FuncGenFoil: Airfoil Generation and Editing Model in Function Space

Jinouwen Zhang[1*] **Junjie Ren**[1,2] **Qianhong Ma**[1,8] **Jianyu Wu**[1] **Aobo Yang**[3]
Yan Lu[1,4] **Lu Chen**[1,5] **Hairun Xie**[6,7] **Jing Wang**[6,8]
Miao Zhang[6] **Wanli Ouyang**[1,4] **Shixiang Tang**[1,4⋆]
[1] Shanghai Artificial Intelligence Laboratory [2] Fudan University
[3] Hong Kong University of Science and Technology [4] The Chinese University of Hong Kong
[5] State Key Lab of CAD&CG, Zhejiang University
[6] Shanghai Aircraft Design and Research Institute
[7] Innovation Academy for Microsatellites of CAS [8] Shanghai Jiao Tong University
[*]zhangjinouwen@pjlab.org.cn [⋆]tangshixiang@pjlab.org.cn

Abstract

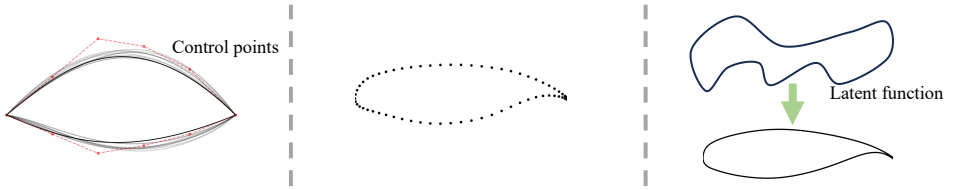
Aircraft manufacturing is the jewel in the crown of industry, in which generating high-fidelity airfoil geometries with controllable and editable representations remains a fundamental challenge. Existing deep learning methods, which typically rely on predefined parametric representations (e.g., Bézier curves) or discrete point sets, face an inherent trade-off between expressive power and resolution adaptability. To tackle this challenge, we introduce **FuncGenFoil**, a novel function-space generative model that directly reconstructs airfoil geometries as function curves. Our method inherits the advantages of arbitrary-resolution sampling and smoothness from parametric functions, as well as the strong expressiveness of discrete point-based representations. Empirical evaluations demonstrate that **FuncGenFoil** improves upon state-of-the-art methods in airfoil generation, achieving a relative **74.4%** reduction in label error and a **23.2%** increase in diversity on the AF-200K dataset. Our results highlight the advantages of function-space modeling for aerodynamic shape optimization, offering a powerful and flexible framework for high-fidelity airfoil design.

1 Introduction

The airfoil inverse design problem serves as a central aspect of aircraft manufacturing. Traditionally, given geometric requirements, engineers first select the most similar airfoils from well-known airfoil datasets (e.g., NACA [8]) and rely on a trial-and-error strategy [51]. Considering the mission of the aircraft, an initial airfoil design that meets these conditions is preliminarily created. Then, through iterative rounds of physical analyses, such as aerodynamic and mechanical evaluations, the airfoil is optimized to achieve improved performance until the specified requirements are met. In practice, such direct design procedures are highly inefficient and time-consuming, often taking months. To minimize development and design time, as well as associated costs, automated design methods have been introduced as efficient alternatives in aircraft manufacturing engineering. In particular, machine learning-based design and optimization techniques have gained significant attention. However, before applying these algorithms to airfoil design, it is crucial to determine appropriate methods for representing airfoils within these algorithms.

Existing methods for airfoil representation can generally be divided into two categories: parametric-model-based approaches [61] and discrete-point-based methods [37, 49]. First, parametric-model-based methods predefine function families, *e.g.,* Bézier curves [7], Hicks-Henne curves, and NURBS,

39th Conference on Neural Information Processing Systems (NeurIPS 2025).



(a) Airfoil as parametric curves (b) Airfoil as discrete points (c) Airfoil as function curves

Figure 1: The conceptual difference between FuncGenFoil and previous airfoil representation methods. In many previous approaches, airfoils are represented either as parametric models, as shown in (a), or as discrete point-based models, as shown in (b). In contrast, (c) illustrates FuncGenFoil's approach, where an airfoil is treated as a continuous function mapped from a latent function, enabling a generative model in function space.

and leverage mathematical optimization techniques or generative models to determine the parameters of these functions for generating a new airfoil. These methods rigorously preserve key geometric properties of the defined function families, *e.g.,* higher-order smoothness. Furthermore, such functional representations of airfoils allow arbitrary sampling of control points in real manufacturing, given engineering precision constraints. Despite these benefits, parametric-model-based methods suffer from a significantly reduced design space; *i.e.,* selecting a specific function family excludes other possible shapes, thus limiting the upper bound of airfoil design algorithms. Second, discrete-point-based methods directly generate multiple points to represent airfoil shapes. These methods maximize the airfoil design space but cannot inherently maintain certain important mathematical properties, *e.g.,* continuity. Furthermore, they cannot directly generate control points at arbitrary resolutions, as the number of generated points is typically fixed for each model after training.

To address the trade-offs between these two mainstream approaches, we ask: *Can we design an algorithm that leverages the advantages of both?*

In this paper, we address this question by proposing **FuncGenFoil**, a novel function-space generative model for airfoil representation (see Figure 1). Unlike previous data-driven generative methods in finite-dimensional spaces, *e.g.*, cVAE [25], cGAN [41], diffusion models [19], or flow matching models [35], which directly generate discrete point outputs, we leverage recent advances in diffusion and flow models defined in function space [13, 23, 32, 33, 42, 52] to produce infinite-dimensional outputs as general continuous functions. Simultaneously, our approach models airfoil geometry using Neural Operator architectures [1, 2, 27, 31], enabling the generation of diverse airfoils beyond the design spaces of predefined geometric function families, thanks to its general operator approximation capability. Our method combines the advantages of both parametric-model-based and discrete-point-based approaches. Due to its intrinsic representation in function space, the generated airfoils are continuous, and both training and sampling can be performed at arbitrary resolutions, facilitating downstream optimization and manufacturing processes.

Specifically, we use the flow matching framework [35], an improved alternative to diffusion models, and FNO [31], a resolution-free neural operator, as the backbone of our generative model to design FuncGenFoil. During training, we perturb the airfoil into a noise function through straight flows and let neural operators learn the airfoil's deconstructed velocity via flow matching. During inference, we reconstruct the airfoil by reversing the flow direction starting from a Gaussian process. Beyond generation, our method also supports airfoil editing, generating new airfoils from an original airfoil by conforming to certain geometric constraints. This can be accomplished by a few steps of fine-tuning with no additional data other than the original airfoil. Constraints can be provided in various forms, such as specifying contour points that the airfoil must pass through or setting geometric constraint parameters, such as thickness.

In summary, our main contributions are threefold: (1) We propose generating airfoil shapes in function space to achieve important properties for aircraft engineering, *i.e.,* arbitrary-resolution control point sampling and maximal design space; (2) We design FuncGenFoil, the first controllable airfoil generative model in function space, which effectively incorporates neural operator architectures into the generative modeling framework; and (3) We further enhance FuncGenFoil with airfoil editing capabilities through minimal adaptations. Experimental results indicate that our proposed method

achieves state-of-the-art airfoil generation quality, reducing label error by 74.4% and increasing diversity by 23.2% on the AF-200K dataset, as validated through aerodynamic simulation analysis. In addition, our method is the first to successfully perform airfoil editing by fixing or dragging points at arbitrary positions, achieving nearly zero MSE (less than 10^{-7}).

2 Related Works

Generative Models. Generative models based on score matching [20, 55] and flow matching [34, 36, 57] have significantly advanced machine learning, achieving state-of-the-art results in areas such as image generation [47], text generation [16], and video generation [21, 43]. However, most of these models operate in finite-dimensional spaces and rely on fixed discretizations of the data. Such formulations hinder transferability across discretizations and neglect function-level constraints (e.g., continuity, smoothness), motivating the need for generative modeling in function space.

Neural Architectures for Function Modeling. Designing neural architectures to handle function spaces remains a major research challenge. Standard networks typically assume fixed-size inputs, making them unsuitable for arbitrary resolutions. Implicit neural representations, such as SIREN [54], harness random Fourier features [44] to represent continuous and differentiable objects through position embeddings. Similarly, NeRF [40] treats input coordinates as continuous variables, offering flexible resolution for function outputs. Neural operators [1, 2, 27, 31] and Galerkin transformers [5] further generalize neural architectures to process sets of points as functional inputs, enabling function-space learning. Recently, a neural operator with localized integral and differential kernels has been introduced to improve its capability in capturing local features [38]. To handle complex and variable geometries, the point cloud neural operator has been proposed, which adaptively processes point cloud datasets [63].

Generative Models in Function Space. Early Neural Processes [14, 15] drew upon Gaussian processes [46], and later methods such as GASP [11], Functa [10], and GANO [45] treat data as function evaluations to enable discretization-independent learning. Energy-based and diffusion models [13, 23, 32, 33, 42], along with flow-based approaches like FFM [24] and OpFlow [52], further extend these ideas. Ultimately, developing comprehensive generative models in function space requires defining suitable stochastic processes, score operators, and consistent neural mappings, along with specialized training methods for numerical stability—challenges that remain largely unresolved.

Airfoil Design and Optimization. Airfoil design is essential for aircraft and wind turbines. Geometric parameterization supports efficient shape modeling and optimization. Techniques such as Free-Form Deformation (FFD) [12] and NURBS [48] are widely used in CAD/CAE tools [9, 18], but their independent control points can cause instability, high dimensionality, and suboptimal solutions. Modal methods, including Proper Orthogonal Decomposition [3], global modal [4], and compact modal [30] approaches, reduce dimensionality by encoding global features, though they struggle with large or detailed shape changes. Class-Shape Transformation (CST) [28] offers interpretable and differentiable shapes but lacks flexibility for large deformations and is sensitive to parameter choices. Therefore, a high-degree-of-freedom representation is needed for robust, nonlinear shape modeling. Unlike general computer vision tasks [26, 56], engineering AI models are domain-specific and often limited by sparse data. Generative models are advantageous here, as they can exploit unlabeled or untested samples. In airfoil design, VAEs, GANs, and diffusion models [7, 29, 37, 58, 61, 62], along with CFD-based mesh representations [59, 60], have been used to map latent spaces to airfoil shapes. However, most existing methods model discrete airfoil points, limiting flexibility for downstream applications.

3 FuncGenFoil: Function-Space Generative Model for Airfoils

In contrast to existing airfoil generation methods, FuncGenFoil is constructed as a function-space generative model capable of producing airfoil geometries as continuous functions rather than discrete points, thereby leveraging the advantages of both parametric-model-based and discrete-point-based methods. In this section, we detail the processes for airfoil generation and editing tasks, respectively.

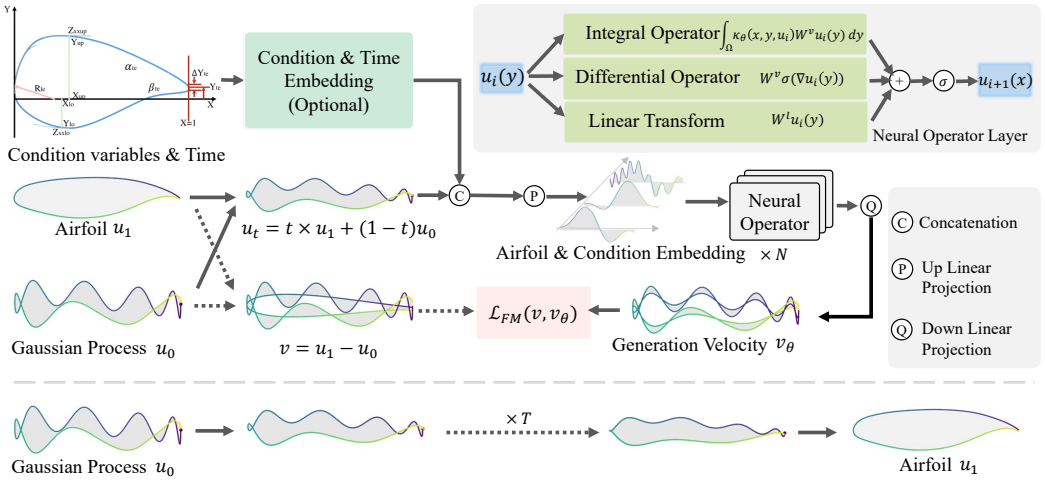


Figure 2: **Top:** Overview of **FuncGenFoil**'s neural network and training scheme. The model is a Fourier Neural Operator designed for point cloud data, although any other neural operator capable of general-purpose function-space approximation may be used. The model takes as input a function u_t (point cloud data at an arbitrary resolution d), optional design condition variables c, and the generation time t. It then processes this input function consistently and outputs the current velocity operator $v_\theta(u_t, c, t)$ for calculating the flow matching loss. **Bottom:** Inference with **FuncGenFoil** is conducted by first sampling a random latent function from a Gaussian process and then reconstructing the airfoil by solving an ODE.

3.1 Airfoil Generation

Airfoil Parametrization. Since an airfoil curve (x, y) has circular topology, we introduce the variable $\alpha \in [0, 1]$ as the domain of the function, denoting $y(\alpha) = f(x(\alpha))$ and $x(\alpha) = \frac{\cos(2\pi\alpha)+1}{2}$,[1] as shown in Figure 7. The entire FuncGenFoil framework is essentially an ordinary differential equation (ODE)-based generative model that produces airfoil curve functions by solving an ODE along continuous time $t \in [0, 1]$ as $u_1 = u_0 + \int_{t=0}^{t=1} v_t \, \mathrm{d}t$, where $u_1 = y$, starting from some latent function u_0.

Velocity Operator. The velocity operator v_t gradually transforms a latent function u_0 sampled from a stochastic process $\mathcal{P}$ into an airfoil function u_1 belonging to the target airfoil distribution $\mathcal{Q}$. The velocity operator is a key component, as it must handle function inputs consisting of point sets with arbitrary resolution or positions. Specifically, the operator takes as input the partially generated airfoil $u_t = [u_t(\alpha_0), u_t(\alpha_1), u_t(\alpha_2), \dots]$ and outputs a velocity function corresponding to these positions, $[v_t(\alpha_0), v_t(\alpha_1), v_t(\alpha_2), \dots]$. We realize this by establishing a parameterized neural operator v_θ with model weights θ, constructed as a Neural Operator model capable of processing function-space data at arbitrary resolutions in a single unified model, as shown in Figure 2. Specifically, FuncGenFoil acts as a conditional continuous-time generative model, where the velocity operator $v_\theta(u_t, c, t)$ takes the noised airfoil u_t, optional conditioning variables c, and generation timestamp t as inputs, and consistently outputs the deformation v_t as a function. We train $v_\theta(u_t, c, t)$ using Operator Flow Matching [53].

Training. We train v_θ under the simplest denoising training process. Given an airfoil geometry u_1, we compute its corresponding noised sample at time t as follows:

$$u_t|_{u_0,u_1} = t \times u_1 + (1-t) \times u_0 \tag{1}$$

and then the ground-truth velocity $v_t(u_t)|_{u_0,u_1}$ given u_0 and u_1 can be computed as:

$$v_t(u_t)|_{u_0,u_1} = \frac{du_t}{dt} = u_1 - u_0. \tag{2}$$

We can train v_θ by matching the velocity operator using Flow Matching loss:

$$\mathcal{L}_{\text{FM}} = E_{t\sim[0,1],u_t}\left[\|v_t(u_t) - v_\theta(u_t, c, t)\|^2\right], \tag{3}$$

[1] For the remainder of this paper, we omit the explicit dependence on α for clarity and convenience.

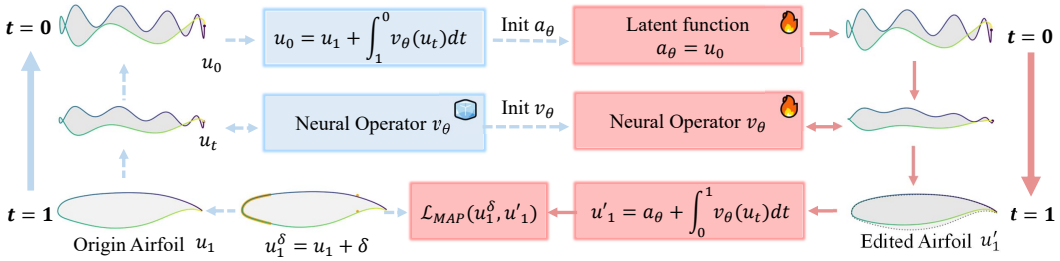


Figure 3: Airfoil editing by **FuncGenFoil**. Given a original airfoil u_1, an editing requirement δ and target airfoil u_1^δ. We first infer its latent function u_0 reversely, and make it learnable as a_θ. Then we sample a new airfoil $u_1^{'}$, and conduct a regression in function space via maximum a posteriori estimation $\mathcal{L}_{\text{MAP}}$. After a few iterations of fine-tuning, we can generate edited airfoil u_1^δ with high accuracy.

which according to Conditional Matching Theorem, has same gradients with Conditional Flow Matching loss:

$$\mathcal{L}_{\text{CFM}} = E_{t\sim[0,1],u_0,u_1}\left[\|v_t(u_t)|_{u_0,u_1} - v_\theta(u_t|_{u_0,u_1}, c, t)\|^2\right], \tag{4}$$

$$\nabla_\theta \mathcal{L}_{\text{FM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{CFM}}(\theta). \tag{5}$$

See more proofs in Chapter 4 of *Flow Matching Guide and Code* [36].

Inference. Given a trained velocity operator v_θ, the inference process, *i.e.,* airfoil generation process, is equal to deriving airfoil geometry at time $t = 1$, denoted as u_1, based on a latent coding u_0 sampled from the stochastic process $\mathcal{P}$. $\mathcal{P}$ is assumed as a Gaussian Process $\mathcal{GP}(0, K)$ in this work, where K is a covariance kernel function. Therefore, u_1 could be derived by solving the generation ODE numerically (e.g., using the Euler method) as follows:

$$u_1 = u_0 + \int_{t=0}^{t=1} v_\theta(u_t, c, t)\,\mathrm{d}t. \tag{6}$$

Detailed implementations for training velocity operator and model inference are shown in Appendix B, Algorithm 1 and Algorithm 2.

3.2 Airfoil Editing

The airfoil editing task enables the user to modify parts of the geometry of a given airfoil u_1, effectively generating a new airfoil geometry $u_1^{'}$ while preserving the user-edited sections, denoted as u_1^δ. We achieve this by finetuning the pretrained model via maximum a posteriori (MAP) estimation, $\max p(u_1^{'} \mid u_1^\delta)$ where $p(u_1^{'} \mid u_1^\delta)$ is a probabilistic model that constrains the optimized airfoil $u_1^{'}$ fulfilling the editing requirements and following the generation prior.

To achieve the constraint probability model, we disentangled it with Bayes' Rule:

$$\max_{u_1^{'}} p(u_1^{'} \mid u_1^\delta) \;=\; \frac{p(u_1^\delta \mid u_1^{'}) \cdot p(u_1^{'})}{p(u_1^\delta)} \Rightarrow \max_{u_1^{'}} \log p(u_1^\delta|u_1^{'}) + \log p(u_1^{'}) - \log p(u_1^\delta), \tag{7}$$

where $p(u_1^\delta \mid u_1^{'})$ is a Gaussian measure, so its log term becomes a Mean Square Error (MSE) between u_1^δ and $u_1^{'}$, constraining the user edited parts in $u_1^{'}$ keeping consistent with u_1^δ. We can sample $u_1^{'}$ through solving the velocity operator using Neural ODE method[6] and calculate $\log p(u_1^{'})$ at the same time using FFJORD method [17] with Hutchinson trace estimator [22]. $p(u_1^{'})$ is the prior supported by the trained generative model. $p(u_1^\delta)$ is the marginal likelihood, which does not depend on $u_1^{'}$. The final optimization target could be written as:

$$\max_{u_1^{'}} \frac{1}{2\sigma^2} \sum_{i\in\Delta} (u_1^{\delta,i} - u_1^{',i})^2 + \log p(u_1^{'}), \tag{8}$$

where σ is noise scale for editing, Δ denotes point indices in edited part u_1^δ.

For more realistic generation results, the optimization does not directly adjust $u_1^{'}$; instead, we optimize $u_1^{'}$ indirectly by fine-tuning the entire generative model for a few iterations. The fine-tuning process is illustrated in Figure 3, while its details are provided in Alg. 3 in the Appendix. Specifically, we first initialize u_1' as a resample data of u_1, by extracting its latent code, denoted as α_θ, via the inverse of our generative model and re-generating $u_1^{'}$ from this code. Then we treat Equation 8 as the loss function to train θ of the velocity operator and α_θ simultaneously. After the model training, the new $u_1^{'}$ is the new edited generation results.

4 Experiments

4.1 Experimental Settings

Tasks. We evaluate two tasks within the airfoil inverse design problem: *conditional airfoil generation* and *freestyle airfoil editing*. In the conditional generation task, the model is provided with a set of 11 geometric parameters describing the airfoil geometry. Detailed parameter definitions are provided in Table 6 in the Appendix. The model must generate airfoils that satisfy these geometric constraints. In the freestyle editing task, the model receives an original airfoil and a target modification, such as adjusting the position of a specific point on the airfoil. *The selected point can be located anywhere on the airfoil surface.* The model must generate an airfoil reflecting the specified modification.

Metrics. We adopt the metrics introduced in AFBench [37] to evaluate the generated airfoils:

Label Error measures the difference between the PARSEC parameters of the generated or edited airfoil and the intended target parameters, calculated as $\sigma_i = |\hat{p}_i - p_i|$, where σ_i is the label error for the i-th parameter, $\hat{p}_i$ is the i-th geometric parameter of the generated airfoil, and p_i is the corresponding target parameter. Smaller values indicate better alignment with target parameters. To summarize all 11 label errors, we present both the arithmetic mean $\bar{\sigma}_a$ and the geometric mean $\bar{\sigma}_g$ for absolute and relative average errors.

Diversity quantifies the variety among generated airfoils, calculated as $D = \frac{1}{n}\sum_{i=1}^{n} \log\det(L_{S_i})$, where n is the number of subsets, and L_{S_i} is the similarity matrix of the i-th subset, computed based on Euclidean distances between airfoils within the subset. Higher values indicate greater diversity among generated airfoils.

Smoothness measures the geometric smoothness of generated airfoils, calculated as:

$$M = \sum_{n=1}^{N} \text{Distance}(P_n, P_{n-1}P_{n+1}), \tag{9}$$

where P_n is the n-th point, and $P_{n-1}P_{n+1}$ is the line segment connecting its adjacent points. The function $\text{Distance}(P_n, P_{n-1}P_{n+1})$ computes the perpendicular distance from P_n to this line segment. Smaller values indicate better geometric quality.

Datasets. To benchmark our method, we conduct experiments on three datasets: UIUC [50], Supercritical Airfoil (Super), and AF-200K. UIUC contains 1,600 designed airfoil geometries. Super focuses on supercritical airfoils and includes approximately 20,000 airfoil samples. AF-200K consists of 200,000 highly diversified airfoil samples.

Baselines. We compare FuncGenFoil with baseline models proposed in AFBench [37], specifically the conditional VAE (CVAE), conditional GAN (CGAN), the modified VAE with PARSEC parameters and control keypoints (PK-VAE), as well as PK-GAN, PKVAE-GAN, the U-Net-based PK-DIFF, and the transformer-based PK-DIT.

Implementation Details. On the AF-200K dataset, we trained for 2 million iterations with a batch size of 2,048 using 8 NVIDIA 4090 GPUs. On the Supercritical Airfoil and the UIUC dataset, we trained for 1 million iterations with a batch size of 1,024 on 4 single NVIDIA 4090 GPUs. We use Gaussian processes with a Matérn kernel function ($\nu = 2.5, l = 0.03$) as the prior. Other training hyperparameters are detailed in Appendix C.

4.2 Main Results

Conditional Airfoil Generation. As shown in Table 1, FuncGenFoil outperforms the strongest baseline method (PK-DIT) across all metrics. For *label error*, FuncGenFoil achieves reductions in

Table 1: Quantitative comparison between FuncGenFoil and baseline methods across different datasets for the conditional generation task. Label error, diversity, and smoothness of generated airfoils are reported.

Method	Dataset	Label Error ↓ (10^{-3})													$\mathcal{D}$ ↑	$\mathcal{M}$ ↓ (10^{-2})
		σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	$\bar{\sigma}_a$	$\bar{\sigma}_g$		
CVAE	AF-200K	72.9	52.5	35.2	15900	99	95	29000	19.1	15.3	46	104	4131	149.2	-155.4	7.09
CGAN	AF-200K	107	85.0	54.4	23200	143	137	59600	25.3	22.3	53	129	7596	217.6	-120.5	7.31
PK-VAE	AF-200K	63.0	47.9	31.3	8620	66	64	17100	13.5	9.3	33	78	2375	106.0	-150.1	5.93
PK-GAN	AF-200K	81.8	63.0	47.0	21030	120	117	32470	22.5	19.6	50	122	4922	179.5	-112.3	3.98
PKVAE-GAN	AF-200K	56.8	31.7	31.0	5650	46	43	12000	9.1	5.1	28	63	1633	77.6	-129.6	2.89
PK-DIT	AF-200K	11.2	32.3	15.4	1050	**13**	11.5	9790	0.5	**0.5**	**23**	**24**	997	23.5	-93.2	**1.04**
FuncGenFoil	AF-200K	**1.84**	**17.4**	**0.56**	**721**	47.7	**0.98**	**1676**	**0.45**	0.65	160	174	**255**	**14.9**	**-71.6**	1.41
PK-VAE	UIUC	80.7	20.9	12.2	12843	36.9	14.0	37263	1.7	1.9	94.8	109.9	4589	69.1	-93.5	7.29
PK-DIT	UIUC	63.8	51.4	33.6	11830	87	84.9	25700	16.9	11.9	**36**	**98**	3456	129.8	-141.7	6.03
FuncGenFoil	UIUC	**11.6**	**11.8**	**0.38**	**698**	**23.5**	**0.56**	**12339**	**0.27**	**0.44**	110.5	115.0	**1210**	**15.1**	**-91.1**	**1.33**
PK-VAE	Super	10.8	17.5	2.3	1735.6	12.1	3.2	8131	3.5	1.4	98.3	80.6	918	28.4	-122.8	1.38
PK-DIT	Super	52.0	35.0	24.0	3010	29	33.2	10500	8.3	2.6	**27**	**33**	1250	58.1	-123.4	1.97
FuncGenFoil	Super	**0.71**	**8.23**	**0.13**	**201.3**	**4.72**	**0.12**	**174.2**	**0.09**	**0.14**	34.2	36.7	**41.9**	**3.08**	**-103.9**	**1.01**

Table 2: Quantitative evaluation of the FuncGenFoil model across different sampling super-resolutions for the conditional generation task. The training resolution is 257.

Dataset	Resolution	Label Error ↓ (10^{-3})													$\mathcal{D}$ ↑	$\mathcal{M}$ ↓ (10^{-2})
		σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	$\bar{\sigma}_a$	$\bar{\sigma}_g$		
Super	257	0.71	8.23	0.13	201.3	4.72	0.12	174.2	0.09	0.14	34.2	36.7	41.9	3.08	-103.9	1.01
	513	0.71	8.19	0.12	200.8	4.81	0.12	174.7	0.09	0.15	48.2	47.8	44.1	3.26	-97.5	0.52
	1025	0.71	8.29	0.12	202.9	4.93	0.12	175.4	0.09	0.15	61.4	57.8	46.5	3.44	-91.1	0.37

arithmetic mean label error of **74.4%** on AF-200K, **65.0%** on UIUC, and **96.6%** on Super. More importantly, it also reduces the geometric mean label error by **36.6%** on AF-200K, **88.4%** on UIUC, and **94.7%** on Super. This substantial decrease in label error underscores the effectiveness of our approach for generating airfoils that more precisely adhere to target geometric parameters. In terms of *diversity*, FuncGenFoil demonstrates notable improvements, surpassing the best baseline methods by **23.2%**, **35.7%**, and **15.8%** on AF-200K, UIUC, and Super, respectively. This highlights the model's superior capability in capturing and generating a broader spectrum of valid airfoil designs. Additionally, the generated airfoils exhibit enhanced surface *smoothness*, as evidenced by reductions of **4.70** and **0.96** in smoothness values (10^{-2}) on the UIUC and Super datasets, respectively. This improvement is particularly crucial for aerodynamic performance, as smoother airfoil surfaces contribute to reduced drag and improved flow characteristics. The limited coverage of the training dataset constrains the model's effectiveness to a specific operational range, which is detailed in Table 9. A performance comparison between FuncGenFoil and classical methods is provided in Appendix D.1.

Freestyle Airfoil Editing. Table 3 shows average performance over 300 editing cases using FuncGenFoil. In each case, the model adjusts 2 to 4 randomly selected positions on the airfoil surface to target locations over 10 fine-tuning steps, with editing scales ranging from 1×10^{-4} to 3.2×10^{-3}. Results demonstrate that FuncGenFoil achieves accurate airfoil editing with minimal errors (less than $\mathbf{2.75 \times 10^{-7}}$ MSE) and high surface smoothness (less than $\mathbf{1.16 \times 10^{-2}}$ smoothness value) after only a few fine-tuning steps if the edit scale is less than $\mathbf{4 \times 10^{-4}}$. The edit error increases at an accelerating rate with larger edits but remains relatively low overall. Figure 4 illustrates example editing requirements and corresponding generated airfoils during the fine-tuning stage, demonstrating that our model effectively completes freestyle editing tasks by generating accurate airfoils according to user-specified edits.

Table 3: Quantitative evaluation of the airfoil editing task across different editing scales. Mean squared error (MSE) between generated and target airfoils and smoothness of generated airfoils are reported.

Dataset	Edit Scale	MSE ↓ (10^{-7})	$\mathcal{M}$ ↓ (10^{-2})
Super	0.0001	2.41	1.16
	0.0002	2.45	1.15
	0.0004	2.75	1.15
	0.0008	4.32	1.26
	0.0016	15.5	1.35
	0.0032	61.7	1.49

Any-Resolution Airfoil Generation. One of the key advantages of FuncGenFoil is its ability to generate airfoils at any resolution while maintaining high generation quality. This is achieved

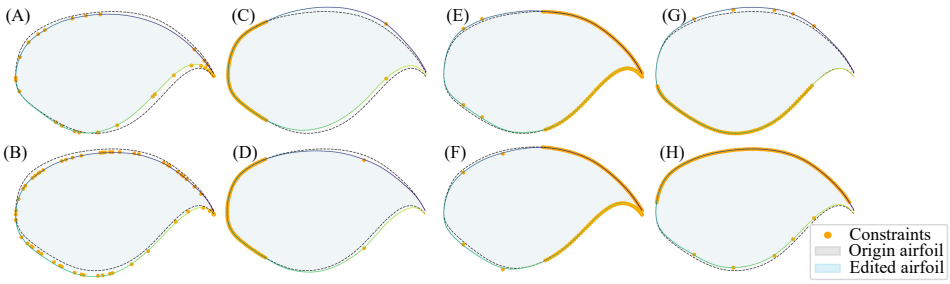


Figure 4: Examples of eight different FuncGenFoil instances performing airfoil editing over 20 training iterations. The orange points and sections represent the editing requirements as constraints; the gray region is the original airfoil, while the blue region shows the generated edited airfoil. The results demonstrate that the generated airfoil quickly adapts to the editing requirements within a few iterations, achieving a natural and smooth function regression. The editing scheme can be completely customized according to the user's preference.

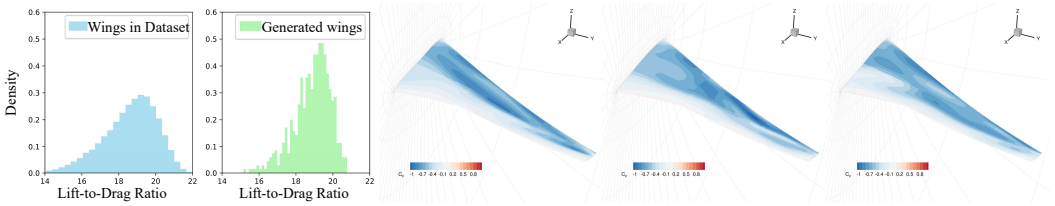


Figure 5: **Left**: Comparison of the lift-to-drag ratio histograms for CRM wings in the original dataset and the samples generated by our FuncGenFoil model. **Right**: Visualization of the pressure coefficient for the generated CRM wings, obtained through aerodynamic simulation.

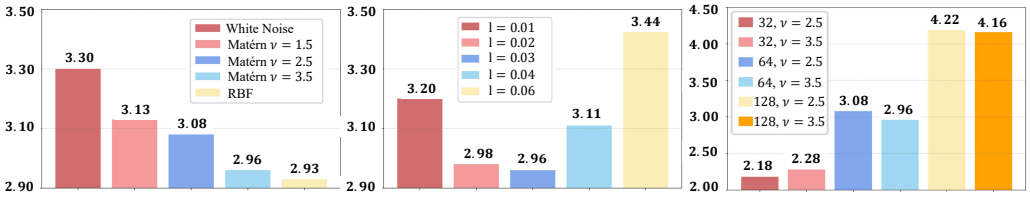
by representing airfoils as functions and learning resolution-independent function transformations, enabling flexible and consistent airfoil generation across different scales. To evaluate high-resolution generation, we use the model trained on the Supercritical Airfoil dataset at a resolution of 257 and sample new airfoils at resolutions of 513 and 1025. We then assess these higher-resolution airfoils using the same metrics as in the conditional generation task. The results are presented in Table 2. We observe consistent generation quality at $2\times$ and even $4\times$ the training resolution, with a maximum increase of $\mathbf{4.6}\times\mathbf{10^{-3}}$ in average label error, a decrease of $\mathbf{12.8}$ in diversity, and an increase of $\mathbf{0.64}\times\mathbf{10^{-2}}$ in smoothness value.

Aerodynamic Simulation. To further assess the physical properties of the generated airfoils and validate the effectiveness of our method, we analyze their aerodynamic performance using the NASA Common Research Model (CRM) [2] and perform Reynolds-Averaged Navier-Stokes (RANS) computational fluid dynamics (CFD) simulations on the generated samples. The CRM dataset contains 135,000 CRM wing geometries along with their corresponding aerodynamic performance, computed using the RANS CFD solver ADflow [39]. We pretrain the FuncGenFoil model on all 135,000 CRM wing geometries and generate 500 new CRM wing geometries for RANS CFD evaluation. We analyze the lift-to-drag ratio ($\boldsymbol{L/D}$) of these newly generated geometries and compare them with the original dataset, as shown in Figure 5. Our results indicate that the $\boldsymbol{L/D}$ distribution of the generated samples closely aligns with that of the original dataset, with the highest density occurring around $\boldsymbol{L/D = 19}$, demonstrating the model's ability to learn and generate physically plausible wing geometries. Additionally, we visualize the coefficient of pressure contours for selected CFD cases in Figure 5. These visualizations confirm that FuncGenFoil can generate airfoils with diverse aerodynamic performance characteristics.

4.3 Ablation Study

Kernels of Gaussian Process. The type of Gaussian process kernel used as a prior in the model influences the function space into which the data is diffused. We perform ablation experiments across several kernel types, including the white noise kernel, the Matérn kernel with smoothness parameter $\boldsymbol{\nu \in [1.5, 2.5, 3.5]}$, and the radial basis function (RBF) kernel. We measure the geometric mean $\bar{\boldsymbol{\sigma}}_{\boldsymbol{g}}$

[2] https://commonresearchmodel.larc.nasa.gov/



(A) $\bar{\sigma}_g$ under diverse smoothness (B) $\bar{\sigma}_g$ under diverse characteristic length (C) $\bar{\sigma}_g$ under diverse Fourier mode

Figure 6: **(A)**: Comparison of the geometric mean error $\bar{\boldsymbol{\sigma}}_{\boldsymbol{g}}$ using kernels of different smoothness but same characteristic length $\boldsymbol{l = 0.03}$. **(B)**: Comparison of the geometric mean error $\bar{\boldsymbol{\sigma}}_{\boldsymbol{g}}$ using Matérn kernel $\boldsymbol{\nu = 3.5}$ with different characteristic length $\boldsymbol{l}$. **(C)**: Comparison of the geometric mean error $\bar{\boldsymbol{\sigma}}_{\boldsymbol{g}}$ using different Fourier mode in FNO using Matérn kernel of same $\boldsymbol{l = 0.03}$.

Table 5: Quantitative experiments on the impact of ODE solver and time step on sampling quality.

ODE Solver	Time Steps	Label Error ↓ (10^{-3})													$\mathcal{D}\uparrow$	$\mathcal{M}\downarrow$ (10^{-2})
		σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	$\bar{\sigma}_a$	$\bar{\sigma}_g$		
Euler	**10**	0.71	8.23	0.13	201.3	4.72	0.12	174.2	0.09	0.14	34.2	36.7	41.9	3.08	-103.9	1.01
	50	0.62	3.61	0.07	92.5	2.08	0.06	96.8	0.06	0.07	14.5	15.7	20.6	1.60	-107.6	0.99
	100	0.67	3.13	0.09	83.9	1.84	0.08	90.6	0.07	0.06	12.3	13.1	18.7	1.55	-106.4	0.99
Midpoint	**10**	0.68	5.29	0.08	161.3	3.12	0.08	178.9	0.06	0.09	21.4	22.4	35.8	2.17	-105.1	1.00
	50	0.75	3.82	0.12	137.9	2.34	0.10	153.4	0.08	0.06	14.1	14.9	29.8	1.97	-103.3	0.99
	100	0.82	3.94	0.14	125.7	2.22	0.13	137.4	0.08	0.07	14.4	14.2	27.2	2.06	-101.6	0.99
RK4	**10**	15.9	46.3	1.21	4406	25.7	1.24	4795	1.17	1.68	333	396	911	36.6	-67.5	3.05
	50	0.92	7.63	0.24	482.7	5.55	0.22	511.8	0.16	0.19	39.3	41.3	99.1	4.68	-92.9	1.07
	100	0.82	5.65	0.18	160.6	3.21	0.17	170.8	0.10	0.12	22.0	22.8	35.1	2.78	-96.9	1.00

under different smoothness, characteristic length, and Fourier mode settings, as shown in Figure 6. We observe that FunGenfoil benefits from employing smoother kernels, which enhance both the smoothness and accuracy of the generated airfoils. This effect arises because the parameter $\boldsymbol{\nu}$ primarily controls the smoothness of the function space, and the RBF kernel corresponds to the case $\boldsymbol{\nu \to \infty}$. Additionally, we find that there exists an optimal characteristic length at $\boldsymbol{l = 0.03}$ for the kernel. In our experiments, FunGenfoil does not exhibit improved performance with an increased number of Fourier modes. This phenomenon may be due to the predominantly low-frequency characteristics of supercritical airfoils, such that adding higher modes in the FNO layer introduces unnecessary model complexity. More detailed experimental results are provided in Table 11 in the Appendix.

Editing with Prior via ODE Inversion. In Table 4, we compare MSE of constraint condition calculations in the airfoil editing task using two initialization schemes: (1) using the latent variables obtained through ODE inversion of the original airfoil as the prior, and (2) using a zero prior $\boldsymbol{u_0 = 0}$ without ODE inversion. Both schemes are fine-tuned with the same number of iterations. The ODE inversion-based prior significantly reduces the MSE compared to the zero prior, with a maximum reduction of **84.85**. This underscores the importance of incorporating original airfoil information during the initialization process.

Table 4: Ablation study on the effect of different latent variable initialization methods for the airfoil editing task.

Edit scale	MSE↓(10^{-7})	
	w/ ODE inv.	**w/o** ODE inv.
0.0001	2.42 (↓83.88)	86.3
0.0002	2.46 (↓84.14)	86.6
0.0004	2.75 (↓84.85)	87.6

ODE Numerical Method. Model inference involves solving an ordinary differential equation (ODE), and the choice of numerical integration scheme affects its performance. Using the same velocity operator $\boldsymbol{v_\theta}$ trained on the supercritical airfoil dataset and sampling from a Gaussian prior with a Matérn kernel ($\boldsymbol{\nu = 2.5, l = 0.03}$), we evaluate FuncGenFoil across various time step sizes and ODE solvers (Euler, midpoint, and fourth-order Runge-Kutta (RK4) methods). Results in Table 5 show that FuncGenFoil benefits from larger time steps using Euler method. However, RK4 may degrade performance at excessively small step sizes, potentially because the model enters regions where the velocity operator is inadequately trained.

5 Conclusion and Limitations

In this work, we tackle the critical challenge of generating high-fidelity airfoil geometries that effectively balance expressiveness, smoothness, and resolution flexibility. We introduce **FuncGenFoil**, a function-space generative model leveraging neural operators and flow matching, which represents airfoils as continuous, smooth geometries without resolution constraints while preserving the expressiveness of data-driven methods. Comprehensive experimental results show that FuncGenFoil outperforms state-of-the-art techniques in terms of label error, diversity, and smoothness, highlighting its potential for high-fidelity airfoil design. Furthermore, the generated wing geometries have been validated through aerodynamic simulations. This work paves the way for more efficient, scalable, and versatile airfoil generation, with significant applications in aerodynamic shape optimization for aircraft manufacturing.

As for limitations, although we have taken a step toward general object shape modeling in function space, the focus of this paper is primarily on airfoils or aircraft wings, which have relatively simple geometries and smooth surfaces yet are highly significant for aerodynamic performance. Thus, our current approach has limitations regarding the scope of geometry. In future studies, if we aim to extend our method to modeling entire aircraft or dealing with objects of general shapes, substantial theoretical work and experimental analysis will be required, especially when a suitable coordinate system is unavailable for describing complex, irregular, or non-smooth geometries. This direction is part of our ongoing efforts.

Acknowledgements

Funding: This work was supported by the Shanghai Artificial Intelligence Laboratory, the JC STEM Lab of AI for Science and Engineering, funded by The Hong Kong Jockey Club Charities Trust, the Research Grants Council of Hong Kong (Project No. CUHK14213224), the Natural Science Foundation of China (No. U23A2069), the AI for Science Seed Program of Shanghai Jiao Tong University (Project No. 2025AI4S-HY02), and the AI for Science Program, Shanghai Municipal Commission of Economy and Informatization (No. 2025-GZL-RGZN-BTBX-01010).

Support and Collaboration: We thank the team at Shanghai Aircraft Design and Research Institute and Shanghai Jiao Tong University for their valuable discussions and continued support.

References

[1] Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2019.

[2] Kamyar Azizzadenesheli, Nikola Kovachki, Zongyi Li, Miguel Liu-Schiaffini, Jean Kossaifi, and Anima Anandkumar. Neural operators for accelerating scientific simulations and design. April 2024.

[3] Gal Berkooz, Philip Holmes, and John L. Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 1993.

[4] Olivier Brüls, Pierre Duysinx, and Jean-Claude Golinval. The global modal parameterization for non-linear model-order reduction in flexible multibody dynamics. *International journal for numerical methods in engineering*, 2007.

[5] Shuhao Cao. Choose a transformer: Fourier or galerkin. *NIPS*, 34:24924–24940, 2021.

[6] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *NIPS*, volume 31. Curran Associates, Inc., 2018.

[7] Wei Chen and Mark Fuge. Béziergan: Automatic generation of smooth curves from interpretable low-dimensional parameters, 2021.

[8] Russell M Cummings, William H Mason, Scott A Morton, and David R McDaniel. *Applied computational aerodynamics: A modern engineering approach*, volume 53. Cambridge University Press, 2015.

[9] III Dannenhoffer, John F. An overview of the engineering sketch pad. In *AIAA SciTech 2024 Forum*, Orlando, FL, January 2024.

[10] Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Jimenez Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *ICML*, Proceedings of Machine Learning Research. PMLR, 2022.

[11] Emilien Dupont, Yee Whye Teh, and Arnaud Doucet. Generative models as distributions of functions. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *AISTATS*. PMLR, 2022.

[12] Gerald E. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Morgan Kaufmann, 2002.

[13] Giulio Franzese, Giulio Corallo, Simone Rossi, Markus Heinonen, Maurizio Filippone, and Pietro Michiardi. Continuous-time functional diffusion processes. *NeurIPS*, 36, 2024.

[14] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and S. M. Ali Eslami. Conditional neural processes. In Jennifer Dy and Andreas Krause, editors, *ICML*, Proceedings of Machine Learning Research. PMLR, 2018.

[15] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural processes, 2018.

[16] Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. In *NeurIPS*, 2024.

[17] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, and David Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *ICML*, 2019.

[18] Andrew S. Hahn. Vehicle sketch pad: A parametric geometry modeler for conceptual aircraft design. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2010.

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H Larochelle, M Ranzato, R Hadsell, M F Balcan, and H Lin, editors, *NIPS*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

[20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *NIPS*, 33:6840–6851, 2020.

[21] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *NeurIPS*, volume 35, pages 8633–8646. Curran Associates, Inc., 2022.

[22] M.F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 1990.

[23] Gavin Kerrigan, Justin Ley, and Padhraic Smyth. Diffusion generative models in infinite dimensions. In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, editors, *AISTATS*, volume 206 of *Proceedings of Machine Learning Research*, pages 9538–9563. PMLR, 25–27 Apr 2023.

[24] Gavin Kerrigan, Giosue Migliorini, and Padhraic Smyth. Functional flow matching. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics*, 2024.

[25] Diederik P Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[26] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023.

[27] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: learning maps between function spaces with applications to pdes. *J. Mach. Learn. Res.*, 24(1), January 2023.

[28] B. M. Kulfan. Universal parametric geometry representation method. *Journal of Aircraft*, 2008.

[29] Jichao Li, Xiaosong Du, and Joaquim RRA Martins. Machine learning in aerodynamic shape optimization. *Progress in Aerospace Sciences*, 134:100849, 2022.

[30] Jichao Li and Mengqi Zhang. Adjoint-free aerodynamic shape optimization of the common research model wing. *AIAA Journal*, 2021.

[31] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *ICLR*, 2021.

[32] Jae Hyun Lim, Nikola B. Kovachki, Ricardo Baptista, Christopher Beckham, Kamyar Azizzadenesheli, Jean Kossaifi, Vikram Voleti, Jiaming Song, Karsten Kreis, Jan Kautz, Christopher Pal, Arash Vahdat, and Anima Anandkumar. Score-based diffusion models in function space, 2025.

[33] Jen Ning Lim, Sebastian Vollmer, Lorenz Wolf, and Andrew Duncan. Energy-based models for functional data using path measure tilting. In *AISTATS*, 2023.

[34] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *ICML*, 2023.

[35] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[36] Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

[37] Jian Liu, Jianyu Wu, Hairun Xie, Guoqing Zhang, Jing Wang, Wei Liu, Wanli Ouyang, Junjun Jiang, Xianming Liu, Shixiang Tang, et al. Afbench: A large-scale benchmark for airfoil design. In *NeurIPS 2024 Dataset and Benchmark Track*, 2024.

[38] Miguel Liu-Schiaffini, Julius Berner, Boris Bonev, Thorsten Kurth, Kamyar Azizzadenesheli, and Anima Anandkumar. Neural operators with localized integral and differential kernels. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024.

[39] Charles A. Mader, Gaetan K. W. Kenway, Anil Yildirim, and Joaquim R. R. A. Martins. ADflow—an open-source computational fluid dynamics solver for aerodynamic and multidisciplinary optimization. *Journal of Aerospace Information Systems*, 2020.

[40] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[41] Mehdi Mirza. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[42] Jakiw Pidstrigach, Youssef Marzouk, Sebastian Reich, and Sven Wang. Infinite-dimensional diffusion models, 2023.

[43] Adam Polyak, Amit Zohar, and Andrew Brown et al. Movie gen: A cast of media foundation models, 2024.

[44] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *NIPS*, 20, 2007.

[45] Md Ashiqur Rahman, Manuel A Florez, Anima Anandkumar, Zachary E Ross, and Kamyar Azizzadenesheli. Generative adversarial neural operators. *TMLR*, 2022.

[46] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*. Springer, 2003.

[47] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10684–10695, 2022.

[48] I. J. Schoenberg. Spline functions and the problem of graduation. *Proceedings of the National Academy of Sciences of the United States of America*, 52(4):947–950, October 1964.

[49] Vinothkumar Sekar, Mengqi Zhang, Chang Shu, and Boo Cheong Khoo. Inverse design of airfoil using a deep convolutional neural network. *AIAA Journal*, 2019.

[50] Michael S Selig. Uiuc airfoil database. 1996.

[51] Prashant Sharma, Bhupendra Gupta, Mukesh Pandey, Arun Kumar Sharma, and Raji Nareliya Mishra. Recent advancements in optimization methods for wind turbine airfoil design: A review. *Materials Today: Proceedings*, 2021.

[52] Yaozhong Shi, Angela F Gao, Zachary E Ross, and Kamyar Azizzadenesheli. Universal functional regression with neural operator flows. In *NeurIPS 2024 Workshop on Bayesian Decision-making and Uncertainty*, 2024.

[53] Yaozhong Shi, Zachary E. Ross, Domniki Asimaki, and Kamyar Azizzadenesheli. Stochastic process learning via operator flow matching, 2025.

[54] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020.

[55] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR*, 2021.

[56] Peng Su, Kun Wang, Xingyu Zeng, Shixiang Tang, Dapeng Chen, Di Qiu, and Xiaogang Wang. Adapting object detectors with conditional domain normalization. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 403–419. Springer, 2020.

[57] Alexander Tong, Kilian FATRAS, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *TMLR*, 2024. Expert Certification.

[58] Zhen Wei, Edouard R. Dufour, Colin Pelletier, Pascal Fua, and Michaël Bauerheim. Diffairfoil: An efficient novel airfoil sampler based on latent space diffusion model for aerodynamic shape optimization. In *AIAA AVIATION FORUM AND ASCEND*, 2024.

[59] Zhen Wei, Benoît Guillard, Pascal Fua, Vincent Chapin, and Michaël Bauerheim. Latent representation of computational fluid dynamics meshes and application to airfoil aerodynamics. *AIAA Journal*, 2023.

[60] Zhen Wei, Aobo Yang, Jichao Li, Michaël Bauerheim, Rhea P. Liem, and Pascal Fua. Deepgeo: Deep geometric mapping for automated and effective parameterization in aerodynamic shape optimization. In *AIAA AVIATION FORUM AND ASCEND*, 2024.

[61] Hairun Xie, Jing Wang, and Miao Zhang. Parametric generative schemes with geometric constraints for encoding and synthesizing airfoils. *Engineering Applications of Artificial Intelligence*, 2024.

[62] Aobo Yang, Jinouwen Zhang, Jichao Li, and Rhea Liem. Data-driven aerodynamic shape optimization and multi-fidelity design exploration using conditional diffusion-based geometry sampling method. In *ICAS*, 2024.

[63] Chenyu Zeng, Yanshu Zhang, Jiayi Zhou, Yuhan Wang, Zilin Wang, Yuhao Liu, Lei Wu, and Daniel Zhengyu Huang. Point cloud neural operator for parametric pdes on complex and variable geometries. *arXiv preprint arXiv:2501.14475*, 2025.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction can accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We address the limitation of our work in the section of limitation of this paper.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Proof for theoretical result are all given or referenced properly in this paper.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper fully disclose all the information needed to reproduce the main experimental results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: All codes and data are provided and can be obtained via open access. We provide detailed instructions to faithfully reproduce the main experimental results. See readme file in supplemental material.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: All the training and test details are given in the paper.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The metrics of all experiments in this paper is tiny and will not affect main results. Considering the size of the table, we decided not to provide error bars. All results can be reproduced running the evaluation code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide detailed information on the computer resources we use in Experiment Settings section.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conform Ethics Guide lines of NeurIPS.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discuss both potential positive societal impacts and negative societal impacts of the work performed in Appendix.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Code and data used in the paper are properly credited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve potential risks incurred by study participants.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: LLM is used only for formatting purposes in this paper.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

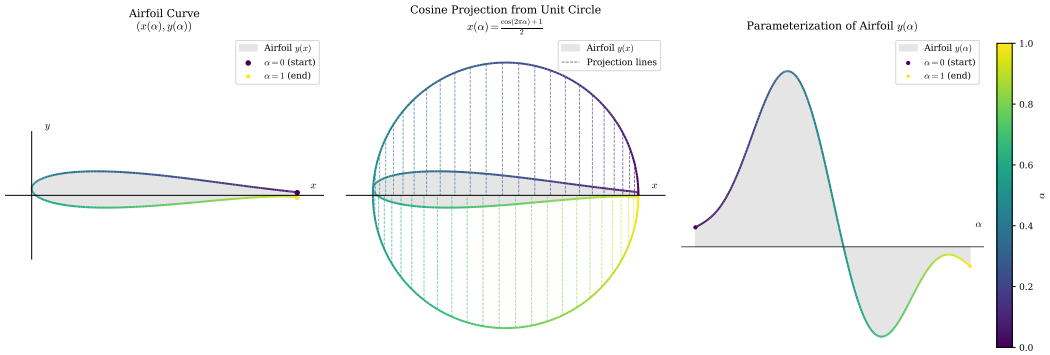
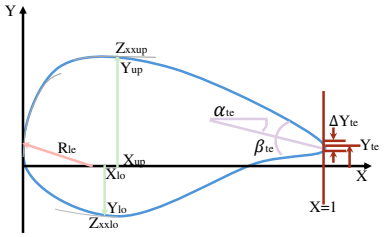


Figure 7: Illustration of the airfoil curve parameterization employed in FuncGenFoil.

Index	Symbol	Geometric Meaning
1	$\mathbf{R_{le}}$	leading edge radius
2	$\mathbf{X_{up}}$	upper crest position x
3	$\mathbf{Y_{up}}$	upper crest position y
4	$\mathbf{Z_{xxup}}$	upper crest curvature
5	$\mathbf{X_{lo}}$	lower crest position x
6	$\mathbf{Y_{lo}}$	lower crest position y
7	$\mathbf{Z_{xxlo}}$	lower crest curvature
8	$\mathbf{Y_{te}}$	trailing edge position
9	$\mathbf{\Delta Y_{te}}$	trailing thickness
10	$\boldsymbol{\alpha_{te}}$	trailing edge angle up
11	$\boldsymbol{\beta_{te}}$	trailing edge angle down

(a) PARSEC parameters and its geometric meaning.



(b) Demonstration of PARSEC parameters.

Table 6: Eleven PARSEC parameterization in airfoil design.

A Problem Setting for Airfoil Generation

A.1 Airfoil Parametrization

In this paper, we perform a circular parameterization of the airfoil curves, as illustrated in Figure 7. Specifically, we project a unit circle onto the x-axis and adopt the angle $\boldsymbol{\alpha}$ as the function domain. The parameterized airfoil function begins at the trailing edge on the suction side, where $\boldsymbol{\alpha} = \mathbf{0}$, and ends at the trailing edge on the pressure side, where $\boldsymbol{\alpha} = \mathbf{1}$. In this manner, we transform the airfoil data from a circular topology into a well-defined functional form suitable for reconstruction by generative models.

A.2 Design Parameters of Airfoil Geometry

We adopted the same design condition parameters as provided in AFBench [37], as shown in Table 6. The parameters listed below—including the leading-edge radius, the curvature at the maximum thickness of the upper and lower surfaces, and the trailing-edge angle of the airfoil—were obtained via curve fitting. Note that our implementation of the trailing-edge angle calculation slightly differs from that of AFBench, as we found the original approximation method not invariant to increased sampling precision. Specifically, AFBench employs B-spline curves for interpolation at the trailing edge and calculates its gradient at the endpoint, which becomes unstable as resolution increases. In contrast, our method assumes the last $\mathbf{2\%}$ length of the airfoil curve near the trailing edge is nearly straight, and uses linear regression to determine the angle.

Algorithm 1 Model Training.

Input: data resolution $\boldsymbol{d}$, data $\boldsymbol{u_1}$, design condition variables $\boldsymbol{c}$ (optional).
Parameter: Gaussian process $\boldsymbol{\mathcal{GP}(0, K)}$ for sampling $\boldsymbol{u_0}$.
Output: velocity operator $\boldsymbol{v_\theta}$.

while training... **do**
  sample $\boldsymbol{t} \sim [\mathbf{0}, \mathbf{1}]$, $\boldsymbol{u_0} \sim \boldsymbol{\mathcal{GP}(0, K)}$ at resolution $\boldsymbol{d}$ and get $\boldsymbol{\{u_{0,i}\}}$.
  Compute $\boldsymbol{v_t}$ at resolution $\boldsymbol{d}$ and get $\boldsymbol{\{v_{t,i}\}}$.
  Compute $\boldsymbol{u_t}$ at resolution $\boldsymbol{d}$ and get $\boldsymbol{\{u_{t,i}\}}$.
  Compute $\boldsymbol{v_\theta(u_t, c, t)}$ at resolution $\boldsymbol{d}$ and get $\boldsymbol{\{v_{\theta,i}\}}$.
  Minimize $\boldsymbol{\|\{v_{\theta,i}\} - \{v_{t,i}\}\|^2}$.
  Compute gradient and update $\boldsymbol{\theta}$.
end while
return $\boldsymbol{v_\theta}$.

Algorithm 2 Model Inference.

Input: sampling resolution $\boldsymbol{d}$, sampling time steps $\boldsymbol{T}$ and steps length $\mathbf{d}\boldsymbol{t}$, latent function $\boldsymbol{u_0}$ (optional), design condition variables $\boldsymbol{c}$ (optional).
Parameter: Gaussian process $\boldsymbol{\mathcal{GP}(0, K)}$ for sampling $\boldsymbol{u_0}$.
Output: airfoil $\boldsymbol{\{y_i\}}$ at resolution $\boldsymbol{d}$.

Let $\boldsymbol{t = 0}$, $\boldsymbol{u_0} \sim \boldsymbol{\mathcal{GP}(0, K)}$ at resolution $\boldsymbol{d}$ and get $\boldsymbol{\{u_{0,i}\}}$.
while $\boldsymbol{t \leq 1}$ **do**
  Compute $\boldsymbol{v_\theta(u_t, c, t)}$ at resolution $\boldsymbol{d}$ and get $\boldsymbol{\{v_{\theta,i}\}}$.
  Compute $\boldsymbol{\{u_{t+\mathrm{d}t,i}\} = \{u_{t,i}\} + \{v_{\theta,i}\mathrm{d}t\}}$.
  $\boldsymbol{t = t + \mathrm{d}t}$.
end while
return $\boldsymbol{\{y_i\} = \{u_{1,i}\}}$

B Algorithm Implementation Details

The algorithm for training the airfoil generative model is presented in Algorithm 1, and the inference procedure using a trained model is described in Algorithm 2. In our implementation, we employed Optimal Flow Matching and utilized a Gaussian process with specific kernel functions, such as the RBF kernel or the Matérn kernel, in the latent space of the generative model.

C Training Details

In most of the experiments, we followed the hyperparameters listed in the Table 7 below to train the models, including both the pre-training and fine-tuning stages. On the AF-200K dataset, we increased the maximum number of iterations to $\mathbf{2000000}$ and the batch size to $\mathbf{2048}$. For the number of time steps used to solve the ODE, we use $\mathbf{10}$ steps for relative balance between fast sampling speed and good performance, except for the fine-tuning stage, where we used 10 steps for training efficiency.

To provide a comprehensive analysis of the model's computational requirements, we evaluate its performance relative to the point-based PK-DIT model. Table 8 details this comparison, presenting the wall-clock training time, per-sample inference latency, and peak GPU memory usage. All benchmarks were conducted on a single desktop machine equipped with an NVIDIA RTX 4090 and an Intel i9-13900K.

D More Experiments

D.1 Comparison with a Classical Optimization Method

To establish a performance benchmark, we compare our method against a classical, optimization-based approach for airfoil reconstruction. This benchmark is designed to estimate the near-optimal accuracy achievable with a standard parametric representation.

The classical method represents the airfoil using a NURBS curve with $\mathbf{24}$ control points. To create a best-case scenario for this method, the optimization for each of the $\mathbf{3825}$ target airfoils in the

Algorithm 3 Model Finetuning (Airfoil Editing).

Input: pretrained neural operator $\boldsymbol{v_\theta}$, original airfoil function $\boldsymbol{u_1}$ (optional) or latent function $\boldsymbol{u_0}$ (optional), editing requirement $\boldsymbol{\delta}$, editing resolution $\boldsymbol{d}$, sampling time steps $\boldsymbol{T}$ and steps length $\mathbf{d}\boldsymbol{t}$.
Parameter: Gaussian process $\boldsymbol{\mathcal{GP}(0, K)}$ for sampling $\boldsymbol{u_0}$, noise level $\boldsymbol{\sigma}$.
Output: new airfoil $\boldsymbol{\{y_i\}}$ at resolution $\boldsymbol{d}$.

Inversely sample original airfoil function $\boldsymbol{\{u_{1,i}\}}$ through $\boldsymbol{v_\theta}$ and get its latent function $\boldsymbol{\{u_{0,i}\}}$.
Set $\boldsymbol{\{a_{\theta,i}\} = \{u_{0,i}\}}$.
while finetuning... **do**
    Sample $\boldsymbol{\{u^{'}_{1,i}\}}$ through $\boldsymbol{v_\theta}$ from $\boldsymbol{\{a_{\theta,i}\}}$.
    Compute $\boldsymbol{\mathcal{L}_{\mathrm{MAP}}}$.
    Compute gradient and update $\boldsymbol{\theta}$.
end while
Sample $\boldsymbol{\{u^{'}_{1,i}\}}$ through $\boldsymbol{v_\theta}$ from $\boldsymbol{\{a_{\theta,i}\}}$.
return $\boldsymbol{\{y_i\} = \{u^{'}_{1,i}\}}$

Table 7: Training hyperparameters and model parameters in conditional airfoil generation tasks

Training hyperparameters	**Value**
Max learning rate	$\mathbf{5 \times 10^{-6}}$
Batch size	$\mathbf{1024}$
Batch size (AF-200K)	$\mathbf{2048}$
Optimizer (pre-training and fine-tuning)	Adam
Optimizer scheduler	Cosine Annealing
Warmup iterations	$\mathbf{2000}$
Max training iterations	$\mathbf{1000000}$
Max training iterations (AF-200K)	$\mathbf{2000000}$
ODE solver time steps	$\mathbf{10}$
ODE solver type	Euler
Model parameters	**Value**
Fourier neural operator layers	$\mathbf{6}$
Fourier neural operator modes	$\mathbf{64}$
Fourier neural operator hidden channels	$\mathbf{256}$
Gaussian process kernel	Matérn
Kernel noise scale $\boldsymbol{n}$	$\mathbf{1}$
Kernel order $\boldsymbol{\nu}$	$\mathbf{2.5}$
Kernel characteristic length $\boldsymbol{l}$	$\mathbf{0.03}$

test dataset was initialized with control points configured to match the *average* airfoil shape. This starting point is already very close to the final target, ensuring stable and rapid convergence. For each target, the optimization was performed using the Adam optimizer for 100 steps with a learning rate of $\mathbf{3 \times 10^{-4}}$, continuing until the Mean Squared Error (MSE) plateaued. The total computation cost for fitting the entire test set was approximately 20 GPU-hours on an NVIDIA A800 GPU.

The results of this comparison are summarized in Table 10. As expected, the NURBS fitting baseline performs exceptionally well, closely approaching what might be considered a practical upper bound on accuracy for a $\mathbf{24}$-parameter representation. FuncGenFoil's accuracy is competitive with this strong baseline. Analyzing the geometric errors reveals that the NURBS fitting method excels at the trailing edge, likely because the control points can effectively constrain this region. However, FuncGenFoil demonstrates superior performance at the leading edge. The classical method struggles with the high curvature of the leading edge, where the $\mathbf{24}$ control points lack the flexibility to capture the geometry accurately.

It is crucial to highlight a fundamental difference in the problem setup that heavily favors the classical method. Its optimization begins from a well-conditioned starting point, whereas generative models like FuncGenFoil or PK-DIT must learn to construct airfoils from a state of maximum entropy (i.e., pure Gaussian noise)—a significantly more challenging task. We note that the classical optimization process is delicate; initializing the control points more randomly often caused the optimization to become unstable and diverge.

Table 8: Time cost comparison of different generative models.

Model	Wall-clock for 1 000 epochs	NFEs at test time	Mean inference time	GPU memory at test
PK-DIT (score matching)	$\approx$ 10 h	50 (DDIM)	220 ms	$\approx$ 200 MB
FuncGenFoil (flow matching)	$<$ 6 h	10	50 ms	$\approx$ 200 MB

Table 9: Effective ranges of the 11 geometric parameters for supercritical dataset.

Index	Parameter	Minimum	Maximum
1	leading edge radius	0.0073	0.0140
2	upper crest position x	0.3960	0.5200
3	upper crest position y	0.0592	0.0784
4	upper crest curvature	-0.4580	-0.2100
5	lower crest position x	0.3180	0.4100
6	lower crest position y	-0.0589	-0.0414
7	lower crest curvature	0.3730	0.8050
8	trailing edge position	-0.0001	0.0001
9	trailing thickness	0.0020	0.0075
10	trailing edge angle up	-0.5514	-0.2254
11	trailing edge angle down	-0.4477	-0.1397

Therefore, this experiment should not be interpreted as a direct comparison. Instead, it provides a reference for the best-case reconstruction performance of a model-free parametric method under ideal optimization conditions.

D.2 More Ablations Experiments

We conduct a comprehensive ablation study on the choice of Gaussian process kernel types, model modes of Fourier Neural Operators, and characteristic lengths, as shown in Table 11 for the Matérn kernel, Table 12 for the radial basis function (RBF) kernel, and Table 13 for the white noise kernel, which is commonly used in diffusion or flow models in finite-dimensional spaces. In general, appropriate selections of these settings can significantly enhance model performance. Specifically, the Gaussian prior should exclude function spaces where the data is unlikely to reside, making the prior space as restricted as possible to facilitate effective model training. For example, a dataset with high smoothness is unlikely to belong to the function space generated by a white noise prior but is more plausibly sampled from a Gaussian prior with an RBF kernel.

In Table 12, we observe that the RBF kernel exhibits a similar tendency to the Matérn kernel, with both achieving optimal performance at the characteristic length $\mathbf{0.03}$. As for the white noise kernel in Table 13, since its characteristic length is effectively $\mathbf{0}$, we studied the influence of the noise scale $\boldsymbol{n}$ over the range $[\mathbf{1.0, 2.0, 4.0}]$. We found that excessively large diffusion white noise negatively impacts model performance, likely because such noise scales project the function into a larger white noise space, making it more difficult for the neural operator to learn the intrinsic dynamics.

D.3 Generated Airfoil Examples

Figure 8 showcases a variety of airfoil geometries generated by FuncGenFoil models. These models were trained on datasets of supercritical airfoils, demonstrating their capability to produce diverse and plausible shapes.

E Societal Impacts

This work will accelerate research in AI for Science and Engineering and generally has positive social impacts. On the other hand, AI models for engineering design and optimization could accelerate technological diffusion within society, potentially raising issues related to intellectual property or unauthorized technology transfer to entities or individuals intending to design specific engineering products or construct items harmful to society.

Table 10: Performance Comparison: Classical Optimization vs. FuncGenFoil. The classical "NURBS fitting" method is initialized close to the target, representing a near-optimal benchmark.

Dataset	Algo.	Label Error ($\times 10^{-3}$) ↓												
		σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	$\bar{\sigma}_a$	$\bar{\sigma}_g$
Super	NURBS fitting	10.8	17.4	4.5	64.5	9.79	3.56	113.2	1.53	1.72	0.065	0.106	20.6	3.97
	FuncGenFoil	0.71	8.23	0.13	201.3	4.72	0.12	174.2	0.09	0.14	34.2	36.7	41.9	3.08

Table 11: FuncGenFoil model performance with different FNO mode and Matérn kernel for conditional airfoil generation.

Modes	ν	l	Label Error ↓ (10^{-3})													$\mathcal{D}$ ↑	$\mathcal{M}$ ↓ (10^{-2})
			σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	$\bar{\sigma}_a$	$\bar{\sigma}_g$		
32	2.5	0.01	0.62	6.06	**0.09**	200.4	3.58	0.09	185.3	**0.07**	0.12	26.7	29.1	41.1	2.51	-105.6	1.02
		0.02	0.52	5.10	0.10	97.4	3.26	0.11	107.0	0.08	0.12	26.2	**27.6**	24.3	2.22	-105.7	**0.99**
		0.03	0.54	**4.78**	0.11	**82.3**	3.00	0.11	93.2	0.09	0.12	**26.0**	27.7	**21.6**	**2.18**	-105.6	**0.99**
	3.5	0.01	0.63	5.64	**0.09**	158.3	3.46	**0.08**	156.9	**0.07**	0.12	27.6	29.2	34.7	2.41	-105.8	1.01
		0.02	**0.48**	5.08	0.10	89.4	3.12	0.10	97.5	0.08	0.12	26.8	28.2	22.8	**2.18**	-105.9	**0.99**
		0.03	0.53	5.17	0.11	83.5	**2.90**	0.11	**89.5**	0.10	0.14	29.4	30.7	22.0	2.28	-105.2	**0.99**
64	1.5	0.01	0.82	8.57	0.10	370.2	5.21	0.09	342.5	0.08	0.12	36.3	37.5	72.9	3.30	-104.0	1.09
		0.02	0.65	8.26	0.13	222.9	4.97	0.11	219.6	0.08	0.13	37.4	35.9	48.2	3.12	-103.6	1.04
		0.03	0.71	8.18	0.14	183.7	4.82	0.12	207.2	0.09	0.14	34.6	36.6	43.3	3.13	-104.0	1.02
	2.5	0.01	0.80	8.77	0.10	291.2	5.23	0.10	277.6	0.08	0.12	35.8	37.7	59.8	3.20	-103.7	1.06
		0.02	0.68	8.66	0.13	192.4	4.78	0.11	182.8	0.09	0.12	36.2	37.8	42.2	3.06	-103.5	1.02
		0.03	0.71	8.23	0.13	201.3	4.72	0.12	174.2	0.09	0.14	34.2	36.7	41.9	3.08	-103.9	1.01
	3.5	0.01	0.88	8.80	0.11	269.3	5.26	0.09	248.9	0.08	0.12	36.5	38.0	55.3	3.20	-103.4	1.05
		0.02	0.72	8.52	0.11	177.1	4.84	0.11	174.6	0.08	0.13	33.2	37.4	39.7	2.98	-103.6	1.02
		0.03	0.66	7.80	0.12	161.1	4.54	0.11	164.2	0.09	0.15	35.1	38.3	37.5	2.96	-103.4	1.01
		0.04	0.82	8.19	0.13	170.0	4.45	0.13	159.7	0.09	0.16	36.6	34.8	37.7	3.11	-103.6	1.01
		0.06	0.87	8.82	0.16	183.0	5.01	0.13	183.9	0.10	0.17	41.8	41.2	42.3	3.44	-102.8	1.02
		0.12	1.39	10.7	0.20	261.0	6.23	0.15	230.3	0.12	0.19	53.4	44.9	55.3	4.27	**−99.9**	1.05
128	2.5	0.01	1.09	10.4	0.11	708.5	7.13	0.10	613.5	0.09	**0.11**	40.7	41.1	129.3	4.16	-102.8	1.27
		0.02	1.15	10.0	0.11	577.5	6.83	0.11	570.2	0.09	0.14	37.8	42.3	113.3	4.14	-103.4	1.20
		0.03	1.05	10.1	0.12	598.0	6.48	0.14	487.7	0.09	0.15	36.9	40.1	107.3	4.22	-103.0	1.23
	3.5	0.01	1.04	10.3	0.10	628.9	7.05	0.09	576.8	0.08	0.12	39.3	42.0	118.7	4.05	-102.9	1.23
		0.02	0.93	9.94	0.12	560.6	6.81	0.10	484.6	0.09	0.15	41.5	40.8	104.1	4.05	-103.2	1.19
		0.03	1.11	9.93	0.11	548.2	6.49	0.12	520.3	0.09	0.16	38.6	39.0	105.8	4.16	-103.1	1.23

Table 12: FuncGenFoil model performance with different RBF kernel for conditional airfoil generation.

Modes	l	Label Error ↓ (10^{-3})													$\mathcal{D}$ ↑	$\mathcal{M}$ ↓ (10^{-2})
		σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	$\bar{\sigma}_a$	$\bar{\sigma}_g$		
64	0.001	1.26	**7.60**	**0.09**	1224	5.34	**0.07**	1046	**0.06**	**0.08**	24.9	**28.6**	213	3.68	-103.7	1.42
	0.01	0.77	8.67	0.12	223.6	5.05	0.09	174.7	0.08	0.12	35.2	37.1	47.0	3.06	-103.6	1.03
	0.02	**0.62**	8.33	0.12	162.2	4.59	0.11	165.9	0.09	0.17	38.0	42.5	38.4	3.04	-103.2	**1.01**
	0.03	0.66	7.80	0.12	**147.9**	**4.43**	0.12	**154.2**	0.09	0.15	**33.2**	37.9	**35.1**	**2.93**	-103.6	1.00
	0.04	0.84	8.79	0.13	197.1	4.59	0.12	168.4	0.09	0.17	35.8	37.9	41.3	3.20	-103.2	1.02
	0.06	1.15	9.97	0.15	220.3	6.21	0.14	215.5	0.10	0.16	38.3	44.2	48.7	3.73	-99.9	1.05
	0.12	1.12	10.9	0.17	230.5	7.71	0.17	265.6	0.11	0.18	40.0	42.3	54.4	4.14	**−95.1**	1.08

Table 13: FuncGenFoil model performance with white noise kernel of different noise scale n for conditional airfoil generation.

Modes	n	Label Error ↓ (10^{-3})													$\mathcal{D}$ ↑	$\mathcal{M}$ ↓ (10^{-2})
		σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	$\bar{\sigma}_a$	$\bar{\sigma}_g$		
64	1	1.15	7.18	0.09	913.8	5.16	0.07	964.5	0.06	0.08	26.2	26.1	176.8	3.44	-104.7	1.39
	2	1.46	7.98	0.10	1274	5.94	0.08	1268	0.08	0.10	30.6	32.4	238.3	4.17	-102.7	1.64
	4	2.27	9.14	0.12	2015	7.24	0.11	2059	0.09	0.14	41.5	41.4	379.7	5.61	-99.9	2.09

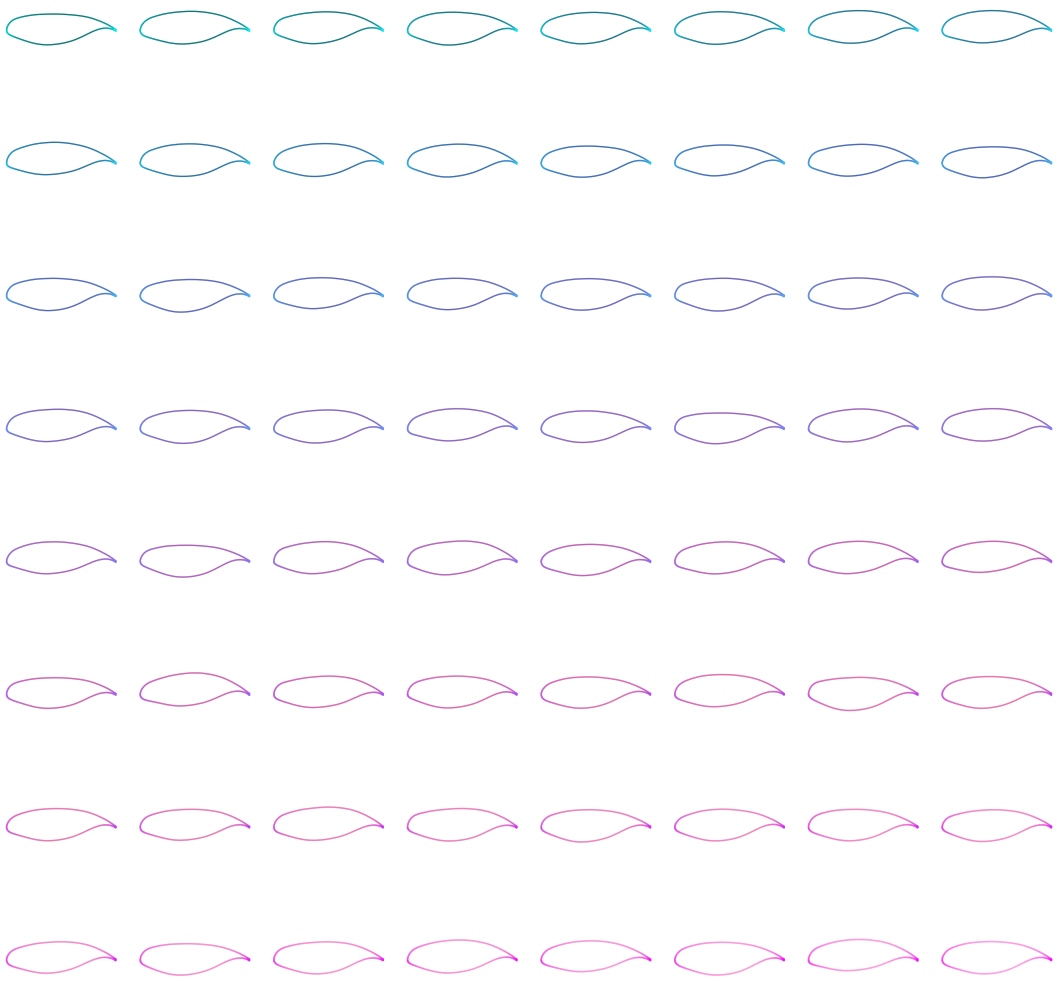


Figure 8: A diverse set of airfoil geometries generated by FuncGenFoil models trained on supercritical airfoil data.