid the user in assessing surrogate trustworthiness. This metric is shown in Figure 10 for the same airfoil and flow conditions as in Figure 9. Notably, regions with uncertain flow features are immediately flagged. This includes obvious regions of uncertainty, such as post-stall, but also more subtle ones, such as the region near $\text{Re}_c = 80 \times 10^3$ and $C_L = 1.0$. Another subtle example is a small region in the $\text{Re}_c = 100 \times 10^6$ case near $C_L = 1.0$, where movement of the stagnation point causes sudden changes in the bottom-surface transition location, and thus greater uncertainty.

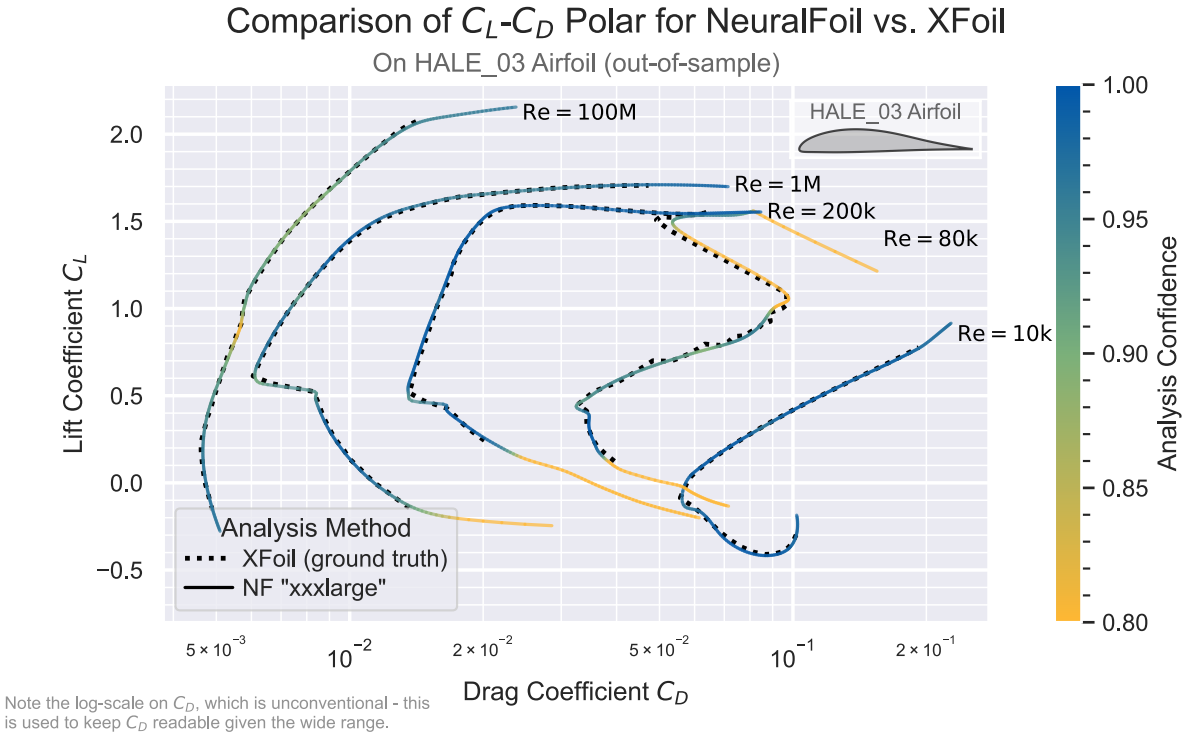


Fig. 10 Demonstration of NeuralFoil's self-reported "analysis confidence" metric, shown in an identical analysis setup as Figure 9. In this figure, color represents the value of the analysis confidence metric, which varies throughout the input space.

C. NeuralFoil Accuracy on the Test Dataset

To report the accuracy of NeuralFoil on a broader range of airfoil shapes and flow conditions, we can measure performance on the test dataset. This dataset was generated using the same methods described in Section III.E, but was not used during training. In total, this test dataset consists of 395,665 cases.

The analysis accuracy on this test dataset is shown in Table 4. At a basic level, the figures of merit are accuracy (here, treating XFoil as a "ground truth") and computational speed. This table details both considerations. The first set of columns shows the error with respect to XFoil on the test dataset. The second set of columns gives the runtime speed of the models, both for a single analysis and for a large batch analysis. Here, the benefit of NeuralFoil's vectorization is shown: for large batch analyses, runtimes many orders of magnitude faster than XFoil are possible, with minimal loss in accuracy.

Table 4 Performance comparison of NeuralFoil ("NF") physics-informed machine learning models versus XFoil in terms of accuracy (treating XFoil as a ground truth) and speed, as measured on the test dataset of Section III.E. Runtime speeds are measured on an AMD Ryzen 7 5800H laptop CPU.

Aerodynamics Model	Mean Absolute Error (L_1-norm) of Given Metric on the Test Dataset, with respect to XFoil				Computational Cost to Run	
	Lift Coeff. C_L	Fractional Drag Coeff. $\ln(C_D)$ [†]	Moment Coeff. C_M	Transition Locations x_{tr}/c	Runtime (1 run)	Total Runtime (100,000 runs)
NF "xxsmall"	0.041	0.079	0.0072	0.038	4 ms	0.94 sec
NF "xsmall"	0.030	0.057	0.0053	0.027	5 ms	0.96 sec
NF "small"	0.027	0.049	0.0046	0.023	5 ms	1.14 sec
NF "medium"	0.020	0.039	0.0034	0.017	6 ms	1.34 sec
NF "large"	0.016	0.028	0.0025	0.011	9 ms	2.35 sec
NF "xlarge"	0.014	0.025	0.0022	0.009	11 ms	2.77 sec
NF "xxlarge"	0.012	0.021	0.0018	0.007	18 ms	4.81 sec
NF "xxxlarge"	0.011	0.020	0.0016	0.005	61 ms	12.42 sec
XFoil	0	0	0	0	73 ms	2,553 sec

[†] The deviation of $\ln(C_D)$ can be thought of as "the typical relative error in C_D". (E.g., $0.020 \rightarrow 2.0\%$ error.)

D. Accuracy-Speed Tradeoff vs. XFoil

When developing machine learning surrogate models based on conventional physics solvers, far too often in the literature a speedup is claimed without controlling for accuracy. After all, it is typically trivial to make the conventional physics solver faster by sacrificing accuracy (e.g., by changing the level of discretization). Therefore, any fair and meaningful speed comparison between a machine learning surrogate and a conventional solver must control for accuracy.

Figure 11 gives this speed-accuracy tradeoff for NeuralFoil, compared to the most common conventional alternative, XFoil. For NeuralFoil, this tradeoff is controlled by the model size (as described in Table 2). For XFoil, this tradeoff is controlled by the number of points used to discretize the airfoil, which was varied from 20 to 260 in Figure 11.

In this study, each model is evaluated on a range of airfoils and flow conditions. To minimize issues with non-convergence in XFoil, a new range of aerodynamic cases is used. The airfoil shapes are NACA 4-series airfoils with the location of maximum camber fixed at $x/c = 0.4$. Thickness is varied in the range $[0.08, 0.16]$ and maximum

camber is varied in $[0.00, 0.06]$; both are independently varied in increments of 0.01. This creates a total of $9 \times 7 = 63$ airfoil shapes. Each airfoil is evaluated at three Reynolds numbers: 500×10^3, 2×10^6, and 8×10^6. This gives a total of 189 aerodynamic cases. In all cases, $\alpha = 5°$, $M_\infty = 0$, and $N_{\text{crit}} = 9$. This selection of aerodynamic cases is relatively "easy" and is deliberately chosen to give a runtime speed advantage to XFoil. This is because XFoil, as an iterative algorithm, tends to have an increasing computational cost as flow cases become more difficult. By contrast, NeuralFoil has a fixed computational cost per analysis. Overall, 95% of cases converged with XFoil, averaged over all discretization levels.

The "ground truth" reference solutions for this study are taken from a high-resolution XFoil simulation with 289 points, which is the highest resolution that XFoil 6.98 allows before it begins silently truncating wake points[‡‡‡].

In Figure 11, the x-axis shows the accuracy of each model, measured as the mean relative error of the drag coefficient[§§§] across all 189 cases. The y-axis shows the runtime speed, which is the median time to run each of the 189 cases. By using the median runtime, XFoil again gains a runtime speed advantage, as slow convergence creates a strong positive skew in the distribution of XFoil's runtimes.

As shown in Figure 11, all of NeuralFoil's model sizes give a speedup over XFoil, even when controlling for accuracy. With naïve looping over all 189 cases, NeuralFoil achieves the same accuracy level as XFoil at speeds more than 8x faster. However, NeuralFoil really shines when taking advantage of its vectorization and analyzing all 189 cases simultaneously. Here, speedups can be nearly 1,000x at the same accuracy level.

[‡‡‡]This is due to a fixed-size array that XFoil allocates named `IDX`, which is truncated to avoid overflowing.
[§§§]Or, equivalently, the mean absolute error of $\ln(C_D)$

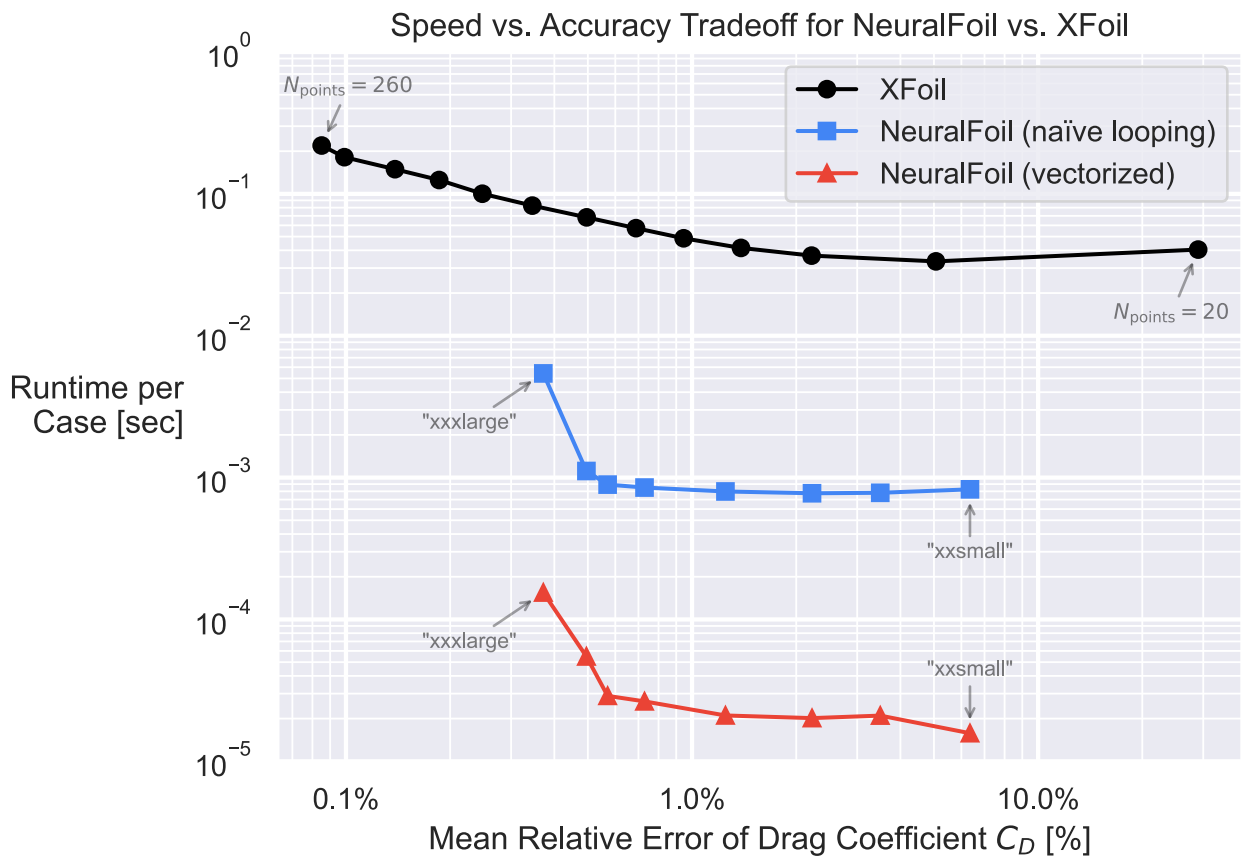


Fig. 11 Comparison of runtime speed for NeuralFoil and XFoil, while controlling for accuracy. Evaluated on a varied database of NACA airfoils, with ground truth from XFoil at its highest allowable resolution. XFoil runs are varying resolution; NeuralFoil runs are varying model sizes. Speeds are measured on an AMD Ryzen 7 5800H laptop CPU.

Several other notable observations can be made from Figure 11. First, NeuralFoil's accuracy on "easy" cases appears far better than Table 4 suggests. This is because the test dataset in Table 4 includes many post-stall and transition-sensitive cases, as described in Section III.E. For example, the mean relative error of drag for the "xxxlarge" model is 2.0% on the test dataset, while the same metric on this easier dataset is 0.37%. This reinforces that, when comparing accuracy across studies, the difficulty of the evaluation cases must be considered.

Secondly, NeuralFoil achieves a "knee" in the speed-accuracy tradeoff curve roughly near the "xlarge" model size. This is therefore taken as the default model size for NeuralFoil, though different use cases may encourage other choices.

E. Sample Validation on Transonic Case

To demonstrate the effect of compressibility on NeuralFoil's aerodynamics estimates, we can compare NeuralFoil to various other approaches in a transonic case study. This case study analyzes a RAE2822 supercritical airfoil at flow conditions of $\mathrm{Re_c} = 6.5 \times 10^6$ and $\alpha = 1°$, with natural transition and $N_{\mathrm{crit}} = 9$. The freestream Mach number is varied from subsonic to transonic conditions. The results of this study are shown in Figure 12. Here, NeuralFoil's results are

compared to those from four existing approaches:

- XFoil 6.98 [**?**]: This uses a boundary-element potential flow solver with a Karman-Tsien compressibility correction for the outer flow, and an integral boundary layer (IBL) model for the boundary layer. Transition is handled with an e^N method.
- XFoil 7.02: This uses a grid-based full-potential solver for the outer flow; compared to a linearized potential flow solver, this has stronger theoretical grounding for transonic analysis and the ability to directly capture shock waves. The boundary layer and transition models are the same as those in XFoil 6.98.
- MSES [**?**]: This is a hybrid Euler / Full-Potential solver that has yet-stronger theoretical grounding as shocks become stronger. The boundary layer and transition models are similar to those in XFoil 6.98, though viscous-inviscid coupling is achieved via a displacement-body approach rather than a wall-transpiration approach. This model is believed to be the most accurate of those listed here for transonic flows with weak shocks.
- SU2 [**?**]: This is a RANS solver with a Spalart-Allmaras turbulence model [**?**]. Transition is handled using a Langtry-Menter (LM) correlation-based model [**?**], using the Malan correlation [**?**]. Freestream turbulence intensity is set to 0.07%, which approximately corresponds to $N_{\text{crit}} = 9$ following equations from Drela [**?**]. However, this is not a direct equivalence, and the LM model predicts slightly earlier transition than e^N-based methods in this case, causing higher viscous drag.

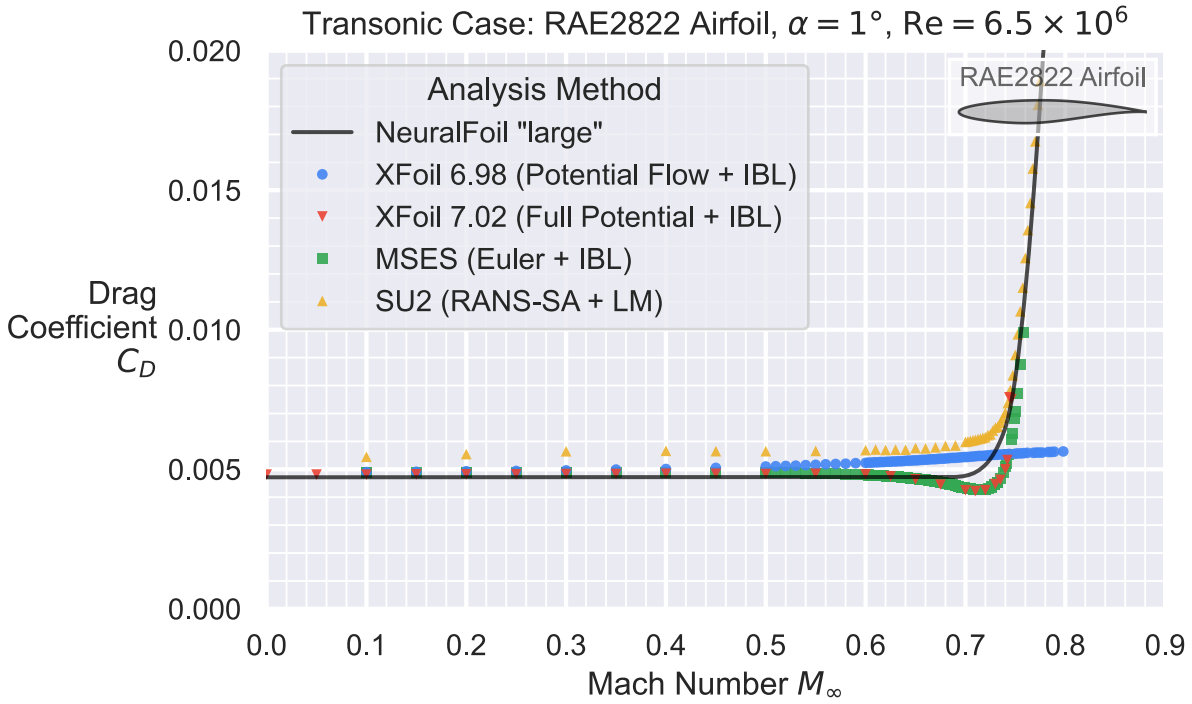


Fig. 12 Validation of NeuralFoil at predicting the emergence of transonic flow, using the RAE2822 airfoil at $\mathrm{Re_c} = 6.5 \times 10^6$ and $\alpha = 1°$. NeuralFoil's results are compared to those from XFoil 6.98, XFoil 7.02, MSES, and SU2. NeuralFoil's results are shown for the "large" model described in Table 2.

NeuralFoil's results in Figure 12 are qualitatively similar to those obtained with other methods, particularly with respect to the location and steepness of the transonic drag rise. Some differences remain, however. Notably, models that use both an integral-boundary-layer approach and a higher-fidelity transonic treatment (i.e., XFoil 7.02, MSES) capture a slight decrease in drag in a window between the critical and drag-divergent Mach numbers. This effect is believed to be real, and it occurs because the locally-supersonic flow in this regime causes a favorable pressure gradient that delays the top-surface boundary layer transition. Such an effect is not captured by NeuralFoil, SU2, or XFoil 6.98, as capturing this effect requires the combination of both shock-resolution and advanced transition modeling.

The results of these analyses can be quantitatively compared in Table 5, which shows the critical and drag-divergent Mach numbers predicted by each method. Corroborating the values in this table, Drela cites the critical Mach number for this case as $M_{\mathrm{crit}} = 0.71$ [**?**]. NeuralFoil appears to underestimate the critical Mach number slightly in this case, although the drag-divergent Mach number agrees very closely with other methods.

Table 5 Predictions of critical and drag-divergent Mach numbers for the RAE2822 airfoil at $\mathrm{Re_c} = 6.5 \times 10^6$ and $\alpha = 1°$, using various methods.

	Critical Mach Number M_{crit}	Drag-Divergent Mach Number M_{dd} [†]
NeuralFoil "large"	0.671	0.739
NeuralFoil "xxxlarge"	0.673	0.741
XFoil 6.98 (Potential Flow + IBL)	0.688	N/A
XFoil 7.02 (Full Potential + IBL)	0.705	0.739
MSES (Euler + IBL)	0.714	0.738
SU2 (RANS-SA + LM)	-	0.738

[†] Defined as $\partial(C_D)/\partial M_\infty \geq 0.1$, following Mason [**?**].

F. Comparison to other Airfoil Machine Learning Models

Here, we compare and contrast NeuralFoil with several prior attempts to apply machine learning techniques to airfoil aerodynamics analysis. Peng et al. [**?**] apply a convolutional neural network to this task, where the input is an array of airfoil coordinates and the output is a scalar lift coefficient. Bouhlel et al. [**?**] use feedforward neural networks to predict airfoil lift and drag coefficients; this is augmented by Sobolev regularization that adds gradient information into the loss function. Du et al. [**?**] use multilayer perceptrons, recurrent neural networks, and mixture-of-experts approaches to predict both scalar (lift and drag) and vector (pressure distribution) quantities.

Compared to prior work, NeuralFoil is trained on a significantly broader input space, particularly with respect to angle of attack and transition assumptions. This comparison is illustrated in Table 6. The accuracy values listed in Table 6 are not directly comparable, as they are computed on different datasets and with different metrics. Nevertheless, NeuralFoil achieves similar accuracy to prior work, despite including many extreme cases in its evaluation sample (e.g., post-stall flows, transitional Reynolds numbers, and high-curvature airfoils associated with 18 shape variables). This is possible partially because of the much larger volume of training data, but also due to the structural embedding of physics knowledge into the model described throughout Section III.

Accurate and rapid aerodynamics analysis across this broader input space has the potential to enable airfoil design optimization without overly restricting the design space. Furthermore, certain types of robust optimization become possible. For example, performing a multipoint optimization study with respect to N_{crit} becomes possible with NeuralFoil, allowing the designer to mitigate the tendency of optimizers to "over-optimize" to one specific transition

location[¶¶¶].

Table 6 Comparison of NeuralFoil to prior literature in airfoil aerodynamics machine learning. NeuralFoil achieves similar accuracy, despite including a much broader range of flow conditions in its training and evaluation distributions. Models from Du et al. [?], Bouhlel et al. [?], and Peng et al. [?] use separate subsonic and transonic models.

		NeuralFoil	Du et al. [?]	Bouhlel et al. [?]	Peng et al. [?]
Diversity of "airfoil design space" used for both training and evaluation	**Airfoil shape diversity**	18 shape variables	16 movable control points	14 shape vars. (subsonic) 8 shape vars. (transonic)	UIUC database (~1500 fixed shapes)
	Angle of attack range	Uniform + normal distrib.; central 95% in range $[-17^\circ, +18^\circ]$	0° to 3°	-0.5° to 6° (subsonic) -1.5° to 4.5° (transonic)	-2° to 10°
	Reynolds number range	Log-normal distrib.; central 95% in range $[1.9\times10^3, 262\times10^6]$	10^4 to 10^{10} (training)	None specified	13×10^6 (fixed)
	Mach number range	0 for learned core, although compressibility correction allows prediction up to M_{crit}	0.3 to 0.6 (subsonic) 0.6 to 0.7 (transonic)	0.3 to 0.6 (subsonic) 0.7 to 0.75 (transonic)	0.3 (fixed)
	Transition, turbulence, and forced trips	N_{crit} varied from 0 to 18; Trips varied from LE to TE	None specified ($\tilde{\nu} = 0$ assumed)	None specified ($\tilde{\nu} = 0$ assumed)	None specified ($N_{\text{crit}} = 9$ assumed)
	Training data	XFoil (7.5M samples)	RANS-SA via ADflow (86k samples)	RANS-SA via ADflow (42k samples)	XFoil (16k samples)
Performance, as evaluated on the corresponding distribution above	C_L **mean absolute error**	0.012	0.010	Unspecified	1.0%
	C_D **mean relative error**	2.0%	1.7%	0.3%	N/A (not attempted)

[¶¶¶]For more details on this, see Drela [**?**]. In general, airfoil shape optimization algorithms tend to geometrically "fill in" locations where laminar separation bubbles would otherwise go, which dramatically worsens off-design performance.

V. Airfoil Design Optimization with NeuralFoil

Although the accuracy of NeuralFoil with respect to XFoil has been demonstrated in Section IV, this alone is not sufficient to demonstrate that NeuralFoil is useful for design optimization. A major reason for this is that engineering design optimization is an adversarial process, where the optimizer aims to exploit not only the underlying physics, but also any errors in the model. Exploitation of the physics is desirable and ultimately the goal of engineering design optimization, but exploitation of model errors leads to designs that are ostensibly promising but lose their luster when analyzed with other computational tools or actually built and flown.

To assess whether a model such as NeuralFoil is prone to model error exploitation, we set up an aerodynamic shape optimization problem with a known answer from the literature, and evaluate how close the optimized result matches this. In aerodynamic shape optimization (and aircraft design more broadly), there are relatively few problems that are rigorously specified, and even fewer where the correct optimized result is provided in exacting detail. However, one such case study that provides a useful design problem is Drela's 1998 *Pros & Cons of Airfoil Optimization* [**?**], which discusses airfoil design for the *MIT Daedalus* human-powered aircraft [**? ? ?**]. Basic figures for this aircraft, along with its centerline airfoil, the DAE-11, are shown in Figure 13.

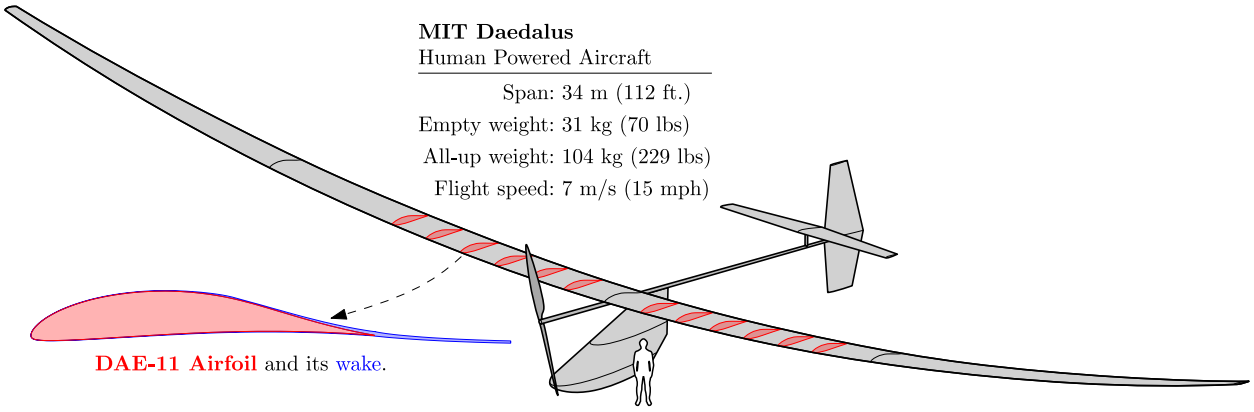


Fig. 13 Drawing of the *MIT Daedalus* human-powered aircraft, along with its centerline airfoil, the DAE-11. This airfoil is the subject of the airfoil design optimization problem posed in this section.

The design of the DAE-11 airfoil is effectively defined by the following airfoil design optimization problem; this formulation is taken directly from Drela [**?**] with minor modifications[17]:

- **Objective**: Minimize $C_{D,\text{mean}}$ at $\text{Re}_c = 500\text{k} \cdot \left(\frac{C_L}{1.25}\right)^{-0.5}$ and $\text{M}_\infty = 0.03$
  - Where $C_{D,\text{mean}}$ is the weighted average of C_D values at $C_L = [0.8, 1.0, 1.2, 1.4, 1.5, 1.6]$, with relative weights of $[5, 6, 7, 8, 9, 10]$ at each respective C_L
- **Variables**: airfoil shape (18 variables; described in Section III.B) and angle of attack α
- **Constraints**:

[17] Specifically, a constant-$\text{Re}_c\sqrt{C_L}$ (fixed-lift) polar is used instead of a constant-Re_c polar; the C_M constraint is clarified to apply to all C_L operating points; and the θ_{TE} constraint is matched to the DAE-11 airfoil

- $C_M \geq -0.133$ (at all C_L operating points)
- Trailing-edge angle $\theta_{\text{TE}} \geq 6.03^\circ$ (allows manufacturability)
- Leading-edge angle $\theta_{\text{LE}} = 180^\circ$ (prevents the formation of a sharp leading edge)
- At $x/c = 0.33$, $t/c \geq 0.128$ (to accommodate the main spar)
- At $x/c = 0.90$, $t/c \geq 0.014$ (to accommodate the rear spar)

When posing airfoil design optimization problems (using NeuralFoil or with any other tool), it is often helpful to add basic regularization constraints; these tend to make airfoil optimization better-behaved by ruling out non-physical and unrealistic airfoils. For the simple airfoil design problem described above, these regularization strategies are not necessary for convergence. However, they are discussed here as "best practices" for airfoil optimization on more complicated problems:

The first such constraint is that the airfoil thickness must be positive everywhere. This prevents the creation of self-intersecting airfoils, which can be analyzed in both XFoil and NeuralFoil but are clearly not physically-valid shapes.

The second such constraint aims to guide the optimizer away from airfoils that have excessively high local surface curvature, as these have boundary layer behavior that is inherently difficult to characterize. A useful global metric for this curvature, which we call the wiggliness w, is mathematically similar to a metric by Wahba [**?**] that has long proven successful in univariate spline regularization:

$$w(\text{airfoil}) = \int_0^1 \left(\frac{d^2}{dx^2} y_{\text{lower}}(x)\right)^2 + \left(\frac{d^2}{dx^2} y_{\text{upper}}(x)\right)^2 dx$$

We can loosely approximate the spirit of this wiggliness measure from the discrete Kulfan parameterization as follows:

$$w(\text{airfoil}) \approx \sum_{i=2}^{N-1} \left(c_{\text{lower},i+1} - 2 \cdot c_{\text{lower},i} + c_{\text{lower},i-1}\right)^2 + \left(c_{\text{upper},i+1} - 2 \cdot c_{\text{upper},i} + c_{\text{upper},i-1}\right)^2$$

where $c_{\text{lower},i}$ and $c_{\text{upper},i}$ are the ith Kulfan (CST) coefficients of the lower and upper surfaces, respectively. A reasonable heuristic for this regularization constraint is to restrict this metric to no more than four times that of the original initial guess airfoil (which is typically some generic airfoil, such as a NACA0012).

Using the problem formulation described above, we can solve this airfoil design optimization problem by coupling NeuralFoil with the AeroSandbox aircraft design optimization framework [**?**]. This adds automatic differentiation capabilities to NeuralFoil, allowing efficient optimization using gradient-based methods. This optimization problem is solved in approximately 7 seconds on a standard laptop; the speed of this solution provides rapid feedback to the designer on how to improve the optimization problem formulation to capture design intent.

Figure 14 compares three airfoil designs produced with different methodologies, each aimed at solving the *MIT Daedalus* airfoil design problem described previously:

- **NeuralFoil-optimized**, which is the result of optimizing while using NeuralFoil as the aerodynamics analysis

tool

- **XFoil-optimized**, which is the result of optimizing while using XFoil as the aerodynamics analysis tool
- **Expert-designed**, which is the original DAE-11 airfoil designed by Drela [**?**] and used on the as-flown *MIT Daedalus* aircraft

For comparison, Figure 14 also includes the initial guess airfoil that was provided to the optimization algorithms (a simple NACA0012). This showcases that the optimization process is robust to initial guesses that are relatively far from the optimal solution.

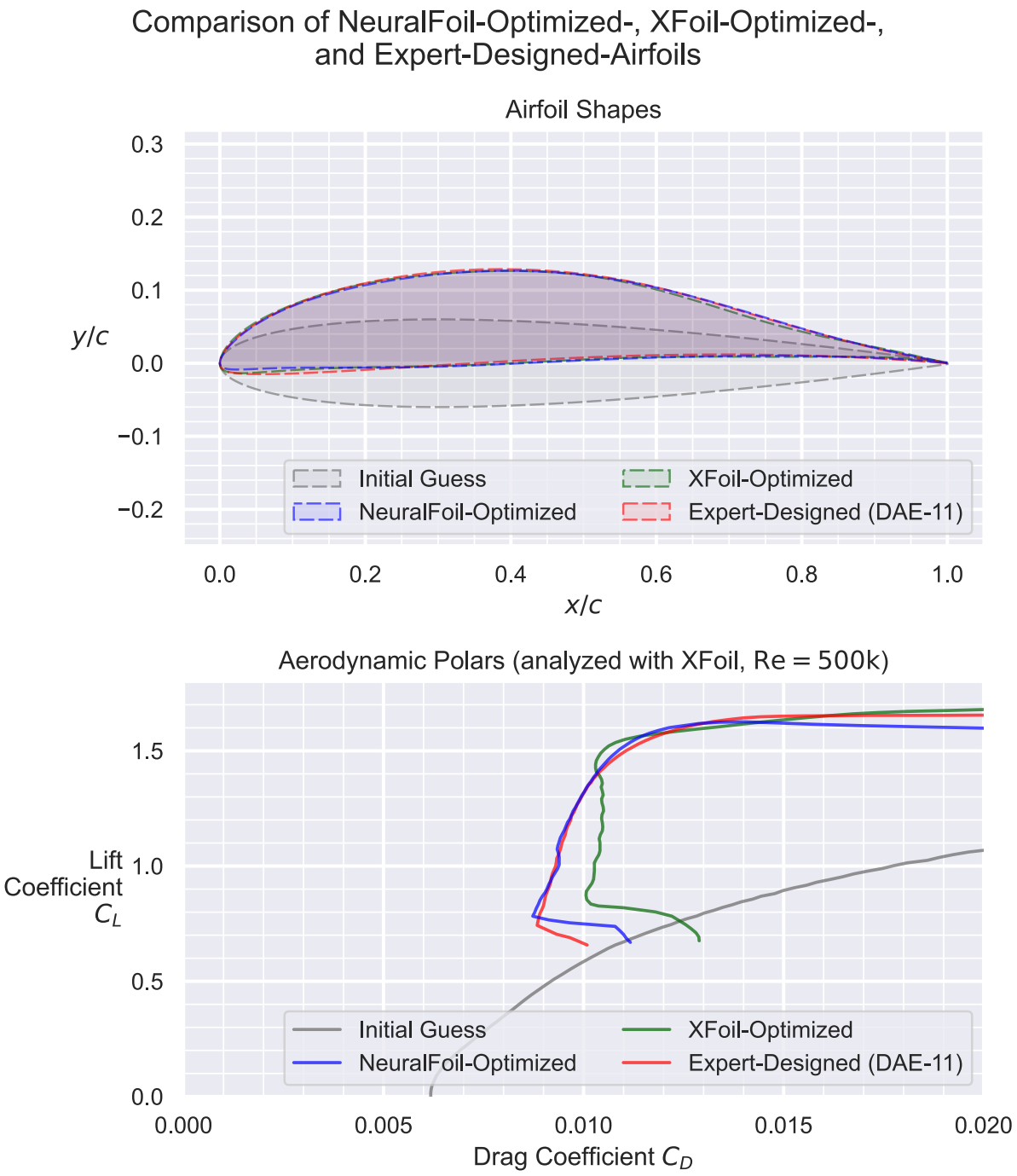


Fig. 14 NeuralFoil-optimized airfoils have similar performance and shape to expert-designed airfoils. Adversarial optimization is not observed, as the NeuralFoil-optimized and XFoil-optimized airfoils yield similar performance when analyzed using XFoil. The gray "Initial Guess" airfoil is used to initialize NeuralFoil optimization.

The airfoil produced using NeuralFoil optimization is quite similar in shape to the those produced using XFoil optimization and expert-designed airfoils, when given the same design objectives and constraints. Likewise, aero-

dynamic performance is quite similar across all three airfoils. Notably, the aerodynamic polars depicted in Figure 14 were produced by post-optimality analysis using XFoil. Given the minimal discrepancy between the NeuralFoil-optimized and XFoil-optimized airfoils, this suggests that NeuralFoil is reasonably resistant to model error exploitation on problems of practical interest.

Interestingly, the geometric differences between the expert-designed and optimized airfoils (both from XFoil and NeuralFoil) in Figure 14 are attributable to design goals that were not factored into the quantitative problem formulation. For example, the optimized airfoils both exhibit a small amount of lower-surface concavity in the vicinity of $x/c \approx 0.15$, while the expert-designed DAE-11 has a flatter lower surface. Drela discusses reasons for this discrepancy in [**?**], noting that the concavity that the optimizer prefers would cause the wing covering (in the case of *MIT Daedalus*, a thin shrunk covering of Mylar) to lift off of the surface of the rib, creating a bubble. This undesirable effect is not captured in the quantitative problem formulation, providing a cautionary tale that an optimized airfoil is only as good as the problem formulation that led to it.

Nevertheless, the strength of NeuralFoil-powered optimization is that it allows one to generate optimized airfoils that are remarkably similar to expert-designed airfoils in mere seconds. This optimization capability is more-than-adequate for conceptual aircraft design, and it provides an excellent starting point for expert-guided airfoil design refinement.

VI. Computational Reproducibility Statement

NeuralFoil is implemented as an open-source Python package with minimal dependencies (only NumPy [**?**] for the actual aerodynamic modeling), allowing easy installation across a variety of platforms.

All source code used in this paper is publicly available at `https://github.com/peterdsharpe/neuralfoil`. Model architecture, weights, training scripts, and usage examples are included in this repository. For end-users, NeuralFoil is best accessed through the open-source AeroSandbox aircraft design optimization framework [**?**]; this provides some of the advanced features (e.g., 360° angle of attack, compressibility corrections, and control surface deflections) described in this work that are not yet available in the standalone NeuralFoil package.

VII. Conclusion

In this work, we introduce NeuralFoil, a physics-informed machine learning tool for airfoil aerodynamics analysis. Relative to existing popular tools such as XFoil, NeuralFoil offers improved computational efficiency, even after controlling for equivalent accuracy. Its ability to handle a wide range of practical airfoil shapes and flow conditions makes it a versatile tool in aerodynamic analysis.

The accuracy of NeuralFoil with respect to XFoil is demonstrated across several test cases, with a particular focus on airfoils that NeuralFoil was not trained on (i.e., out-of-distribution). Much of the accuracy achieved here is due to the embedding of domain-specific physics knowledge into the model architecture, which increases the parameter-efficiency

and generalizability of the resulting model.

Also presented in this work is an application of NeuralFoil to airfoil design optimization for the *MIT Daedalus* human-powered aircraft, where it quickly produces designs close in performance and shape to expert-crafted airfoils. Incorporation of geometric constraints into this study shows that NeuralFoil-based optimization is able to capture the non-aerodynamic considerations that are typical of practical design problems. This case study also demonstrates the real-world value of many optimization-friendly features that are embedded into the model architecture (e.g., continuity, differentiability, and static code execution paths). Finally, these results also suggest that NeuralFoil is not prone to adversarial exploitation of model errors.

Another contribution of this work is a new technique that enables machine learning surrogate models to estimate their own trustworthiness, a critical capability for robust engineering design. This "analysis confidence" metric can be used directly to guide human-in-the-loop analysis, or it can be directly constrained during a formal optimization process to improve the robustness of the resulting designs.

Future work could include extending NeuralFoil to handle multi-element airfoils, which would require revisiting the geometry parameterization among other considerations. Another useful research direction would be to include compressibility corrections directly within NeuralFoil's learned model, as opposed to using an analytical correction to incompressible results. Currently, the outer flow receives a compressibility correction, but the effects of this modified pressure distribution are not then fed back into the boundary layer model. Therefore, some transonic boundary layer physics (such as an extended laminar run within a supersonic zone that causes a favorable pressure gradient) are fundamentally lost. This does, however, increase the dimensionality of the input space, and thus may affect training data requirements. Related to that, another possible next study could investigate how model accuracy changes with the amount of training data used. NeuralFoil was trained with roughly two orders of magnitude more aerodynamic cases than similar studies, although it is not yet known whether this is strictly necessary to achieve the model's performance.

As a Python-based, open-source package, NeuralFoil offers a practical and versatile solution that is accessible to both academia and industry. Its accurate and rapid performance offer value throughout the aircraft development process, though this capability is especially interesting in the early stages of aircraft development where quick feedback on design changes is crucial.

Acknowledgments

The authors acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing high-performance computing resources that have contributed to the research results reported within this work.

References

[1] Eleshaky, M. E., and Baysal, O., "Airfoil shape optimization using sensitivity analysis on viscous flow equations," 1993.

[2] Liebeck, R. H., "A class of airfoils designed for high lift in incompressible flow," *Journal of Aircraft*, Vol. 10, No. 10, 1973, pp. 610–617.

[3] Eppler, R., "Airfoil Design and Data," , 1990.

[4] Tao, T. S., "Design of a Series of Airfoils for the PSU Zephyrus Human-Powered Aircraft," , 2010.

[5] Drela, M., *A User's Guide to MSES 3.05*, Massachusetts Institute of Technology, Cambridge, MA, USA, 2007.

[6] He, X., Li, J., Mader, C. A., Yildirim, A., and Martins, J. R., "Robust aerodynamic shape optimization—from a circle to an airfoil," *Aerospace Science and Technology*, Vol. 87, 2019, pp. 48–61.

[7] Morgado, J., Vizinho, R., Silvestre, M., and Páscoa, J., "XFOIL vs CFD performance predictions for high lift low Reynolds number airfoils," *Aerospace Science and Technology*, Vol. 52, 2016, pp. 207–214.

[8] Kuzmin, A., "Non-unique transonic flows over airfoils," *Computers & Fluids*, Vol. 63, 2012, pp. 1–8.

[9] de Silva, B. M., Manohar, K., Clark, E., Brunton, B. W., Kutz, J. N., and Brunton, S. L., "PySensors: A Python package for sparse sensor placement," *Journal of Open Source Software*, Vol. 6, No. 58, 2021, p. 2828. https://doi.org/10.21105/joss.02828, URL https://doi.org/10.21105/joss.02828.

[10] Selig, M., "UIUC Airfoil Data Site," , 1996. URL https://m-selig.ae.illinois.edu/ads.html.