Unlocking the Black Box of Variational Autoencoders for Generative Airfoil Design

A Symbolic Regression Approach to Latent Space Interpretation

Rahiq Ullah

Delft University of Technology

TU Delft

Delft University of Technology
Faculty of Aerospace Engineering

Unlocking the Black Box of Variational Autoencoders for Generative Airfoil Design

A Symbolic Regression Approach to Latent Space Interpretation

by

Rahiq Ullah

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday June 24, 2025 at 15:00 PM.

Student number:	4841867		
Project duration:	October 1, 2024 – June 24, 2025		
Thesis committee:	Ir. J.A. Melkert,	Chair,	TU Delft
	Dr. ir. C. Varriale,	Supervisor,	TU Delft
	Ir. K. Swannet,	Supervisor,	TU Delft
	Dr. N.A.K. Doan,	Examiner,	TU Delft

Cover: AI generated image (pollinations.ai) of aircraft from neural net

TU Delft

Preface

This thesis is the culmination of nearly 7 years as a student in Delft. This represents some of the best years of my life, where I had the opportunity to continuously challenge myself while also meeting wonderful people along the way. From the associations I have been a part of to the privilege of going abroad and serving on AeroDelft's board for a year, being a student provided me with opportunities and experiences I would have never gotten otherwise.

The last 9 months have been almost exclusively dedicated to working on my thesis. Despite it being a generally lonely endeavor, the countless coffee breaks, lunches, and support from fellow friends helped me get through everything, for which I am very thankful.

I am also very thankful to my supervisors, Kilian Swannet and Carmine Varriale, who have supported me throughout this thesis and pushed me to get the best out of myself. Special thanks to Kilian, who was available every week and was always open for a discussion about my work, usually taking much longer than expected due to mutual enthusiasm and interest about the topic. Next to this, I would like to thank Joris Melkert and Anh Khoa Doan for agreeing to be a part of my thesis committee.

Finally, I would like to thank my family who have supported me along the way, during my thesis and beyond. Without the hard work and support of my parents, I would never have gotten the opportunity to be where I am today. And of course my twin brother, Ragib, with whom I could unwind from my thesis work every weekend and even helped me by proofreading my work.

As this chapter closes, I look forward to spending more time with friends and family again and a future where I hope to play an important role in the pursuit of sustainable aviation.

Rahiq Ullah
Delft, June 2025

Abstract

The black-box nature of deep generative models like Variational Autoencoders (VAEs) limits their practical application in engineering design, particularly in aerospace, where interpretability is crucial for reliability and safety. This work investigates the use of Symbolic Regression (SR) to improve the interpretability of VAE latent spaces for airfoil shape optimization. Two approaches are developed: a latent analysis of a trained β-VAE and a novel SR-VAE model that integrates SR directly into VAE training.

The first approach shows that SR can approximate the decoder via analytical equations linking latent variables to geometric airfoil features. This even enabled parametric reconstruction independent of the decoder, though accuracy was limited for airfoils with extreme thickness or uncommon trailing edges.

The second approach investigates several SR integration strategies, with a per-batch method followed by retraining with fixed equations achieving the best balance between generalizability and reconstruction accuracy. The parametric equations of this final SR-VAE show an improvement over those from the latent analysis while preserving, and in some aspects even improving, the generative capability of the decoder. The latent space itself showed limited change due to the use of warm starts, suggesting that interpretability through SR is improved primarily at the output level.

The practical applicability of the decoders and equations obtained in this work are tested and compared to CST parameterization, using inverse design tests, as well as constrained and unconstrained optimization cases. The SR-VAE decoder consistently showed the highest reconstruction fidelity for inverse design, but limited use in practical optimizations due to its poor generalizability far from the training set mean. Although SR-based parameterizations show limited reconstruction fidelity in inverse design, they demonstrate competitive performance and the fastest convergence in optimization tasks.

Overall, this work demonstrates that SR can bridge the gap between black-box generative models and interpretable, equation-based design, opening new pathways for explainable AI in engineering contexts and beyond.

Contents

List of Figures

List of Tables

Nomenclature

Symbols

Symbol	Definition	Unit
$a_j^{(l)}$	Activation value of neuron j in layer l	[-]
$b_j^{(l)}$	Bias of neuron j in layer l	[-]
c_n	Camber-distribution value at chordwise point x_n	[-]
c_{max}	Maximum camber	[-]
C_D	Drag coefficient	[-]
$\bar{C}_D$	Mean drag coefficient	[-]
C_L	Lift coefficient	[-]
C_M	Moment coefficient	[-]
D_{KL}	Kullback–Leibler divergence	[-]
$\boldsymbol{x}$	Input airfoil vector fed to the encoder	[-]
$\boldsymbol{x}'$	Reconstructed airfoil vector from the decoder	[-]
$\boldsymbol{z}$	Latent-space vector	[-]
L_{KL}	KL-divergence loss	[-]
L_{recon}	Reconstruction loss	[-]
L_{SR}	Symbolic-regression loss	[-]
$\ell(\alpha, \beta)$	GAN loss function with generator and discriminator outputs	[-]
$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}$	Evidence Lower Bound (ELBO) objective function	[-]
$p_\theta(x)$	Model likelihood of data sample x given parameters θ	[-]
$p_\theta(x\|z)$	Likelihood of x conditioned on latent z (decoder)	[-]
$p_\theta(z)$	Prior distribution of latent variables	[-]
$q_\phi(z\|x)$	Variational posterior (encoder)	[-]
s	Standard score (z-score, normalised residual)	[-]
t_n	Thickness-distribution value at chordwise point x_n	[-]
t/c	Normalised thickness distribution	[-]
t_{max}	Maximum thickness	[-]
$w_{jk}^{(l)}$	Weight from neuron k (layer $l{-}1$) to neuron j (layer l)	[-]
x	Normalised chordwise coordinate	[-]
x/c	Ratio of chordwise position to chord length	[-]
x_n	Discrete chord position used in SR datasets	[-]
$x_{c\text{max}}$	Chordwise location of maximum camber	[-]
$x_{t\text{max}}$	Chordwise location of maximum thickness	[-]
z_i	i-th element of latent vector z	[-]
Z_i	Latent variable when used explicitly in SR equations	[-]

Symbol	Definition	Unit
α	Angle of attack	[deg]
β	KL-weighting term in a β-VAE loss	[-]
$\boldsymbol{\phi}$	Variational parameters of the encoder network	[-]
$\boldsymbol{\theta}$	Parameters of the decoder network	[-]
δ	Inverse tangent of the camber slope	[-]
γ	Weight on SR loss term during training	[-]
λ	SR-equation complexity regularisation coefficient	[-]
μ	Mean of latent variable distribution	[-]
σ	Standard deviation of latent distribution	[-]
$\sigma(\cdot)$	Activation function (e.g. sigmoid/ReLU)	[-]
θ_{LE}	Leading-edge angle	[deg]
θ_{TE}	Trailing-edge angle	

Abbreviations

Abbreviation	Definition
AE	Autoencoder
ANN	Artificial Neural Network
CST	Class-Shape Transformation
CVAE	Conditional Variational Autoencoder
ELBO	Evidence Lower Bound
GAN	Generative Adversarial Network
KL	Kullback-Leibler
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
PIVAE	Physics-Informed Variational Autoencoder
PySR	Python Symbolic Regression library
ReLU	Rectified Linear Unit
RMSE	Relative Mean Squared Error
SR	Symbolic Regression
SR-VAE	Symbolic Regression-integrated Variational Autoencoder
UIUC	University of Illinois at Urbana-Champaign Airfoil Database
VAE	Variational Autoencoder

1. Introduction

Data-driven methods such as deep learning, are one of the most significant and rapidly growing fields of technology, with their applications becoming increasingly ubiquitous [1]. These methods have opened up new avenues of research and innovation, largely due to their ability to analyze large datasets, make predictions, and provide insights previously thought unattainable [2]. Aerospace engineering is no exception. Brunton et al. highlight the potential of these methods to address high-dimensional, non-convex, and multi-objective optimization problems common in aircraft design, while also emphasizing the challenge of making machine learning models interpretable, generalizable and explainable [3].

One such application is to leverage the power of deep generative models to attain a low-dimensional universal parameterization for airfoils [4, 5]. A deep generative model, such as a Variational Autoencoder (VAE) [6], can extract hidden features and patterns from an airfoil database. Established airfoils, often the result of multidisciplinary optimizations, cluster within a smaller design space than one defined by traditional airfoil parameterizations, such as polynomial-based methods. Machine learning models such as a VAE can exploit this to create a more efficient airfoil representation without sacrificing flexibility [7]. However, a common challenge for these methods is that the low-dimensional representation is abstract and not directly tied to physical space. Due to the complex nature of artificial neural networks, it is difficult to uncover the meaning of the new dimensions within this “latent” space.

In literature, this limitation is often referred to as the “black box” problem. This problem inhibits the overall interpretability, explainability, and reliability of deep learning models, and thereby its usability in engineering. Since airfoil shape optimization is fundamental to aerodynamic design, ensuring the reliability and usability of the model is crucial. Interpretability allows users quantitative and independent control of airfoil features, facilitating knowledge transfer and the imposition of constraints [7]. Although unlocking the black box of deep learning models such as VAEs is an active research area [8, 9], research focusing on interpretability of airfoil-based deep generative models remains in its infancy, where interpretability is often qualitative rather than quantitative.

This is where equation identification methods like Symbolic Regression (SR) show significant potential in interpreting the latent space. SR is another rapidly growing subfield of machine learning, but inherently focuses on physical interpretability by aiming to find symbolic equations that simultaneously fit the data well and are generalizable to uncovered regimes. Existing VAE models can potentially be made interpretable without the need to retrain them, by using SR to generate equations that describe physical relationships as a function of the latent variables. It can potentially even be used to devise a new training procedure, forcing the VAE to create a latent space that is interpretable through simple equations. The use of SR in the interpretation of deep learning models is very rare in deep learning research and is not encountered at all within the context of airfoils.

Hence, this research investigates the feasibility of using SR to improve the latent space interpretability of VAEs for airfoil shape optimization. First, the application of SR to the trained VAE model by Swannet et al. [4, 5] will be studied, examining its ability to extract physical features as well as parameterizations from the latent space. Afterwards, the potential for developing a new VAE training procedure that incorporates SR will be explored. While the application is quite specific, the findings of this research could have broader implications for deep learning models in general, for engineering and beyond.

This report is structured as follows. Chapter 2 contains the literature review, which serves as the foundation of this research. This chapter includes the introduction of important concepts and previous work done within this field. Chapter 3 outlines the research questions and expected results. The methodology of the research phases is detailed in Chapter 4. This is followed by Chapter 5, where the results that follow from the methodology are presented and discussed, including an optimization study. Finally, relevant conclusions will be drawn in Chapter 6, followed by recommendations in Chapter 7.

2. Literature Review

This chapter presents the main results of the conducted literature study. The overall focus of the research is the application of deep generative models to airfoil shape optimization, and the interpretability of the latent space. First, in Section 2.1, literature focused on the fundamentals of deep learning is covered, including artificial neural networks, the learning algorithms and the concept of generative modeling. This is done to create a solid understanding of deep learning, before diving into the more advanced generative models. Section 2.2 discusses such a generative model: the variational autoencoder (VAE). This section starts at the basic autoencoder, and builds towards the disentangled β-VAE, which is the main focus of this work. This is followed by Section 2.3, which covers another generative model: the generative adversarial network (GAN). Afterwards, the challenge of interpreting the latent space of both models is discussed in Section 2.5. Finally, a potential technique to interpret the latent space, called symbolic regression, is further researched in Section 2.6.

2.1. Deep Learning Fundamentals

To understand deep generative models, the fundamentals of deep learning will be briefly explained in this section. For more details, please refer to relevant literature, such as the book by Nielsen [10].

2.1.1. Artificial Neural Networks

Artificial neural networks, henceforth called neural networks, lie at the very basis of artificial intelligence and deep learning. The basic architecture of such a neural network is shown in Figure 2.1.

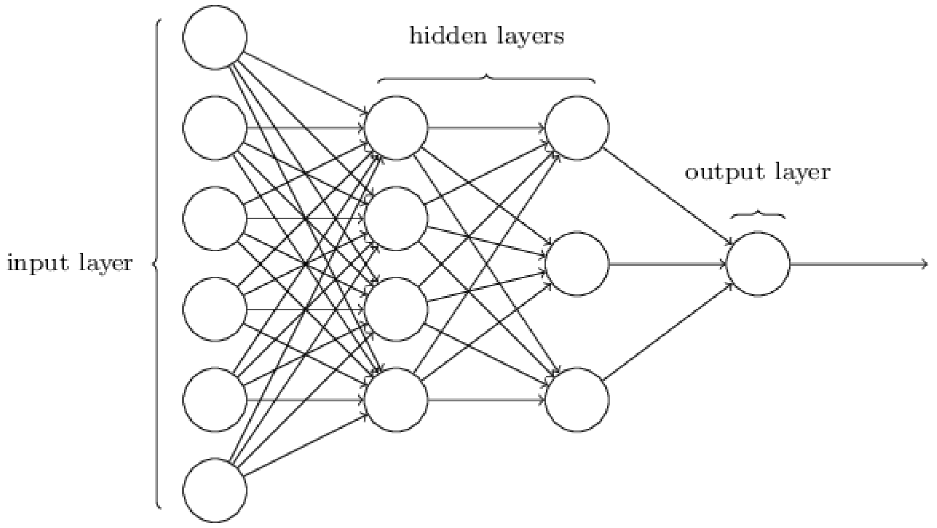


Figure 2.1: Architecture of a artificial neural network [10]

As can be seen in the figure, a neural network consists of an input layer, hidden layers and an output layer [10]. Each layer consists of multiple so-called neurons. These neurons can be active or inactive, acting as simple switches that process and transmit information. The neurons in the input layer and output layer often directly correspond to the input and output data. For instance, if the data consists of images with 784 pixels, every neuron in the input layer will represent a specific pixel. For a discriminative model, the hidden layers are used to process the input neurons. A neuron can be considered a function that yields an activation based on the outputs of all the neurons in the previous layer. Considering a single neuron and its connections would yield the following function:

$$a^{(1)} = \sigma\left(w_1^{(1)}a_1^{(0)} + w_2^{(1)}a_2^{(0)} + w_3^{(1)}a_3^{(0)} + w_4^{(1)}a_4^{(0)} + \cdots + w_n^{(1)}a_n^{(0)} - b_0^{(1)}\right) \tag{2.1}$$

Note that the superscript $^{(0)}$ denotes the layer of the neural network. So the neuron in the subsequent layer (1) will be connected to all neurons with activation values a_n in the previous layer (0), where each one of these connections has their own associated weights w_n and bias b_0. The sigmoid function σ, which is a type of activation function, ensures that the value of the activation falls between 0 and 1. This activation function was especially used in early days of deep learning research, nowadays there are many other activation functions like ReLU and LeakyReLU [11].

To extend this formulation to the complete neural network instead of only a single neuron, the following matrices and equations can be constructed [10]:

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_j^{(1)} \end{bmatrix} = \sigma \left(\begin{bmatrix} w_{0,0}^{(1)} & w_{0,1}^{(1)} & \cdots & w_{0,j}^{(1)} \\ w_{1,0}^{(1)} & w_{1,1}^{(1)} & \cdots & w_{1,j}^{(1)} \\ \vdots & \vdots & \ddots & \\ w_{k,0}^{(1)} & w_{k,1}^{(1)} & \cdots & w_{k,j}^{(1)} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_k^{(0)} \end{bmatrix} + \begin{bmatrix} b_0^{(1)} \\ b_1^{(1)} \\ \vdots \\ b_j^{(1)} \end{bmatrix} \right) \tag{2.2}$$

$$a_j^{(l)} = \sigma \left(\sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)} \right) \quad \rightarrow \quad \boldsymbol{a}^{(l)} = \sigma \left(\boldsymbol{w}^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)} \right) \tag{2.3}$$

Where $w_{jk}^{(l)}$ is the weight from the kth neuron in the $(l-1)$th layer to the jth neuron in the lth layer [10]. Note that the vector symbols are in bold. Every row corresponds to a connection of one layer and a particular neuron and the next layer. Once again, the activation of layer 1 can be computed based on the activation values of the neurons in the previous layer 0, as well as the weights and biases associated with every connection of the neuron with that previous layer.

These neural networks are often called a multilayer perceptron (MLP), but deep convolutional networks (CNN) are often used instead, which is similar but weights and biases are assigned differently [10].

2.1.2. Learning

Within the context of neural networks, learning entails finding the right weights and biases to minimize a cost or loss function. This is often defined as the mean squared error (MSE) [10]:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{b}) = \frac{1}{n} \sum_x \left\| f(\boldsymbol{x}) - \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{b}) \right\|^2 \tag{2.4}$$

Where $\boldsymbol{y}$ is the output vector and $\boldsymbol{x}$ is the data input of the neural network with n defined as the sample size. Through gradient descent, the gradient is evaluated at every point and a step is taken in the direction of steepest descent. A cost function needs a continuous input, which is one of the main reasons why artificial neurons are continuous rather than binary as in biological neurons.

The algorithm that is used to compute the gradient of the cost function is called backpropagation. The complete calculus and derivation behind this technique are considered out of scope for this report. In essence however, backpropagation is the process of computing the partial derivatives of the cost function with respect to the weights and biases. Since these derivatives are in turn influenced by the activation and its associated weights and biases of the previous layer, the algorithm effectively "propagates backwards'. Repeating this process throughout the entire neural network will lead to the desired gradient vector of the cost function $\nabla \boldsymbol{\mathcal{L}}$.

This is essentially what happens when the neural network is trained with all the training data. In practice however, instead of using all the training data every time, the data is randomized and put in batches. The gradient is then computed based on one of these batches. While the gradient will not be in the true direction of steepest descent, it is still a good approximation and more computationally efficient. This process is called stochastic gradient descent (SGD). The learning process is controlled by the so-called hyperparameters, which are parameters such as the number of epochs (or iterations) per training, the size of the mini-batch and the learning rate η. Determining the batch size involves a trade-off between convergence speed and generalization: larger batches typically converge faster but generalize poorly, while smaller batches generalize better at the cost of slower convergence [12, 13]. The learning rate indicates the number of steps taken during optimization [5].

2.1.3. Generative Modeling

In deep learning literature, generative models are treated as generators of new data. However, generative models have a much broader purpose than this and have a distinct advantage of traditional discriminative models. This is schematically shown in Figure 2.2:

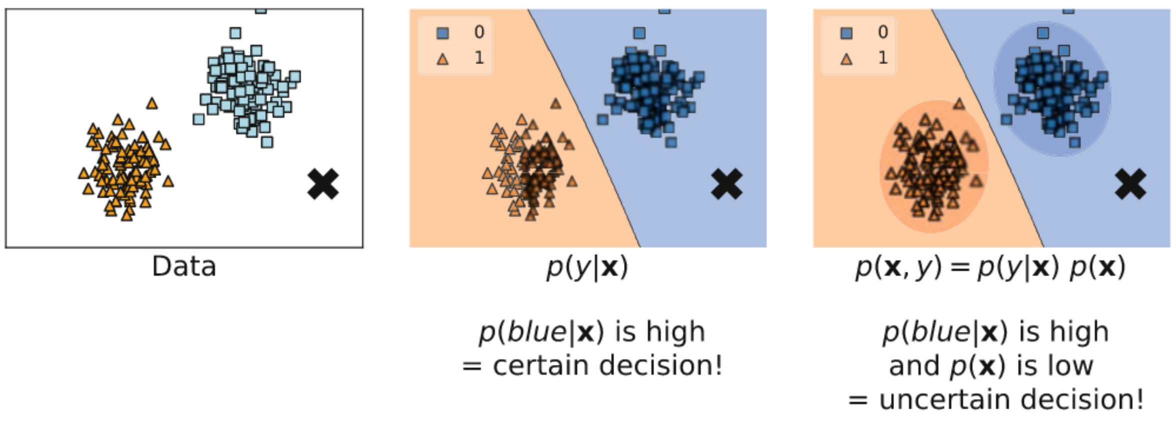


Figure 2.2: Two approaches to classify data (left): discriminative approach (middle) and generative approach (right) [14]

Contrary to discriminative modeling, generative modeling incorporates the understanding of distribution of data, $p(x)$. As can be seen in the figure, the generative model uses this distribution to show that despite the high probability that "X" is blue, it is still an uncertain decision since it is not located near the region of blue data points. The discriminative model does not incorporate this, and is therefore certain of the decision despite the distance to the region of blue data points. This example illustrates that understanding the environment is important to make reliable decisions.

Furthermore, understanding of $p(x)$ can be used to assess whether an object has been observed in the past, it can be used to learn by interacting with the environment and it can lead to a more accurate estimation of a decision's weight. Clearly, generative models have a much broader applicability within AI than only generating data [14]. While in discriminative modeling one aims to learn a predictor based on observations, in generative modeling one tackles the more general task of learning a joint distribution over all the variables [15].

Deep generative modeling can be divided into four main groups: Autoregressive generative models (ARM), Flow-based models, energy based models and latent variable models [14]. This report mainly concerns itself with latter, which includes implicit models like Generative Adversarial Networks (GANs) and prescribed models such as Variational Autoencoders (VAEs).

2.2. Variational Autoencoders

VAEs provide a computationally efficient way of optimizing latent variable models together with a corresponding inference model using SGD [15]. This is further elaborated in this section, starting with the basic autoencoder, building towards a disentangled β-VAE.

2.2.1. The Autoencoder

An autoencoder (AE) consists of two distinct parts: an encoder and a decoder. This is schematically shown in Figure 2.3. The encoder can be considered as a function $z = f(x)$ that maps high dimensional input data x to a lower dimensional embedding vector with hidden variables z. The decoder $x' = g(z)$, reconstructs the data x' by mapping z from embedding space to data space. The embedding space and hidden variables are often referred to as latent space and latent variables respectively [16, 17].

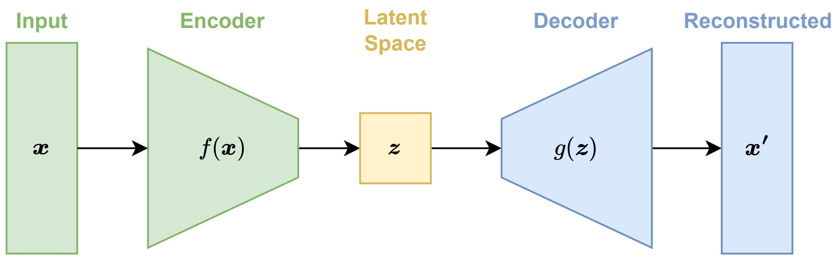


Figure 2.3: Schematic overview of the autoencoder

Real life data is never truly random, and often has some underlying patterns and relationships. The encoder leverages these hidden factors to create a new low dimensional representation of the data in latent space. It can therefore be seen as a discriminative model. The decoder takes this representation and reverses the encoding to generate high dimensional data, effectively making it a generative model.

Autoencoders are usually implemented through neural networks, consisting of the encoder network, decoder network and at least one hidden layer. Especially for fully connected neural networks, these components are usually symmetric in layer shapes [18].

Mathematically, the autoencoder functions can be extended to the following for neural networks [16]:

$$\begin{cases} \boldsymbol{z} = f\left(\boldsymbol{w}_e, \boldsymbol{b}_e; \boldsymbol{x}\right) \\ \boldsymbol{x}' = g\left(\boldsymbol{w}_d, \boldsymbol{b}_d; \boldsymbol{z}\right) \end{cases} \tag{2.5}$$

Here $f(\cdot)$ and $g(\cdot)$ are neural networks, $\boldsymbol{w}_i$ and $\boldsymbol{b}_i$ are the corresponding weight matrices and bias vectors of the encoder and decoder neural network. The autoencoder is trained by minimizing the loss function, this is a version of the MSE defined earlier in equation (2.4):

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} |\boldsymbol{x}_i - \boldsymbol{x}_i'|^2 \tag{2.6}$$

where $\theta = (\boldsymbol{w}_e, \boldsymbol{b}_e; \boldsymbol{w}_d, \boldsymbol{b}_d)$. This optimization can be executed via gradient descent or SGD.

Due to the deterministic nature of the latent space, the AE lacks the capability to reliably interpolate within the input data and generate new realistic data points. While the latent space may be continuous in a mathematical sense, it is typically not regularized or structured in a way that supports meaningful new data points. As such, regular autoencoders are traditionally used for dimensionality reduction or feature extraction rather than generative modelling [16]. To make it a truly generative model capable of generating reliable new data points, the latent space should be represented as a probabilistic model, such that it supports smooth transitions and meaningful sampling across the space.

2.2.2. Variational Inference

The VAE is an extension of AE where a probabilistic framework is introduced to represent the latent space, with the objective to maximize the log likelihood of the data. The problem with maximum likelihood learning in latent variable models like VAEs is that the marginal likelihood of the data under the model shown below is intractable [15]:

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} \tag{2.7}$$

This is where variational inference comes into play. The objective of this technique is to approximate a conditional probability of latent variables given observed variables, with the key idea to solve this through optimization [19]. For the case of VAEs, the parametric inference model $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ can be introduced, with ϕ the parameters of the inference model, commonly called variational parameters. These variational parameters are then optimized such that it approximates the posterior $p_\theta(\boldsymbol{z}|\boldsymbol{x})$: [15]:

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) \approx p_\theta(\boldsymbol{z}|\boldsymbol{x}) \tag{2.8}$$

This can be used for the construction of the objective function from the log likelihood of the data.

2.2.3. ELBO

The optimization objective of the VAE is to maximize the evidence lower bound (ELBO), also called variational lower bound. As will be shown, maximizing the ELBO essentially maximizes the intractable log-likelihood of the data. The log-likelihood of the data and its associated variational lower bound can be derived as shown below [15, 16].

First, the log-likelihood of the data is multiplied by $\int q_\phi(\boldsymbol{z}|\boldsymbol{x})dz = 1$ to introduce the inference model from Equation (2.8). Then, the complete expression is factored into the integral, since $p_\theta(\boldsymbol{x}) \notin \boldsymbol{z}$.

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \int q_\phi(\boldsymbol{z}|\boldsymbol{x})d\boldsymbol{z} \\ &= \int \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) q_\phi(\boldsymbol{z}|\boldsymbol{x})d\boldsymbol{z} \end{aligned} \tag{2.9}$$

Through the definition of expectation $\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x)dx$, where $f_X(x)$ denotes a probability density function of continuous random variable X, and the rearranged definition of conditional probability $p_\theta(x) = p_\theta(x,z)/p_\theta(z|x)$ [20], the equation can be rewritten as follows:

$$\begin{aligned} &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right]\right] \end{aligned} \tag{2.10}$$

Subsequently, the inference model can be introduced again through a whole fraction, and then the logarithm can be split:

$$\begin{aligned} &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right]\right] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right]\right] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left[\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right]\right] \end{aligned} \tag{2.11}$$

Now, with the definition of Kullback-Leibner (KL) divergence D_{KL}:

$$D_{KL}(P\|Q) = \int_{-\infty}^{\infty} p(x)\log\left(\frac{p(x)}{q(x)}\right)dx = \mathbb{E}_{x\sim p(x)}\left[\log\left(\frac{p(x)}{q(x)}\right)\right] \tag{2.12}$$

The following final expression can be defined with the ELBO:

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right]\right]}_{\substack{\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \\ \text{(ELBO)}}} + D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\right) \tag{2.13}$$

KL divergence denotes how much one probability distribution differs from another [17]. It is per definition non-negative $D_{\text{KL}} \geq 0$ and only equal to 0 here if the inference model is equal to the true posterior distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = p_\theta(\boldsymbol{z}|\boldsymbol{x})$. Therefore, ELBO is the lower bound on the data log-likelihood [15]:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) \leq \log p_\theta(\boldsymbol{x}) \tag{2.14}$$

The ELBO can be further analyzed by once again using the definition of conditional probability and splitting the logarithm [6]:

$$\begin{aligned} \mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right)\right] \\ &= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log\left(\frac{p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right)\right] \end{aligned} \tag{2.15}$$

Finally, once again applying the definition of D_{KL}, leads to the following for the ELBO. Note that this the "reverse" KL divergence due to the fraction in the logarithm, which is why the term is negative.

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - D_{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})\|p_\theta(\boldsymbol{z})\right) \tag{2.16}$$

With this final expression, it can be seen that maximizing the ELBO, $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$ with respect to θ and ϕ essentially means:

1. Maximizing the reconstruction likelihood of the decoder (first term)
2. Minimizing the divergence between the learned distribution and the prior belief of $\boldsymbol{z}$ (second term)

Now, to connect this derivation with the architecture of an autoencoder, the following can be defined:

- Encoder: $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, the learned distribution of the model, given $\boldsymbol{x}$ it samples the distribution of $\boldsymbol{z}$.
- Prior: $p_\theta(\boldsymbol{z})$, represents what the desired latent space should look like, usually a Guassian
- Decoder: $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, given latent variable $\boldsymbol{z}$, it generates samples of the distribution $\boldsymbol{x}$.

2.2.4. Reparameterization Trick

The intention is to differentiate and optimize the ELBO $\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x})$ w.r.t. to both variational parameters ϕ and generative parameters θ. But differentiating w.r.t. ϕ leads to a gradient estimator with a too high and impractical variance [6]. Furthermore, it is not possible to propagate back through $\boldsymbol{z}$ since it is a random variable [15]. The solution to these issues is the so-called reparameterization trick, and is schematically shown in Figure 2.4.

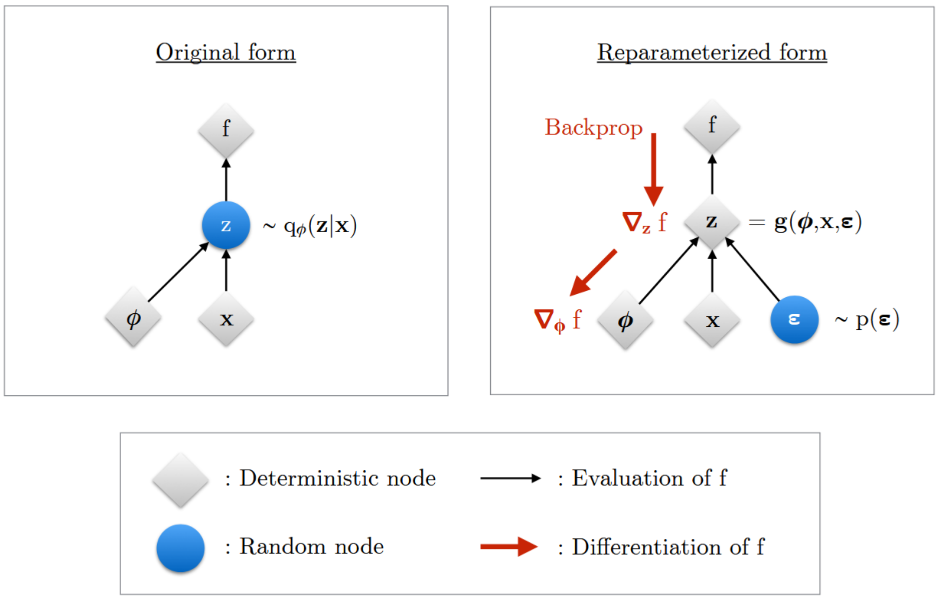


Figure 2.4: Schematic representation of the reparameterization trick, adapted from [15]

The idea is that the random variable can be decomposed such that an independent random variable with a simple distribution can be taken out. This reparameterization makes the variable $\boldsymbol{z}$ a deterministic and differentiable function of ϕ and $\boldsymbol{x}$. Commonly, the following transformation is applied [15]:

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon} \tag{2.17}$$

where $\varepsilon \sim \mathcal{N}(0, \boldsymbol{I})$ is random noise sampled from a Gaussian, μ is the mean value and σ is the standard deviation. This results into the following final architecture of the variational autoencoder:

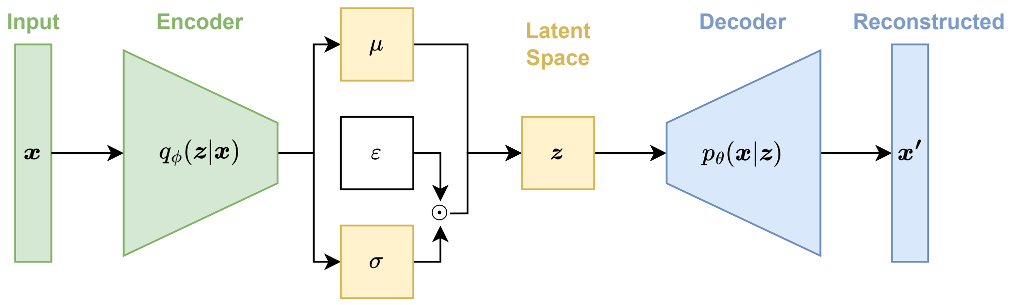


Figure 2.5: Schematic overview of the variational autoencoder

As can be seen by comparing Figure 2.5 with Figure 2.3, the encoded input is mapped into a probabilistic latent space rather than a deterministic vector. The encoder is the parametric inference model $q_\phi(\boldsymbol{x}|\boldsymbol{z})$, that samples $\boldsymbol{z}$ given $\boldsymbol{x}$, and the generative model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ samples $\boldsymbol{x}'$ given $\boldsymbol{z}$. Note that the noise source ε is outside the model, allowing backpropagation through deterministic parameters μ and σ.

2.2.5. Disentanglement

A particularly interesting modification of the VAE is the β-VAE. Here an adjustable hyperparameter β is added to the original objective from Equation (2.16) [21]:

$$\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\boldsymbol{x},\beta) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right] - \beta \cdot D_{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p_\theta(\boldsymbol{z})\right) \tag{2.18}$$

Careful consideration of values for β can result in more disentangled latent representations $\boldsymbol{z}$, where in general a high β is necessary for good disentanglement. Disentanglement means that different, independent characteristics of the data are encoded in separate, distinct dimensions of the latent space. It can essentially be seen as a mixing coefficient that balances the gradient magnitudes of the reconstruction and prior matching components of the ELBO. Unfortunately, there is a trade-off between reconstruction accuracy and disentanglement: good disentanglement often leads to blurry reconstructions while entangled representations lead to the sharp reconstructions [22]. Increasing β gives more weight to D_{KL} term, which forces the learned posterior distribution $q_\phi(\boldsymbol{x}|\boldsymbol{z})$ to be more similar to the prior $p(\boldsymbol{z})$, which in turn forces the latent space to represent the data more independently across its dimensions. Reducing β emphasizes the reconstruction term, allowing the model to have more accurate reconstruction of the input data but weaker regularization. This can lead to overfitting, and reduces the model's ability to generate new outputs. Note that generally, the β-VAE is simply referred to as a VAE. Therefore, unless explicitly mentioned, the VAE can be considered as β-VAE from this point onwards.

2.3. Generative Adversarial Networks

GANs are implicit latent variable models. Like the VAE, the issue of calculating the integral of marginal likelihood occurs (Equation (2.7)). The solution proposed for VAE is to prescribe models for the probabilities and apply variational inference, but another approach used by GANs is to abandon the likelihood-based approach altogether to learn a tractable sample generation process [23].

2.3.1. Architecture

GANs are based on a game or battle between two machine learning models, typically implemented using neural networks [23, 24].

One is called the generator, which defines $p_\theta(\boldsymbol{x})$ implicitly. It is defined by a prior $p_\theta(\boldsymbol{z})$, where the latent vector $\boldsymbol{z}$ functions as the input to the generator function $G_\beta(\boldsymbol{z})$. Here β is a set of parameters that is learned to define the generator's strategy, and $\boldsymbol{z}$ can be considered as a source of randomness. The main goal is to learn the function $G(\boldsymbol{z})$ that can generate realistic samples from the noise $\boldsymbol{z}$.

The other model is the discriminator, which examines samples $\boldsymbol{x}$ and typically returns a probability estimate $D_\alpha(\boldsymbol{x})$ of whether the samples are real, i.e. from the training distribution of the data, or fake, i.e. from the generator. Similarly, α represents the set of parameters that is learned to define the discriminator's strategy. The final architecture is schematically shown in Figure 2.6.

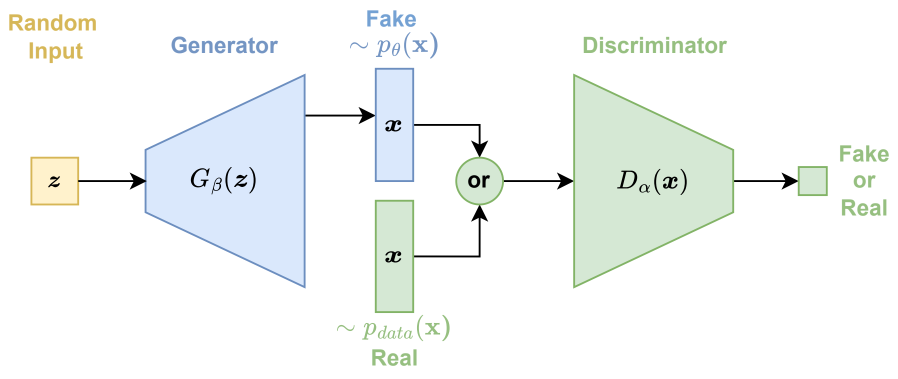

Random Input
Generator
Fake
$\sim p_\theta(\mathbf{x})$
Discriminator
$\boldsymbol{z}$
$G_\beta(\boldsymbol{z})$
$\boldsymbol{x}$
or
$D_\alpha(\boldsymbol{x})$
Fake or Real
$\boldsymbol{x}$
$\sim p_{data}(\mathbf{x})$
Real

Figure 2.6: Schematic overview of the generative adversarial network (adapted from [14])

As can be seen in the figure, the architecture of a GAN is clearly very different from that of a VAE, as was shown in Figure 2.5. As expected, there are no probability distributions or likelihoods, but learned tractable functions like $G_\beta(\boldsymbol{z})$ and $D_\alpha(\boldsymbol{x})$. Note that the generator is very similar to the decoder of the VAE, where a latent vector $\boldsymbol{z}$ is converted to high dimensional data $\boldsymbol{x}$. The main difference here is that the decoder emphasizes a probabilistic structure and interpretable latent space, whereas the generator focuses on realism in the data output.

2.3.2. Loss function

To reiterate, there are two sources of data: fake $\boldsymbol{x} \sim p_\theta(\boldsymbol{x}) = \int G_\beta(\boldsymbol{z})p(\boldsymbol{z})$ and real $\boldsymbol{x} \sim p_{data}(\boldsymbol{x})$. The discriminator assigns the fake data points with 0 and the real data points by 1. Considering the discriminator as a classifier, the following binary entropy loss function can be defined [14]:

$$\ell(\alpha,\beta) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{real}}}\left[\log D_\alpha(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\log\left(1 - D_\alpha\left(G_\beta(\boldsymbol{z})\right)\right)\right] \tag{2.19}$$

Where the first term corresponds to the real data source and the second term contains the fake data source. The objective is to simultaneously maximize $\ell(\alpha,\beta)$ w.r.t. α such that the discriminator is optimized, and to minimize $\ell(\alpha,\beta)$ w.r.t. β such that the generator fools the discriminator as well as possible. This leads to the following objective for the GAN:

$$\min_{\beta}\max_{\alpha} \mathbb{E}_{\boldsymbol{x}\sim p_{\text{real}}}\left[\log D_\alpha(\boldsymbol{x})\right] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\log\left(1 - D_\alpha\left(G_\beta(\boldsymbol{z})\right)\right)\right] \tag{2.20}$$

Here $\ell(\alpha,\beta)$ is referred to as adversarial loss, since two actors have two opposite goals. This adversarial nature ensures that GANs can generate a very realistic outputs, but also make a GAN inherently difficult to train [25].

2.4. Application to Airfoil Parameterization

Airfoil shape optimization is an essential part in the design of wings, blades and propellers. The high dimensional nature of airfoil shapes leads to the need for an efficient airfoil parameterization method. Traditionally, parameterization techniques use a polynomial basis where coefficients can be adjusted to shape airfoils, and have further developed characteristics to meet specific design requirements. Such methods have several advantages: smoothness is ensured in the resulting airfoil shapes, coefficients are relatable to significant physical airfoil features and for methods such as CST and B-splines/Bézier high flexibility is possible by increasing the polynomial degree. However, there is often a trade-off between flexibility and parsimony. This is where data-driven methods such as VAEs and GANs come in, which can extract significant features from historical airfoils to yield a potentially more parsimonious airfoil representation compared to the polynomial approach, without sacrificing flexibility [26].

2.4.1. Generative Adversarial Network Methods

In literature, there are many studies where GANs have been used for airfoil design. Archour et al. [27], Yilmaz et al. [28] and Chen et al. [29] used a conditional GAN (CGAN), where additional physical conditions are fed to the generator next to the random variables, for the inverse design of airfoils [26]. Wada et al. [30] took solving the inverse design problem one step further with the physics guided GAN (PG-GAN), to guide the model to learn physical validity. These models allowed them to generate airfoils with a certain target performance. Other studies did focus more on airfoil parameterization [31–33], where the aim was to derive a low-dimensional representation from high-dimensional airfoil coordinates. The latter is of particular interest for this thesis. However, a key problem in these GAN models is that they often produce infeasible airfoil geometries, which are self-intersecting and lack smoothness [27, 28, 31]. VAE-based models are said to mitigate these problems [26]. This challenge is further addressed through the development of the BézierGAN [31], BSplineGAN [32, 34] and CST-GAN [35]. However, these models still lack in interpretability, limiting the control of physical airfoil features [26].

These are inherent problems that stems from the GAN architecture. The adversarial nature of the model limits the interpretability of the latent space. For this same reason, GANs suffer from unstable training and mode collapse [25, 36]. Since in this thesis the overall objective is on latent space interpretability, the focus is shifted towards VAE models in particular.

2.4.2. Variational Autoencoder methods

There have been several studies where VAEs have been used to construct a new airfoil parameterization. In general, this can be divided into parameterization based on regular disentangled VAEs [4, 5, 26, 37] and VAEs where some kind of conditions are imposed, such as conditional VAEs (CVAE) and physics informed VAEs (PIVAE) [7, 38–43]. Unless specified otherwise, the UIUC Airfoil Coordinate Database [44] is used to train these models.

VAE

Swannet et al. [4, 5] trained a disentangled VAE, where a composite Bézier/B-spline curve is used to smooth out the airfoils from the decoder output. To leverage the inherent feature extraction capabilities of a VAE and minimize user-imposed bias on the inferred latent space, the authors made a conscious decision to not enforce interpretability through user-defined conditions on the latent vector [4]. Through a Pearson correlation matrix, it was observed that the max thickness t_{max} and max camber c_{max} are captured most effectively by the latent parameters, and that a total of six latent variables are needed for the airfoil parameterization. While the paper provides a flexible and parsimonious representation of the airfoils with the capability to generate novel smooth airfoils, the latent space is still not entirely disentangled and interpretable.

Kang et al. [26] employed a different approach, where two separate VAEs were trained, explicitly separating the thickness and camber distribution of the airfoils. This allowed independent control of both and completely eliminated the possibility of generating self-intersecting airfoils. Just like the VAE from Swannet et al. [4, 5], the VAE is trained through coordinate points from the UIUC database. However, rather then using B-spline as post-processing step, the authors restricted the output space to a B-spline layer to ensure smoothness. Once again, t_{max} and c_{max} were dominated by independent latent variables, but these dominant variables also influenced other geometric airfoil properties. This indicated that the latent variables were not completely disentangled and interpretable, due to their coupling to other physical features.

CVAE

A way of getting physically decoupled variables in the latent space is through a CVAE. Yonekura and Suzuki [40] implemented this using NACA airfoils as training data to obtain airfoils for a specified lift coefficient C_L. While this did enable more direct control of C_L, there was still a discrepancy observed between the specified and reconstructed C_L, which was attributed to improper encoding of some airfoil shapes leading to inaccurate reconstructions.

The same authors [41] also used a hyperspherical CVAE, $\mathcal{S}$-CVAE, which uses a vMF distribution rather than a Guassian, and compared it with the regular $\mathcal{N}$-CVAE. Both were trained with NACA and

Joukowski airfoils. It was observed that $\mathcal{S}$-CVAE outperforms the $\mathcal{N}$-CVAE when applied to one type of airfoil since it separates the data in the latent space, but fails to output novel airfoils where the features are mixed for this same reason. Furthermore, the same discrepancy between the specified and reconstructed C_L was observed.

Xie et al. [42] proposed two generative schemes: a "soft constrained" CVAE and a VAE which was "hard constrained". The soft constrained scheme is a CVAE that is constrained by a user imposed condition passed to the encoder and decoder, as usual. The hard constrained scheme is a VAE where a hard projection module ensures that output airfoil is geometrically mapped to a shape that strictly satisfies the user imposed condition. The models are trained with the UIUC database and tested with NACA 4 airfoils (many of which are also present in UIUC) and a supercritical airfoil dataset (not in UIUC). The soft-constrained scheme generates airfoils with geometric features that slightly deviate from what was imposed, but all the airfoils are smooth and reasonable. On the other hand, the hard constrained scheme generates some airfoils which are too unrealistic to be considered, but generates airfoils with more diversity and strictly satisfies the constraints.

Wang et al. [39] proposes a method that is shown to outperform both the hard and soft constrained model with respect to MSE, and demonstrated a better congruency between the specified and reconstructed C_L. Two CVAE-based airfoil generation models were developed, proposing the airfoil freedom design (AFD-CVAE) and airfoil precision design model (APD-CVAE). The former accepts a single input condition and the latter incorporates more input conditions and constraints. For the training data, a VAE was trained on the UIUC database to generate a dataset of $200,000$ airfoils, called Airfoil-VAE. Furthermore, to enable the use of pressure distribution as a condition, the pressure coefficients C_{P_i} are fed through an AE, such that the latent variables can be used as an input to the encoder and decoder in the CVAE. The model shows very good agreement between the imposed and reconstructed pressure distributions. The reason this model outperforms other CVAE models is likely due to the extended UIUC training dataset.

PIVAE

While the CVAE models had some promising results, they were designed for inverse design problems rather than an interpretable latent space. Li et al. [43] did focused on both, proposing the physically interpretable variational autoencoder (PIVAE) and comparing it with a similarly trained CVAE and VAE. The architecture of both the PIVAE and CVAE is shown in Figure 2.7.

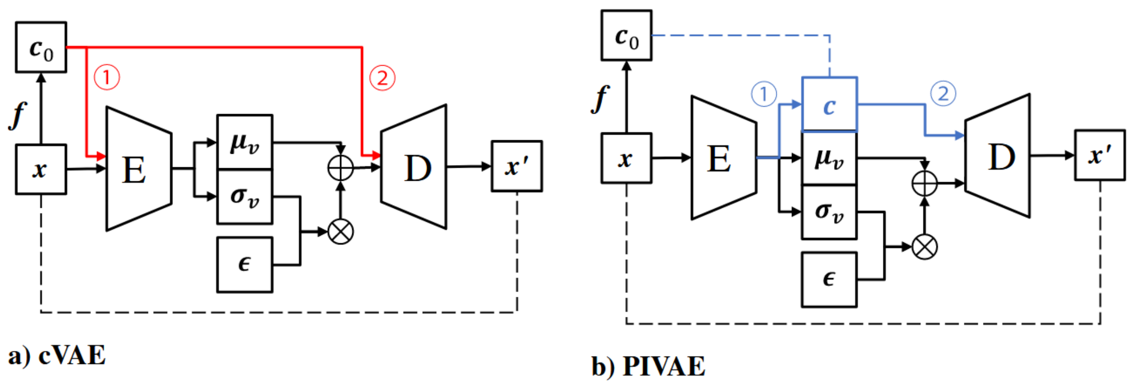


Figure 2.7: The architecture of the CVAE and the PIVAE [43]

The architectures are very similar, but rather than feeding the user defined physical features c_0 to the input of the encoder and decoder, the PIVAE uses part of the encoder output, the physical code c, as decoder input and to represent c_0. Furthermore, an additional MSE_{phys} loss term is added to encourage that $c = c_0$. The models were trained with supercritical airfoil data obtained by perturbing the CST coefficient of three baseline airfoils, leading to 1000 samples for testing and $10,000$ samples for training. It was shown that the PIVAE has the best disentanglement between the physical code and data features, while the latent variables of the VAE were shown to be entangled. The performance and disentanglement of the PIVAE and CVAE is similar when the training set has uniformly distributed labels $p(\mathbf{c_0})$, but both models deteriorate when the label distribution is random.

Kang et al. [7] improved on their earlier work [26] by proposing a novel physics aware variational autoencoder for their camber and thickness VAEs. The authors essentially take the PIVAE from Li et al. [43], but use physical features from the generated sample rather than the training sample. Furthermore, the loss function is slightly changed by introducing a novel technique called "latent sampling regularization" to the MSE$_{\text{phys}}$ term, which essentially entails that random samples are taken into account in the loss function to enforce the physical conditionals. This ensures that more realistic samples can be generated from latent sampling. The adjusted architecture is shown in Figure 2.8.

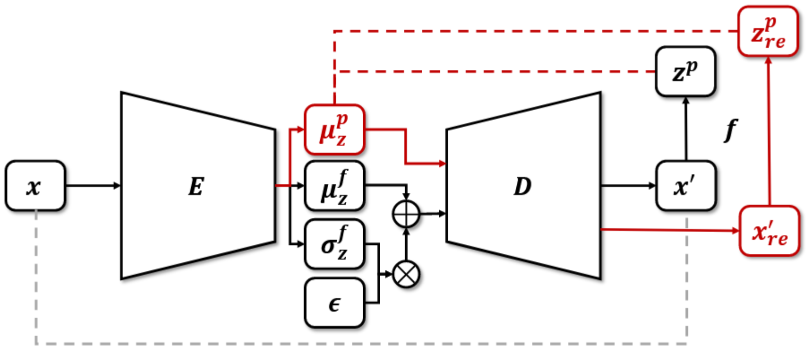


Figure 2.8: Architecture of the physics aware VAE [7]

Here, the subscript re denotes resampling. Just like the PIVAE, the latent representation is separated into "physical" (^{p}) and "free" (^{f}) latent dimensions, where the former is directly mapped to geometric properties like max camber m_{max}, trailing edge direction γ_{TE}, mac thickness t_{max} and leading edge radius r_{LE}. Here it was observed that for the case with latent regularization, the physical latent dimensions aligned very well with the physical features of the airfoil. Without the extra regularization, only the trained samples align well but the random samples deviated significantly. This shows that the earlier issues identified for the PIVAE are alleviated by latent sampling regularization. The result is a latent space representation where each physical latent parameter controls the physical properties of interest. It has been tested in multiple airfoil shape optimizations, and has been shown to outperform traditional parameterization methods.

2.4.3. Future research

While the physics aware VAE seems very promising, there are some distinct areas that warrant further research. As mentioned by the authors themselves, as of yet it is unclear which physical airfoil features are optimal for training the VAE [7]. Furthermore, it could be interesting to see if this novel physics aware VAE architecture performs well in a configuration where only one model is trained, rather than two separate models. This could reduce the complexity and thereby increase the reliability of the VAE model. Another aspect that warrants further research is using Bézier/B-spline line compositions as an input to the model, rather then restricting the output space or using it as a post-processing step. This could help retain the inherent symmetry of the VAE architecture and could make the latent space easier to interpret to a possible parameterization. Furthermore, similar to what Wang et al. [39] did, more data can be used to improve the overall performance of VAEs. Finally, it could also be interesting to research other techniques that do not impose conditions on the VAE to interpret the latent space, such that user defined bias is minimized.

2.5. Latent Space Interpretation

As expected from the field of deep generative modeling, latent space interpretation efforts in literature are usually applied to images. There is a close relationship between disentanglement and interpretability. Disentanglement aims to learn underlying generative latent factors of the data (called factors of variation) and encode them in distinct latent representations [45]. Disentangled variables are expected to contain interpretable semantic information and reflect disjoint factors of variation in the data [46]. Due to this close correlation between disentanglement and interpretation, this section contains both techniques that pursue disentanglement and techniques that aim to tackle interpretation more directly. There is a particular focus on unsupervised methods, since the goal is to explore methods that do not impose conditions on the latent space. Overall, these can be divided into approaches that make adjustments to the loss function and those that make changes to the model architecture.

2.5.1. Manipulation of loss function

Burgess et al. [21] proposed a modification to the training regime of the β-VAE from Subsection 2.2.5, such that disentanglement can be improved without losing too much information of the original data. This is done through the introduction of the capacity C, which is subtracted from the KL divergence term. This C is gradually increased during training from zero to a value large enough to produce good quality reconstructions, effectively forcing disentanglement from the start. This is proven to promote robust learning of a disentangled latent space, as well as a higher reconstruction fidelity compared to the original β-VAE.

Kumar et al. [47] proposed the Disentangled Inferred Prior VAE, or DIP-VAE. The inferred prior is denoted as $q_\phi(\boldsymbol{z}) = \int q_\theta(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$, which should be factorizable along the dimensions for disentanglement, or $q_\phi(z) = \prod_i q_i(z_i)$. Rather than adjusting the D_{KL} regularization term, in this model an extra regularization term was added that represents the distance $D(\cdot \parallel \cdot)$ between the inferred prior $q_\phi(\boldsymbol{z})$ and the disentangled generative prior $p(\boldsymbol{z})$, to facilitate this factorization. It is shown that through the explicit minimization of this term, better control over the disentanglement of the latent space is achieved.

Kim and Mnih [48] did something very similar by proposing the FactorVAE, where a term is added that represents the KL divergence between the inferred prior $q_\phi(\boldsymbol{z})$ and its factorized approximation where each latent variable is independent $\bar{q}(z) := \prod_{j=1}^{d} q(z_j)$. In other words, a total correlation (TC) term is added, which evaluates the dimension-wise independence of $\boldsymbol{z}$ [49]. One of the limitations of this model is that low TC is necessary but not sufficient for complete latent space disentanglement. This is exemplified for the case where all latent dimensions collapse to the prior but one, where the TC would be 0 but the representation would not be completely disentangled [48]. The main difference with DIP-VAE [47] is that it focused on penalizing covariance between latents by matching it to a prior, but did not constrain the mean or higher order moments.

Chen et al. [46] aimed to minimize TC as well, but rewrote the original regularization term rather than introducing a new term. The authors introduced the Total Correlation Variational Autoencoder, or β-TCVAE, as a refinement and plug-in replacement of the β-VAE from Subsection 2.2.5. Here, the KL-divergence term from Equation (2.18) is further decomposed into three terms. One of these is the TC, a measure for the independence between two or more random variables. This forces the model to find latent factors in the data distribution that are statistically independent. In the new loss equation, the hyperparameter β is now the weight associated with this TC term. This model was shown to outperform β-VAE and infoGAN, but its performance was shown to deteriorate when the there is a sampling bias.

With the Relevance Factor VAE (RF-VAE), Kim et al. [50] built on the other disentanglement approaches to enable the distinction between relevant and irrelevant nuisance factors. Once again starting with a TC based derivation for disentanglement, the model implements a relevance factor formulation to identify two groups of factors from data at the same time as the VAE model parameter estimation. For synthetic, controlled datasets the RF-VAE outperformed both the FactorVAE and the β-VAE for all tested datasets. However, similar the disentangled VAEs, discrete factors have an adverse effect on the model's overall performance. Furthermore, the formulation is a lot more extensive than the commonly used β-VAE, which could limit the practicality of using it as a replacement.

The VAE disentanglement methods described here are all unsupervised and make adjustments to the regularization term. A summary is shown in Table 2.1, as adapted from Wang et al. [8]. Note that he definition for TC is applied for conciseness: $TC(q(\boldsymbol{z})) = D_{\text{KL}}(q(\boldsymbol{z}) \parallel \prod_{j=1}^{d} q(z_j))$ [51].

Method	Regularizer	Description
β-VAE	$-\beta D_{\mathsf{KL}}\left(q_\phi(\boldsymbol{z}\|\boldsymbol{x})\\|p(\boldsymbol{z})\right)$	β controls the trade-off between reconstruction fidelity and the quality of disentanglement in latent representations.
Understanding disentangling in β-VAE	$-\gamma\left\|D_{\mathsf{KL}}\left(q_\phi(\boldsymbol{z}\|\boldsymbol{x})\\|p(\boldsymbol{z})\right)-C\right\|$	The quality of disentanglement can be improved as much as possible without losing too much information from the original data by linearly increasing C during training.
DIP-VAE	$-\beta D_{\mathsf{KL}}\left(q_\phi(\boldsymbol{z}\|\boldsymbol{x})\\|p(\boldsymbol{z})\right)-\lambda D_{\mathsf{KL}}\left(q_\phi(\boldsymbol{z})\\|p(\boldsymbol{z})\right)$	Enhance disentanglement by minimizing the distance between $q_\phi(\boldsymbol{z})$ and $p(\boldsymbol{z})$. In practice, we can match the moments between $q_\phi(\boldsymbol{z})$ and $p(\boldsymbol{z})$.
FactorVAE	$-D_{\mathsf{KL}}\left(q_\phi(\boldsymbol{z}\|\boldsymbol{x})\\|p(\boldsymbol{z})\right)-\gamma\, TC(q_\phi(\boldsymbol{z}))$	Directly impose independence constraint on $q_\phi(\boldsymbol{z})$ by penalizing the total correlation $TC(q_\phi(\boldsymbol{z}))$
β-TCVAE	$-\alpha I_q(\boldsymbol{z};\boldsymbol{x})-\beta\, TC(q(\boldsymbol{z}))$ $-\gamma\sum_j D_{\mathsf{KL}}\left(q(z_j)\\|p(z_j)\right)$	Decompose $D_{\mathsf{KL}}\left(q(\boldsymbol{z})\\|p(\boldsymbol{z})\right)$ into three terms: i) mutual information, ii) total correlation (TC), iii) dimension-wise KL divergence, and penalize them respectively.
RF-VAE	$-\sum_{j=1}^{d}\lambda(r_j)D_{\mathsf{KL}}\left(q_\phi(z_j\|\boldsymbol{x})\\|p(z_j)\right)$ $-\gamma TC(q(\boldsymbol{r}\circ\boldsymbol{z}))-\eta\\|\boldsymbol{r}\\|_1$	Introduce relevance indicator variables $\boldsymbol{r}$ by only focusing on relevant parts when computing the total correlation $TC(q(\boldsymbol{r}\circ\boldsymbol{z}))=D_{\mathsf{KL}}\left(q(\boldsymbol{r}\circ\boldsymbol{z})\\|\prod_{j=1}^{d}q(r_j\circ z_j)\right)$, penalizing $D_{\mathsf{KL}}\left(q(z_j\|\boldsymbol{x})\\|p(z_j)\right)$ less for relevant dimensions and more for nuisance dimensions.

Table 2.1: Summary of Disentanglement Methods in Variational Autoencoders (adapted from [8])

2.5.2. Change in architecture

Ding et al. [52] attempts to make the latent representation semantically interpretable by introducing the Guided-VAE. While both a supervised and unsupervised variant is proposed, only the latter will be considered here. In the unsupervised Guided-VAE, a secondary decoder is added to the model architecture to guide the training of the VAE. This decoder is a deformable principle component analysis (PCA), which is essentially a simple linear version of an AE. A corresponding PCA loss term is added to the VAE ELBO, which effectively encourages dimensions of the latent space to behave like principal components, promoting disentanglement and interpretability. The model is shown to achieve a comparable disentanglement to the β-TCVAE. However, the combination of both methods, Guided-β-TCVAE, was shown to outperform VAE, β-VAE, Factor-VAE and β-TCVAE.

Other techniques that aim to make the latent space more interpretable usually pertain to using labeled data and through the imposition of conditions, such as [53, 54]. These are purposefully excluded, as such techniques have already been applied to airfoils in literature [7, 39–43].

2.6. Symbolic Regression

A rapidly growing subfield within machine learning is symbolic regression (SR), where there is a key focus on interpretability [55, 56]. This method describes a supervised learning task where the model space comprises of analytical methods, here it searches the space of simple analytical expressions for accurate and interpretable models through a multi-objective optimization [57]. Furthermore, while SR is a data-driven method, it aims to find a symbolic model that simultaneously fits the data well and is generalizable to uncovered regimes. SR models are even said to be be used to “white box” a “black box” model such as a neural network [56].

This makes it a very promising field of research in the pursuit of latent space interpretability. Therefore, this section goes through the different SR methods and shortly evaluates its potential for this purpose. An additional section is included to cover literature in which SR has already been used to interpret latent spaces. The taxonomy for the different types of methods is adapted from Makke et al. [56], from which a summary of the methods is shown in Figure 2.9. A living review of all SR methods can be found on a Github page by the same authors[1].

[1]https://github.com/nmakke/SR-LivingReview

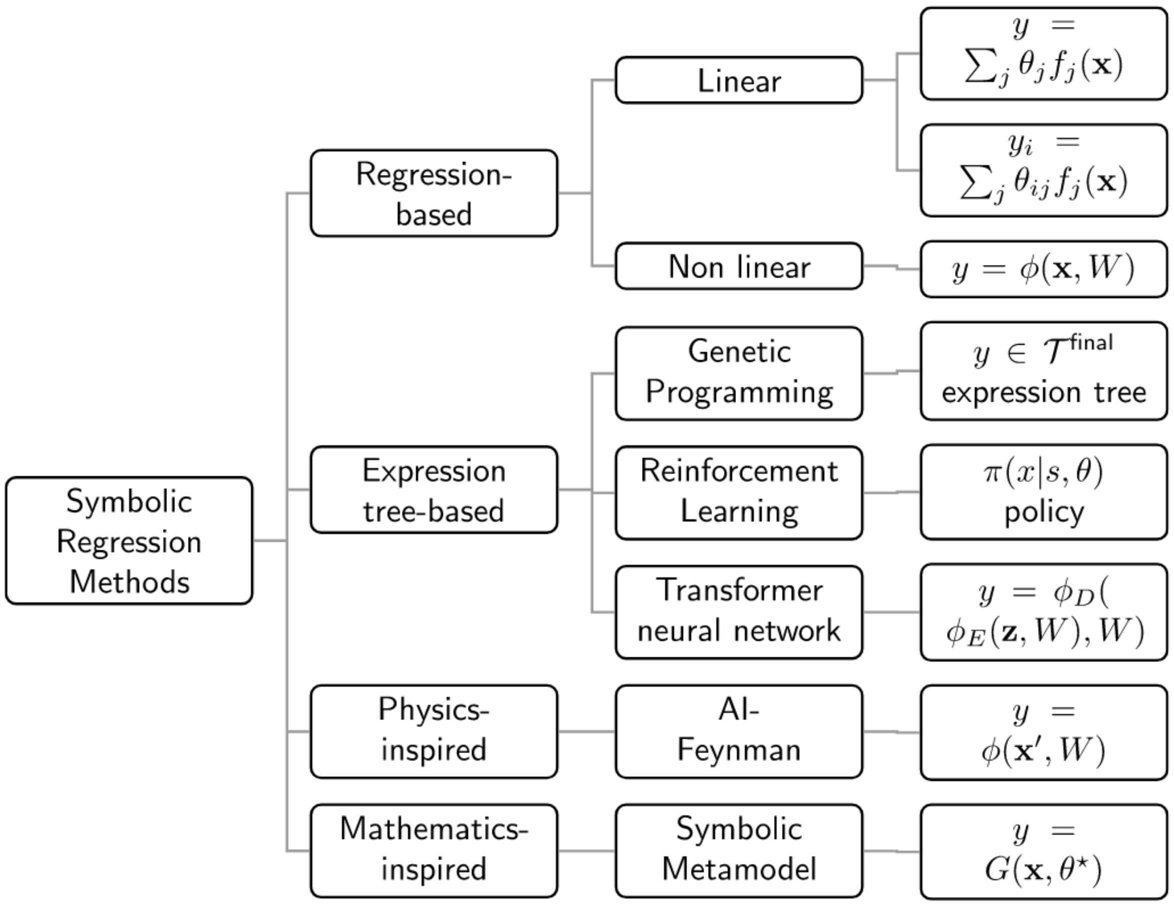

Symbolic Regression Methods
Regression-based
Linear
$y = \sum_j \theta_j f_j(\mathbf{x})$
$y_i = \sum_j \theta_{ij} f_j(\mathbf{x})$
Non linear
$y = \phi(\mathbf{x}, W)$
Expression tree-based
Genetic Programming
$y \in \mathcal{T}^{\text{final}}$ expression tree
Reinforcement Learning
$\pi(x|s,\theta)$ policy
Transformer neural network
$y = \phi_D(\phi_E(\mathbf{z}, W), W)$
Physics-inspired
AI-Feynman
$y = \phi(\mathbf{x}', W)$
Mathematics-inspired
Symbolic Metamodel
$y = G(\mathbf{x}, \theta^\star)$

Figure 2.9: Summary of SR methods [56]

2.6.1. Regression-based

Regression-based SR methods can in turn be divided into linear and non-linear methods.

Linear

Linear SR models are expressed as a linear combination of non-linear functions $h(\boldsymbol{x})$ that are contained in a predefined library L [56]:

$$f(\boldsymbol{x}) = \sum_j \theta_j h_j(\boldsymbol{x}), \qquad h_j(\cdot) \in L \tag{2.21}$$

where $\boldsymbol{x}$ is the input vector, θ_j is a weight coefficient and $h_j(\cdot)$ is a unary operator of L. This reduces the SR problem to only learn parameters of a system of linear equations. If $f(\boldsymbol{x})$ is a linear combination of monomials, it would essentially be a conventional linear regression problem, but L can span a wide range of base functions. The main advantage of this method is that it is deterministic. However, it forces the model structure to be a linear combination of non-linear functions, which limits its applicability to more complicated problems. This means that it is not applicable to composition of functions (e.g. $f(x) = f_1(x) \cdot f_2(x)$), multivariate functions (e.g. exp$(x \cdot y)$ or tan$(x + y)$) and functions with multiplicative or additive factors to its arguments (e.g. exp(λx)) [56].

Currently, state of the art linear SR methods are SINDy, developed by Brunton et al. [58], and SINDy-AE by Champion et al. [59], where the acronym stands for Sparse Identification of Non-linear Dynamics. Both are SR methods designed to learn dynamical systems and governing equations. As the name implies, SINDy uses sparse regression to find the fewest terms required from a library of base functions to model the dynamics. SINDy-AE combines this method with an autoencoder network to allow the simultaneous discovery of coordinate transformations to facilitate the sparse regression [59]. The autoencoder essentially generates a set of reduced coordinates from high dimensional data which can be reconstructed to the full system. A major caveat in this method for latent space representation is that it is designed to model dynamical system, with the inherent assumption that derivatives $\dot{x}(t)$ of the original state are available or can be computed.

Non-linear

Non-linear SR models define the model structure through a neural network, as defined in Equation (2.3). The weights $\boldsymbol{w}$ and biases $\boldsymbol{b}$ are learnable parameters and σ denotes the activation function from the library L. To ensure its interpretability, sparsity is enforced by adding regularization terms. The problem with this method is that backpropagation through some activation functions requires simplification to the search space, which means that it cannot generate simple expressions involving divisions [56].

The most common non-linear SR framework is the Equation Learner (EQL) proposed by Martius and Lampert [60]. The EQL network is in many ways equivalent to a MLP, but rather than denoting one activation function per layer, in EQL each node has a specific activation function from a library L. To solve the aforementioned issue of generating simple expressions for divisions, Sahoo, Lampert, and Martius [61] introduced an extension called EQL$^{\div}$. However, here exponential and logarithm activation functions are not included because of numerical issues [56]. Once again, the applicability is quite limited when this method is considered for latent space interpretability. Furthermore, since the main goal is interpretability, the use of a neural network which requires its own regularization terms for interpretability seems counterproductive.

2.6.2. Expression tree-based

Expression tree-based methods represent mathematical expressions as unary-binary trees where the nodes are denoted as operators (e.g. $+, -, *, \sin, \log$) from a predefined library and terminals as variables or constants [56]. This type of SR method encapsulates genetic programming (GP), reinforcement learning (RL) and transformer-based models.

Genetic programming

GP is a so-called evolutionary algorithm, and is the most widely adopted method for SR [62]. It starts with a randomly generated "population" (set) of "individuals" (trees) that is evolved through "transition rules" (operations) [55, 56]. The main evolutionary operations are mutation, crossover and selection, as shown in Figure 2.10. As can be seen from the figure, mutation involves introducing random variations to an individual, effectively replacing a subtree by a randomly generated one. Crossover on the other hand, entails the exchange of content between individuals, e.g. switching a random subtree of one individual with one of another individual. Finally, the selection operation selects which individual of the current population can continue to the next [56].

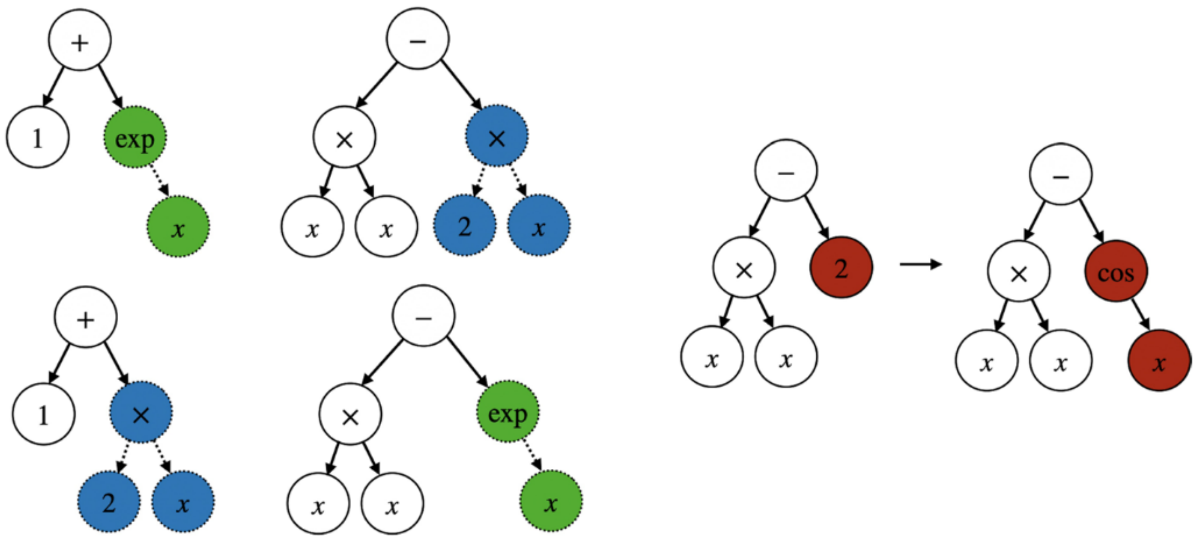


Figure 2.10: GP operations, including crossover (left) and mutation (right), adapted from Makke et al. [56]

During an equation search, each possible operation is scored by the number of data it can handle, the one that is most likely to contribute to the final equation is evaluated on its error. Through continuous examination of the effect of every operation, the process discovers promising candidates with small errors that are used in future operations, while those with poor performance are abandoned. Reasons for termination could be a user-defined maximum step (limit to created populations) and/or a set convergence error. Common metrics for GP-based SR methods are MSE or sum of the squares (SSE), which is identical to MSE but excludes taking the mean [55]. The main advantage of GP based SR is that it allows for large variations in the population, which results in improved performance for out-of-distribution data. Furthermore, due to the nature of the method, GP-based algorithm are not limited to produce one equation as it can be defined to export multiple equations with a range of complexities [55]. However, it is important to note that a low complexity might indicate poor error performance, while a high complexity is prone to overfitting [63]. Furthermore, GP methods do not scale well to high dimensional data sets and are very sensitive to its hyperparameters [64]. Finally, due to the randomness involved in the GP operations, this type of SR is inherently stochastic. This means that two SR models with identical model parameters do not guarantee the same result, which can have important implications for tuning.

A historically commonly used SR framework is Eureqa based on an approach by Schmidt and Lipson [65]. New SR techniques are often compared to Eureqa [57, 64, 66], and it is even referred to as the gold standard of symbolic regression [64]. However, this is a commercial product and said to be fallen out of common use in sciences [57]. As such, Cranmer [57] developed a fast, easy to use and open-source SR framework called PySR. This framework is shown to have more desirable features for scientific equation discovery than various SR tools including Eureqa. Some of these features include denoising, feature selection, custom operator types and custom losses. Since its release, there have been multiple successful applications, such as Grundner et al. [67] who used it to discover a symbolic parameterization for cloud cover and Wong et al. [68] who used it to discover interpretable population models of gravitational wave sources. Furthermore, it is shown to outperform other SR methods in rediscovering physical relations, including EQL and other open source GP-based methods like Operon by Burlacu et al. [69] and GPlearn [70]. The features and its performance make PySR seem like a very suitable candidate to interpret latent space. The main caveat of this method is one that all GP based methods share, which is its inability to handle high dimensional data.

Reinforcement Learning

RL is a machine learning method that aims to learn a policy $\pi(x|\theta)$ by training an agent to execute a task through interaction with the environment in discrete timesteps [56]. It requires four components: state space, action space, state transition probabilities and a reward. In the RL process, the agent selects an action that is sent to the environment. Then, a reward and a new state is sent back to the agent by the environment, which is then used by the agent to improve its policy in the next timestep. In RL based SR methods, the symbolic expression represents a state, the prediction of an element in the sequence is an action, the parent/sibling represent the environment and the reward is commonly defined as the MSE. Usually, these types of methods are a hybrid of RL and other machine learning tools like some kind of neural network [56]. Note that while RL is a machine learning method, it is not data-driven in the same sense as VAEs since it does not require a pre-defined dataset but rather learns dynamically through interaction, exploration and feedback.

In literature, a well known RL based SR framework is Deep Symbolic Regression (DSR), developed by Petersen et al. [64]. It essentially uses RL for SR with the idea to use a large model to search the space of small models. More specifically, a recurrent neural network (RNN) is used to generate a probability distribution over possible mathematical expressions. Expressions are sampled from this distribution, instantiated and evaluated based on their fitness to the dataset. The fitness is used as a reward to train the RNN using a risk-seeking policy gradient algorithm. During training, the RNN adjusts the likelihood of an expression relative to its reward, allocating higher probabilities to better expressions [64]. DSR is shown to outperformn Eureqa as well. However, Cranmer showed that PySR outperforms DSR, and also has more integrated features such as denoising.

Transformers

A transformer is a type of neural network architecture introduced by Vaswani et al. [71] in natural language processing (NLP), and is based on the so-called attention mechanism. This mechanism assigns weight to different parts of the input based on the context, rather than processing the entire input uniformly. A transformer has a encoder-decoder architecture that inputs a sequence of embeddings x_i and outputs a context-dependent sequence of embeddings y_i, through latent representation z_i. In NLP, his input sequence can be e.g. a sentence for the translation of a word where the noun and adjective can be assigned a significant weight. In an SR context, the input data points and mathematical expressions are encoded as sequences of symbolic representations. The encoder only has numeric representation and the decoder has a mixture of symbolic and numeric representations. The purpose of the transformer is to create dependencies between numerical and symbolic sequences and between tokens of the symbolic sequence [56].

There are two types of transformer based approaches for SR: the skeleton approach, such as NeSymRes [72] and SymbolicGPT [73], and the end-to-end (E2E) approach employed by E2ESR [74]. The skeleton approach consists of two steps. First, the decoder uses a function class $\mathcal{F}$ to predict a so-called skeleton, which is parametric function that describes the general shape of the target expression. Second, the constants are fitted through optimization techniques. For example, if $f = \cos(2x_1) - 0.1\exp(x_2)$, then the decoder predicts $f_e = \cos(\circ x_1) - \circ\exp(x_2)$ where $\circ$ is a constant [56]. In the E2E approach, both the skeleton and numerical constants are simultaneously predicted. Thus both symbolic representations of the operators/variables and numeric representations for the constants are used [56]. It is shown that the E2E approach performs well compared to other SR methods outperforms the skeleton approach. However, once again Cranmer [57] has shown that PySR outperforms E2ESR in their equation rediscovery benchmark, and has many more desirable SR features.

2.6.3. Physics-inspired

As the name implies, physics-inspired SR methods leverage physical features to optimize SR. In literature, there is mainly one prominent method in this category developed by Udrescu and Tegmark, called AI Feynman [66]. In essence, it is an multidimensional recursive algorithm that combines the use of a neural network with physics-inspired techniques. These techniques rely on simplifying properties common in physical functions such as: units, low-order polynomial fit, compositionality of small elementary functions, smoothness, symmetry and separability. As such, the AI Feynman algorithm consist of a series of modules that attempt to exploit these properties during its equation search. First a dimensional analysis is attempted, if necessary it continues to search for a suitable equation via a polynomial fit. In case that does not work either, the SR model employs a brute-force approach where it simply tries all symbolic expressions within some class, in order of increasing complexity. This is similar to the GP based methods, where the problem of overfitting occurs in high complexities. If neither of these modules work, a neural network is trained to find simplifying intrinsic properties in the data, such that the dataset is divided into more manageable sub-problems.

The authors constructed the Feynman SR database, and showed that all the basic physical equations and 90% of the more complex bonus equations was solved by the algorithm, outperforming Eureqa [56]. While AI Feynmann performs well with noise, the authors argue that GP-based methods perform even better. Furthermore, unlike GP based methods, Feynman only outputs one equation in its final result instead of a full Pareto frontier [66]. In Cranmer's PySR paper, there is no direct performance comparison, although it is shown that PySR has more desirable SR features [57]. The main limitation here is that this SR method is designed for physical equations, while latent space interpretations are not necessarily physically interpretable. However, since it is a popular easy to use method that is fundamentally different from other SR methods, it can still be interesting to look more into this.

2.6.4. Math-inspired

Makke et al. [56] lists one purely math-inspired SR model, the symbolic metamodeling framework by Alaa and Schaar [75]. The mathematical Meijder G-function is used in this method, the exact math behind this function is considered out of scope for this literature research. This method is fundamentally different from the other listed methods, since it is defined as a general methodology to interpret predictions through converting “black-box” models into “white-box” functions that are interpretable. So it inputs a black-box model and outputs a symbolic expression. Nevertheless, it still shown to reconstruct equations well using real and synthetic data, outperforming a GP-based SR method [75]. However, it is unclear from the paper which GP-based method it is compared to.

Rather than an established SR tool, Alaa and Schaar [75] describe a new framework to obtain an interpretable equation from a black-box model such as a neural network. This makes it less easy to use than some of the other methods listed here for symbolic regression. However, the promise of converting a black-box model to a white-box function is certainly very interesting for latent space interpretability, and will therefore be further discussed in the next section.

2.6.5. Application to interpretability

As of yet, the use of symbolic regression for latent space interpretability is very rare in scientific literature. This subsection will investigate three papers where SR is used in some way to interpret a latent space.

Symbolic metamodel

The symbolic metamodeling framework by Alaa and Schaar [75] is already discussed in the previous section from an SR perspective. However, its main purpose is to obtain an interpretable function from a black-box model such as a neural network. As the name of this method imples, it is a model of a model, i.e. a surrogate model of a trained machine learning model that consists of interpretable mathematical functions. To test its performance, the authors compared it to other model explanation methods, and is shown to perform competitively [75]. However, this framework is specifically built to interpret the entire black-box model by a single equation, using the same dimensions as the input data as can be seen in Figure 2.11.

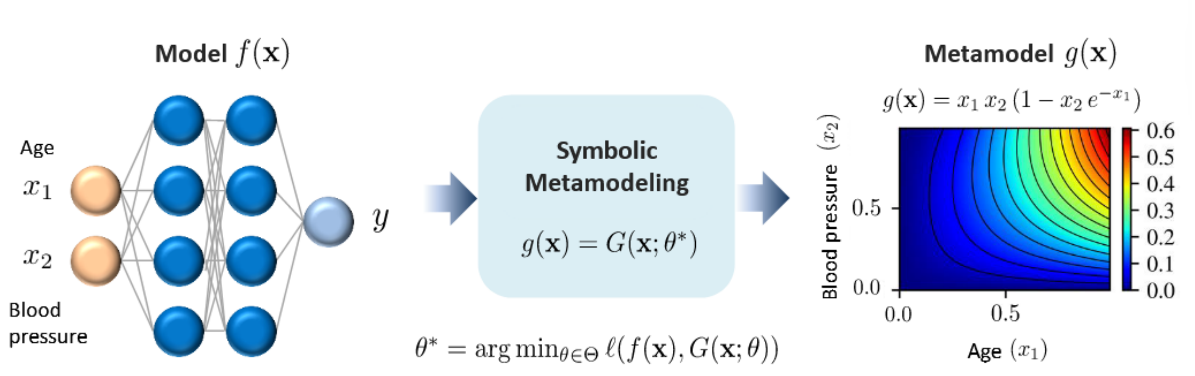

Model $f(\mathbf{x})$
Age
x_1
x_2
Blood pressure
y
Symbolic Metamodeling
$g(\mathbf{x}) = G(\mathbf{x}; \theta^*)$
$\theta^* = \arg\min_{\theta \in \Theta} \ell(f(\mathbf{x}), G(\mathbf{x}; \theta))$
Metamodel $g(\mathbf{x})$
$g(\mathbf{x}) = x_1 x_2 (1 - x_2 e^{-x_1})$
Blood pressure (x_2)
Age (x_1)
0.0
0.5
0.6
0.5
0.4
0.3
0.2
0.1
0.0

Figure 2.11: Schematic depiction of the symbolic metamodeling framework [75]

The main reason for interpreting the latent space is to leverage its efficient low dimensional representation. Since this model obtains an interpretable expression based on the complete model with high dimensionality rather than the low dimensional latent space, it is deemed unsuitable for latent space interpretability. Even though this model is not designed for this purpose, further research into the working of this method could reveal a way to use this model for latent space.

Learning Kondo physics using VAE and SR

Miles et al. [76] employ VAEs to extract physical insight from a dataset of spectral functions. Similar to the application of VAE to airfoils, the authors found a strong correlation between learned latent variables and known physical characteristics. Therefore, SR is used to to model the latent variable as a function of physical input parameters, essentially “rediscovering” equations for the physical characteristics that relate to the latent variables. The paper demonstrates a complete pipeline to train interpretable VAEs, while using SR to discover analytical expressions describing the captured features of the latent space in terms of known physical quantities. The main results include physical equations that model the latent variables which agree very well with actual physical expressions [76].

This approach seems closest to the desire to interpret VAE latent space for generative airfoil design. However, a key difference here is that SR is used to model every latent variable individually as a function of physical parameters, rather than modeling physical parameters as a function of the latent variables. Finally, the paper uses GPlearn to conduct SR, which is shown to perform worse than PySR. It could be interesting to take inspiration from this approach but rather use PySR and to model physical characteristics as a function latent variables.

Symbolic Gradients

Patel et al. [77] aims to retrieve meaningful concepts from latent spaces in a interpretable way, without prior knowledge. The method can be employed to interpret any single neuron within an neural network in closed form. It is based on the same interpretation framework that forms the backend of many other symbolic regression methods such as Eureqa [65], Operon [69], GPlearn [70] and PySR [57].

It is important to note that the method has only been tested on Siamese networks, not on (variational) autoencoders. Furthermore, the method is explicitly developed for a situation where there is no prior knowledge on what physical concept might correlate to the latent variables. If latent space encodings are approximately linearly correlated with known concepts, the authors argue that other SR methods might be more suitable [77]. For the airfoil application, there are already desirable characteristics that one might want to extract from the latent space. This makes the method less suitable for this specific use case. However, the method can come in handy if there are latent variables that do not seem strongly correlated to a known physical concept.

2.7. Knowledge gap

Currently, most efforts to interpret the latent space in the application to airfoils center around changing the inherent VAE architecture in some way [7, 39–43]. Another avenue is to disentangle the latent space through a manipulation of the loss function. In the context of airfoils, a common way to get an interpretable latent space is through the implementation of the β-VAE [4, 5, 26, 37], but many alternative ways to enforce disentanglement and interpretability in VAEs exist [46–48, 50]. All these methods center around some kind of manipulation of the regularization term to ensure that the latent space is disentangled during training. However, as of yet, no method exists that improves interpretability by extracting physical relationships from pre-trained VAE architectures.

SR, a subfield in machine learning with a focus on physical interpretability, shows promise in achieving exactly that. It is commonly used to rediscover physical equations based on real life data and find hidden patterns. This could potentially be used with data generated from the latent space. However, in general the use of SR to interpret latent spaces is very rare, let alone in the application to airfoils. Alaa and Schaar [75] set out to make a black box model interpretable through an SR method called the symbolic metamodelling framework, but it is designed to interpret a complete model such as a neural network rather than the latent space. Patel et al. [77] did explicitly aim to interpret latent spaces, but the method is designed to interpret a single neuron without prior knowledge of possible physical characteristics. In the specific use case of airfoil latent spaces, desired physical characteristics are usually known and are likely a function of multiple latent variables. The use of SR for latent space interpretability that resembles the specific use case the most, is implemented by Miles et al. [76]. Here, SR is used to physically interpret the latent space of the trained VAE. However, it is used to model an individual latent variable as a function of physical parameters, rather than the other way around. At the time of writing, no prior research has been conducted that aims to interpret deep generative models through the identification of physical equations as a function of latent variables.

Therefore, there seems to be a clear knowledge gap, in which the power of SR can be used to assign an understandable physical interpretation to a latent space. In the context of airfoils, it can be used to extract relationships like thickness and camber from latent spaces of existing VAEs, without the need to retrain them. Optimizing deep generative models can be costly, it has even been shown that a latent representation solely optimized for reconstruction typically has a higher reconstruction accuracy than one where interpretability is optimized for as well [78]. SR can potentially even be used to devise a new training procedure, with the focus on obtaining physically interpretable equations. It can be executed at every epoch for multiple physical relationships, forcing the VAE to contain latent variables that can describe physical relationships.

3. Research Plan

Based on the conducted literature review and the identified knowledge gap, this chapter describes the research plan of the overall thesis. First, the main research question and the subsequent subquestions are formulated in Section 3.1, followed by the expected results of the thesis in Section 3.2.

3.1. Research Questions

The main goal is to leverage the power of SR to facilitate interpretability in the latent spaces of VAEs, such that it can be used in engineering applications such as airfoil shape optimization. Based on this and the identified knowledge gap, the main research question is formulated:

Research Question

How can Symbolic Regression (SR) be used to improve the latent space interpretability of Variational Autoencoder (VAE) models for airfoil shape optimization?

To answer this research question, several subquestions are formulated below.

3.1.1. Obtaining physical relationships

There are multiple ways in which SR can be used to obtain interpretable expressions from the latent space. For instance, SR can be used to obtain physical characteristics as a function of the latent variables, latent variables as a function of physical characteristics as done my Miles et al. [76] or even a parameterization as a function of the latent variables.

Subquestion 1

What role can SR play in deriving physically meaningful relationships from the latent space of trained VAEs?

3.1.2. Incorporating SR in VAE training

The VAE framework can be changed in various ways to improve the interpretability of the latent space, as shown in Table 2.1. However, these mainly focused on disentanglement through the inclusion of TC and KL divergence terms. So far, SR has not been used at all for VAE training. Adding such a term can help to force simple interpretable equations from the start. It can even lead to disentanglement if the equation identification is restricted to using one latent variable to describe a certain physical property.

Subquestion 2

How does incorporating Symbolic Regression (SR) into the training process influence the interpretability of the latent space in Variational Autoencoders (VAEs)?

3.1.3. Application to airfoil shape optimization

The initial motivation for using deep generative models was to obtain a low-dimensional representation for airfoils that can improve the multidisciplinary shape optimization. To assess the final model, it can be compared to other VAE-based airfoil representations and conventional airfoil parameterization methods.

Subquestion 3

How does a VAE framework incorporating SR perform in airfoil shape optimization compared to other airfoil representations?

3.2. Expected Results

Through the use of SR, interpretable equations are expected that describe airfoil properties as a function of the latent parameters. PySR is assumed to be the best performing SR method, but if it performs poorly, alternative methods can be considered. The quality of the regression is expected to increase with better disentanglement of the VAE model. Therefore, the traditional VAE model is expected to have the least accurate interpretable equations to describe the latent space.

If SR is incorporated into the training of the VAE, the resulting equations are expected to be simpler and more accurate. The final expected result is a VAE model with an interpretable latent space, with equations that provide a direct mapping to physical characteristics. However, it is expected that the interpretable equations obtained through SR will never be a perfect representation of the latent space but rather an approximation. This could limit the performance in airfoil shape optimization. Nevertheless, it is expected to be more efficient than conventional airfoil parameterization methods due to the low dimensionality of the latent space and the smaller design space.

Finally, the analytical equations obtained through SR are expected to offer more versatile performance than the VAE, which is constrained by its learned latent distributions and tends to degrade in accuracy when extrapolating beyond regions close to the training data mean.

4. Methodology

This chapter contains the methodology to answer the research questions identified in the research plan. First, a general overview of the method will be provided in Section 4.1, to provide clarity on how the method is structured. This is followed by a detailed breakdown of the steps taken for the latent analysis of the β-VAE in Section 4.2. With the lessons learned from this research stage, the method of implementing SR in VAE training will be discussed in Section 4.4. Note that the process behind the optimization study to assess the applicability of the SR equations will be discussed in Chapter 5.

4.1. Overview of Method

As mentioned in Chapter 3, the primary goal is to leverage the power of SR to facilitate an interpretable latent space that can be used in airfoil shape optimization. The process to achieve this is divided into three stages as shown in Figure 4.1, which resembles the research subquesions defined in the research plan. The idea behind these stages is to start with the simplest application of SR, where increasingly more complex and involved steps are taken to facilitate interpretability of the latent space of the VAE.

First, the GP-based SR method by Cranmer, PySR [57], will be used to analyze the latent space interpretability of the base β-VAE model by Swannet et al. [4, 5]. Although a disentangled latent space certainly helps, the use of SR does not require complete disentanglement. The existing Pearson correlation matrices will be used as a reference during the equation identification process. Through this latent analysis, the aim is to obtain interpretable equations for the airfoil properties as a function of the latent variables with the highest correlation. These equations will be assessed and tested for a wide range of airfoils. Afterwards, the training set is adapted to facilitate SR equation identification for thickness and camber distributions. These parameterization equations are subsequently used for airfoil shape parameterization. The performance of this parameterization will be assessed in detail, for a wide range of airfoils. At the same time, different metrics will be assessed such that they can be used throughout the course of the thesis to quantify the performance of the different SR equations.

In the next stage, a new way of training VAEs will be investigated where a novel term incorporating SR will be added to the loss function to force interpretability during training. After a successful implementation, SR will once again be applied and the disentanglement performance will be assessed. SR will once again focus on parameterization equations, since all other geometric or aerodynamic properties can be extracted from the final airfoil shape. Furthermore, using SR to reconstruct airfoils enables seamless integration of SR within the training function. The tuning of the SR model will heavily draw on the results from the latent analysis of the first research phase. The equations and overall interpretability of the latent space will be compared with the results of the latent analysis and the original VAE model.

Finally, the VAE models integrated with SR will be compared with traditional airfoil parameterizations. This will give an indication of the usability and potential of using deep generative models in airfoil design. This is covered after the results of the latent analysis and SR training integration.

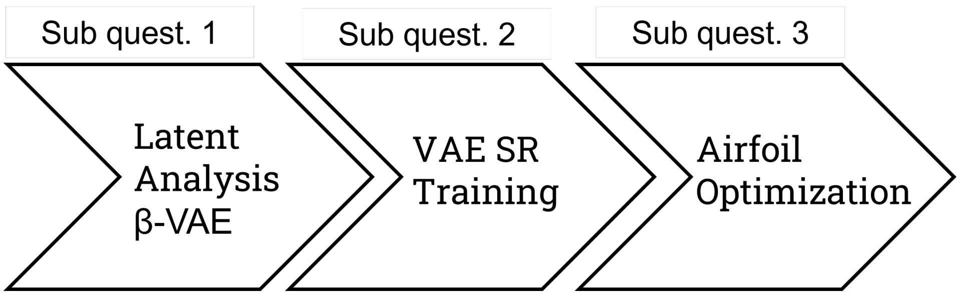


Figure 4.1: Timeline of overall methodology

4.2. Latent Analysis of β-VAE

As described in the Section 4.1, the first step in leveraging SR for latent space interpretability is to apply it to the original β-VAE model by Swannet et al. [4, 5], henceforth referred to as the original VAE. This will assess the ability of SR to interpret existing trained VAE architectures. Note that only the maximum thickness and camber properties are considered, since this is shown to have the strongest correlation in Figure A.1 and because the ability to estimate these features is indicative of the parameterization potential of SR. First, the overall implementation of SR is discussed, followed by the data generation approach from the VAE model. Subsequently, the PySR model tuning process and model evaluation strategy will be outlined. Finally, the parameterization method will be briefly discussed.

4.2.1. SR Implementation

An overview of how SR is applied for the latent analysis of the original VAE is shown in Figure 4.2.

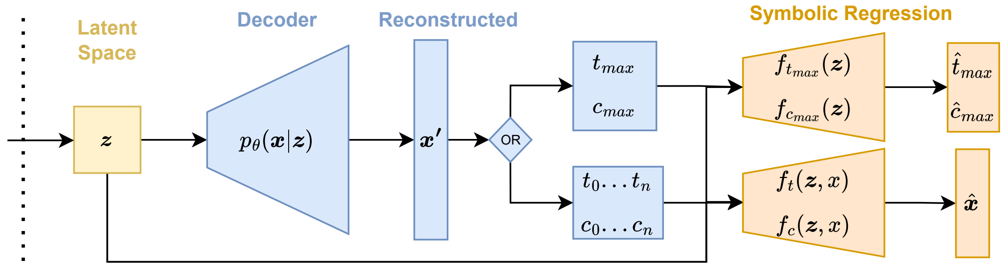


Figure 4.2: Schematic overview of latent analysis of β-VAE for feature identification and parameterization

The figure illustrates the two latent analysis strategies used in this work. In both, latent vectors z are sampled from the latent space and passed through the decoder to generate reconstructed airfoils x'.

In the feature identification branch (top), key geometric features — the maximum thickness t_{max} and camber c_{max} — are extracted from the reconstructed airfoil x' and then expressed as symbolic functions of the latent variables z using SR, i.e., $f_{t_{\text{max}}}(z)$ and $f_{c_{\text{max}}}(z)$.

In the parameterization branch (bottom), the full thickness and camber distributions are extracted at multiple chord locations x, yielding $t_0 \ldots t_n$ and $c_0 \ldots c_n$. SR is then used to identify parametric functions $f_t(z, x)$ and $f_c(z, x)$ that describe the distributions as functions of latent variables and chord location.

Together, these analyses provide interpretable, equation-based approximations of the decoder, either in terms of discrete features or continuous distributions.

4.2.2. Data Generation

As shown in Figure 4.2, input data must be obtained with the various latent variables and the associated airfoil features. PySR can then generate equations that use the former to approximate the latter.

Structure

For feature identification, the input data structure per airfoil is shown in Table 4.1.

Table 4.1: Structure of SR input data for a single airfoil or row, used for feature equation identification

Latent Variables								Airfoil Features			
z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	t_{max}	$x_{t_{max}}$	c_{max}	$x_{c_{max}}$

The input data contains rows with the latent variables and features, each corresponding to an airfoil shape. To obtain parameterizations using SR, this structure has to be adjusted. Multiple thickness and camber values are evaluated at a series of x locations, such that the equations derived from SR can be a function of the latent variables and chord locations x. The adjusted structure is shown in Table 4.2.

Table 4.2: Structure of SR input data for a single row or airfoil, used for parameterization equation identification

Latent Variables								Chord	Distribution	
								x_0	t_0	c_0
z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	$\vdots$	$\vdots$	$\vdots$
								x_n	t_n	c_n

The same cosine-spaced chordwise positions used in the VAE training are used for x_n. Compared to Table 4.1, it can be seen that the airfoil data vector is duplicated to allow the definition of t_n and c_n at multiple chord locations x_n. However, rather than using all the x locations, random x_n locations are sampled for each airfoil with $n = 10$ to limit the size of the training set. If n is set too high, the variation in x would dominate the training data, leading to equations that are only a function of the chord.

Latent variables

To obtain the latent variables, first the UIUC database is passed through the encoder of the VAE model and reparameterized. The reparameterization trick is not strictly necessary, since using the mean should suffice, but has been applied to be consistent with the decoder input. Subsequently, the mean μ, standard deviation σ, and variance σ^2 are obtained from the latent vector for every dimension Z_i. The resulting inferred latent distributions, which contain the probability distribution of every latent dimension with the associated mean, standard deviation, and variance, are shown in Figure 4.3.

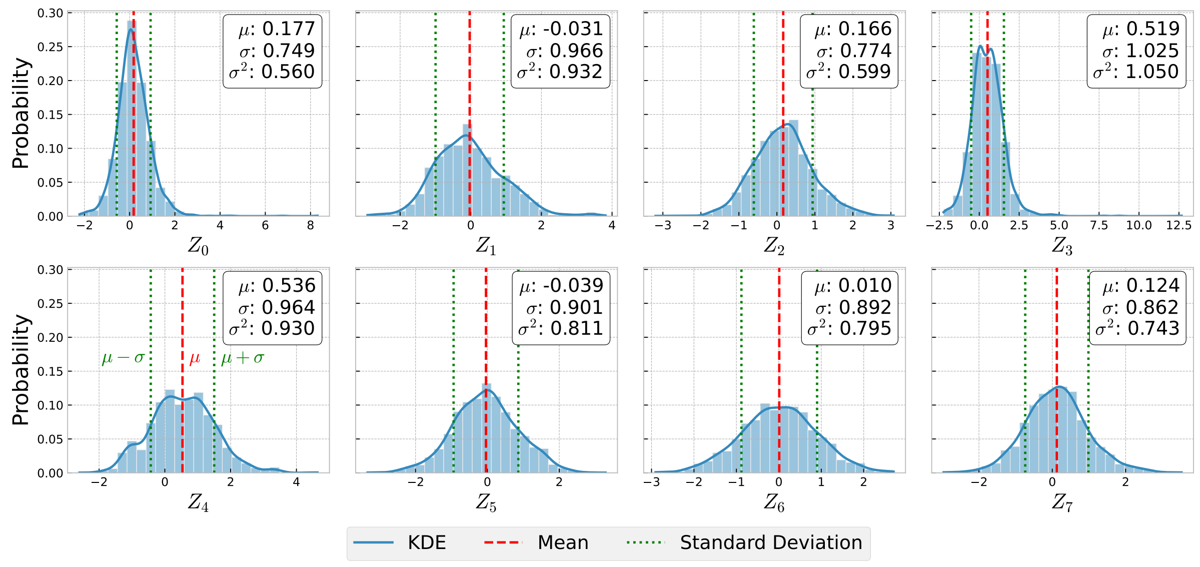


Figure 4.3: Inferred latent distributions from UIUC dataset [4]

As can be seen in the figure, the latent distributions are shaped as Guassians, which is to be expected since this resembles the prior of the β-VAE, where during training the KL divergence between the latent space and prior is minimized. It should be noted that the authors indicated that the decoder of the VAE produces the smoothest geometries when latent parameters are close to the mean, which should be approximately 0, since these correspond to the most common features in the UIUC dataset [4].

Encoding the complete UIUC dataset is useful to generate a test set, but is inefficient as input for SR and leads to a potential danger of overfitting. Therefore, established μ and σ are used for sampling in a range of two standard deviations $\mu \pm 2\sigma$. This can be done by traversing the latent space, i.e. varying one latent variable at a time from left to right tail with a fixed step size, or varying all variables simultaneously sampled from a probability distribution such as uniform or normal at a fixed sample size.

Airfoil features

During the training of the original β-VAE, the UIUC data was standardized by fitting all the airfoil coordinate sets with a B-spline, which is then evaluated to provide upper and lower surface coordinates at 100 fixed cosine-spaced x locations [4]. Therefore, the input and output of the VAE are only the y coordinates of these standardized airfoils.

To obtain the airfoil features associated with every latent vector, the latent vector is fed to the decoder to obtain the coordinates. The complete set of (x, y) coordinates is then once again fitted with a B-spline to ensure smooth airfoils. The geometric properties are then extracted from this processed shape.

4.2.3. PySR settings tuning

PySR is highly configurable and allows for variation in many hyperparameters, each affecting the end result in its own way. While not strictly a tuning parameter, the format of the input data has a significant effect on the performance of the model as well. The complete set of tuned settings for both the feature and parameterization equation search is shown in Table 4.3.

Table 4.3: Tuned PySR settings for feature extraction and parameterization

Setting	Feature	Parameterization
Sampling method	combined	combined
Populations	24	24
Number of iterations	500	500
Binary operators	$*, +, \wedge$	$*, +, \wedge$
Unary operators	—	sin
Complexity constr.	$\wedge : 1$	$\wedge : 1$
Nested complexity constr.	—	sin : $\{$sin : $0\}$
Max equation complexity	20	40
Equation selection	—	—

Note that only settings that are different from PySR's default are covered. This section explains the tuning process and presents a short rationale behind these tuned hyperparameters.

Training data format

To determine the format of the training data, three sampling options are tested: latent traversal data, uniform distributed data and a combination of the two. Latent vectors sampled from a normal distribution have also been considered but were quickly ruled out due to poor performance. After more than 150 runs, the combination of latent traversal and uniform distributed sampled data exhibits the most versatile performance for both t_{max} and c_{max}. This combination ensures that SR captures the effect of an individual latent variable and the random cross coupling of multiple latent variables.

For latent traversal sampling, each of the 8 latent parameters was varied within a range of $\mu \pm 2\sigma$ with 50 steps, resulting in a total of 400 latent vector samples. An equal number of latent vectors were obtained by uniform random sampling of 400 latent vectors. This leads to a final training data size of 800. This is in accordance with the Tuning and Workflow Tips section of PySR's documentation[1], where Cranmer states that training sets that are too large (>1000) should be subsampled or used in batches. While a smaller training set size has been shown to work as well, it degrades the stability of PySR's genetic algorithm and therefore leads to more inconsistent and less reproducible equations.

Populations

The number of populations define how many subpopulations evolve in parallel during the search. This is set to 3 times the number of cores, once again in accordance with advice from the documentation[2].

[1] `https://astroautomata.com/PySR/tuning/`
[2] See footnote 1

Number of iterations

The number of iterations was set to 500 to balance computation time and equation quality. Beyond 500 iterations, the quality of the SR equations did not improve significantly and too few iterations led to inconsistent results.

Operators

PySR allows for the definition of two types of operators: binary and unary operators, where the former takes two arguments and the latter takes one argument. Increasing the number of operators exponentially increases the search space, which is why the operators are mainly limited to those found in the NACA equations [79]. The operators and associated constraints are shown in Table 4.3.

Note that both are nearly identical, but the parameterization SR requires sine as an additional unary operator to account for the occasional periodicity in the thickness and camber lines. With the constraints, exponents are limited to a single term and nested sines are prevented in the equation search.

Max complexity

In PySR, complexity is defined as “the number of nodes in an expression tree” [57]. Generally, a higher complexity leads to a better fit, but there is a danger of overfitting. Overall, the best equations for t_{max} and c_{max} that balance fit and complexity are found with maximum complexity limited to 20. The parameterization equations are more complex in nature, which is they are limited to 50. Due to the nature of the built-in model selection of PySR, the final equation chosen from the Pareto front usually has a complexity of 5 below the maximum.

Equation selection

The output of every PySR run is a Pareto front containing equations and their associated minimized loss across different complexities. Since equations at various complexity levels can be relevant for interpretability, and due to limitations of PySR’s built-in equation selection options, this step is explicitly skipped. For every PySR model, all equations on the Pareto front are considered during model evaluation instead of only the default final equation chosen by PySR.

4.2.4. Equation Evaluation

For the equation evaluation, which pertains to the equation selection among different PySR runs, the equations are tested with data that the PySR instances have never seen before. The test sets and metrics used during the equation evaluation are described in this section.

Test sets

Test sets refer to data structures that are specifically constructed to test the SR equations. They follow the format of the input data per airfoil defined in Tables 4.1 and 4.2, and are used to test for features or thickness and camber predictions. The test data structure for the distribution equations is adjusted to allow additional tests concerning the maximum thickness and camber, as shown in Table 4.4.

Table 4.4: Structure of SR test data per airfoil representation for parameterization equation identification

Latent Variables								Chord	Airfoil Features				Distribution	
z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	x_0 $\vdots$ x_n	t_{max}	$x_{t_{max}}$	c_{max}	$x_{c_{max}}$	x_0 $\vdots$ t_n	x_0 $\vdots$ c_n

Several test sets are generated, such as the UIUC dataset, which consists of 1612 airfoils. The generation of test sets is similar to the generation of the training sets, but rather than sampling the latent vector, the complete vector is decoded to obtain a test set with 1612 latent vectors with their associated airfoil features.

Similarly, a new dataset called AFBench[3] is encoded to obtain the latent vectors which in turn are decoded to obtain the airfoil features. This dataset not only encompasses existing datasets such as

[3]`https://hitcslj.github.io/afbench/`

UIUC and NACA, but also includes 2150 manually-designed supercritical airfoils. These have been further enlarged to $200,000$ airfoils through data synthesis with CST-assisted generation and unconditional generative models such as BézierGAN and diffusion models[80]. Since encoding, decoding and extracting airfoil properties of $200,000$ airfoils is very computationally intensive, 200 random airfoil are sampled from every category in the dataset, leading to a test set of 1600 airfoils.

Other test sets include uniformly sampled latent vectors and vectors obtained by traversing the latent space. These too are limited in size due to computation constraints, but this does not significantly affect the accuracy of the model evaluation.

For parameterization equations, test sets that include thickness and camber values at different chord locations are generated, following the structure of Table 4.4. Additionally, the parameterization equations are verified to match with the maximum thickness and camber values, which are extracted from the decoder output.

Metrics

The main metric used to evaluate the performance of the found equations is the R^2-score. An average is taken over all the test sets to quantify the performance of the equations. The main reason for this choice is the interpretability of R^2, which directly indicates how well the equations capture the variance in airfoil shape. Although MSE is also commonly used, a comparative analysis showed that both metrics yield equivalent rankings for equation selection. This is expected since the same data with the same variance Var(y) are used for both metrics.

$$R^2 = 1 - \frac{\text{MSE}}{\text{Var}(y)} \tag{4.1}$$

Naturally, using solely R^2 would lead to an inherent bias towards more complex equations, since these tend to have a better fit. To account for this bias, a custom score is used that penalizes complexity.

$$\text{Custom Score} = R^2_{\text{average}} - \lambda \cdot C \tag{4.2}$$

Here λ is a tunable weight that determines how strongly complexity C is penalized. This score incorporates a smooth, linear complexity penalty that ensures a gradual trade-off between accuracy and complexity. A threshold-based penalty, where the penalty only applies beyond a certain complexity, has been discarded since it introduces an implicit bias towards the specific complexity threshold. Note that the custom score is used to balance interpretability and accuracy, which is why it is primarily used for feature identification with $\lambda = 0.001$. For parameterization equations, only R^2_{average} is considered.

4.2.5. Parameterization

For the thickness and camber parameterization equations, the best way to assess their performance is by directly using them to parameterize airfoil shapes. This is done by using the thickness and camber definitions defined by NACA [79]. The thickness is measured perpendicular to the camber line, which means that the slope of the camber line has to be calculated to offset the thickness. The following equations can be found in the paper by Ladson et al. [79]:

$$\left(\frac{x}{c}\right) = \left(\frac{x}{c}\right) \pm \left(\frac{y}{c}\right)_t \sin(\delta) \tag{4.3}$$

$$\left(\frac{y}{c}\right) = \left(\frac{y}{c}\right)_{cam} \pm \left(\frac{y}{c}\right)_t \cos(\delta) \tag{4.4}$$

where the angle is defined as the inverse tangent of the camber slope: $\delta = \tan^{-1}\frac{dy_c}{dx_c}$. Slightly rewriting and substituting thickness distribution $(\frac{y}{c})_t = t(x)$ and camber distribution $(\frac{x}{c})_{cam} = c(x)$ yields:

$$x_u = x - \frac{t(x)}{2}\sin(\delta) \tag{4.5}$$

$$y_u = c(x) + \frac{t(x)}{2}\cos(\delta) \tag{4.6}$$

$$x_l = x + \frac{t(x)}{2}\sin(\delta) \tag{4.7}$$

$$y_l = c(x) - \frac{t(x)}{2}\cos(\delta) \tag{4.8}$$

This set of equations is directly used to obtain the x, y coordinates from the parameterization equations.

4.3. Establishing the base VAE

Before SR integration in the training can be investigated, first a proper VAE baseline must be established. The original VAE by Swannet et al. [4, 5] will be used for this, which is the same VAE on which the latent analysis has been performed. However, as will be explained in this section, the definitions of the loss functions within the VAE architecture are changed. Therefore, the original VAE has to be re-tuned to be properly loaded as the base VAE, the method for doing this will be the focus of this section. First, the model architecture is described. Although some hyperparameters of the original VAE are repeated here, the reader is referred to the original papers [4, 5] for a complete rationale. This is followed by the re-tuning process, where different ways to re-tune the original model in the new code are described.

4.3.1. Model architecture

The layer architecture of the base VAE remains unchanged and is shown in Figure 4.4.

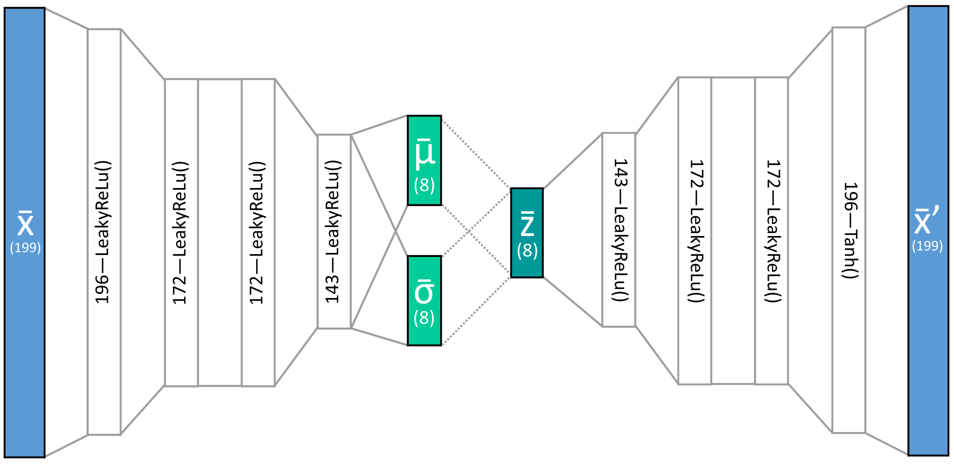


Figure 4.4: base VAE architecture from Swannet et al. [5]

The same batch size of 64 and learning rate of $3 \cdot 10^{-4}$ are used [5]. The loss function for the KL divergence has been changed to more closely resemble the definition of β-VAE from Kingma et al. [6]. MSE is used instead of relative MSE, which is consistent with the loss function used in symbolic regression and has a more direct statistical relationship with the R^2 score used in model selection.

$$\mathsf{L}(\boldsymbol{x}, \boldsymbol{z}) = \mathsf{L}_{\text{recon}} + \beta \cdot D_{\text{KL}} \tag{4.9}$$

$$D_{\text{KL}} = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}[\log q(\boldsymbol{z} \mid \boldsymbol{x}) - \log p(\boldsymbol{z})] \qquad \mathsf{L}_{\text{recon}} = \frac{1}{n} \sum_{i=1}^{n} \left|\boldsymbol{x}_i - \boldsymbol{x}_i'\right|^2 \tag{4.10}$$

Note that, contrary to the ELBO defined in Equation (2.18), the loss $\mathsf{L}(\boldsymbol{x}, \boldsymbol{z})$ in Equation (4.9) is minimized rather than maximized. Therefore, the KL divergence is added instead of subtracted. Due to the changed definition of the loss function, it is necessary to re-tune β.

4.3.2. Re-tuning $\boldsymbol{\beta}$

Three training approaches are attempted to re-tune β and reconstruct the original VAE. First, the VAE is trained and re-tuned following the exact same method from from Swannet et al. [4], using a warm started β-sweep. Second, a β-sweep is conducted where after every β-value training the weights and biases are re-initialized such that every training starts from scratch. Third, the weights and biases of the original VAE are loaded into the new architecture and used as a starting point for training and re-tuning. In all cases, the re-tuning process involves training the VAE with the same hyperparameters across different β values and plotting the loss terms to select a β that balances KL divergence and reconstruction [4].

4.4. SR implementation in VAE training

The next step in using SR to facilitate an interpretable latent space is to implement it in the training of the base VAE. The tuning insights from the latent analysis can directly be used to do this more efficiently. First, the new model architecture of the SR-VAE is presented. This is then followed by the method for the novel SR Loss calculation, which is based on airfoil reconstructions from analytical equations. Subsequently, some minor adjustments to the PySR model are described to enable multiple SR iterations throughout the training process, and different training methods are described for SR integration. Finally, the new model selection to assess the various SR-VAE models is outlined, balancing VAE performance and accurate SR equations.

With the established base VAE, there is a clear starting point for the integration of symbolic regression into VAE training. The goal is to incorporate SR into VAE training such that the overall model is easier to interpret through SR. This should lead to more accurate SR equations that can map latent space to physical space.

4.4.1. Model architecture

To integrate SR into the VAE, a new SR-VAE architecture is defined as shown in figure 4.5:

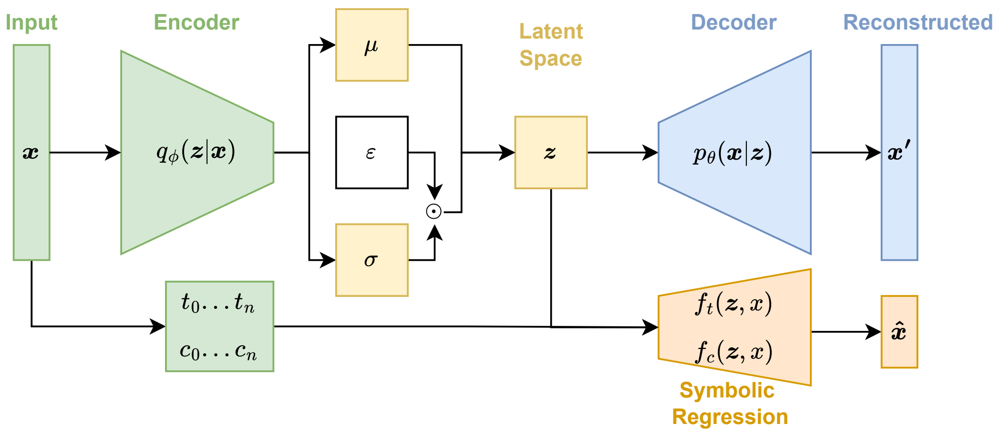


Figure 4.5: Schematic overview of the novel SR-VAE

$$\mathsf{L}(\boldsymbol{x}, \boldsymbol{z}) = \mathsf{L}_{\text{recon}} + \beta \cdot D_{\text{KL}} + \gamma \cdot \mathsf{L}_{\text{SR}} \tag{4.11}$$

The SR-VAE is very similar to the VAE defined in Figure 2.5. However, as also shown in Equation (4.11), the reconstruction losses of both the decoder and SR equations are minimized simultaneously. It is important to note here that the SR itself is not optimized during training, the optimizer only changes the model parameters of the VAE. This means that the encoder and decoder will be trained such that both decoder and SR loss is minimized, which facilitates a latent space where SR can construct more accurate equations. The SR settings are constant throughout the SR-VAE training.

This SR implementation is different from the latent analysis presented in Figure 4.2. For SR-VAE, multiple SR searches are performed during VAE training, and its losses directly affect VAE training. In contrast, during the latent analysis, the VAE model is only used to sample the input and output of the decoder for a single SR search, which does not involve VAE training at all. Although both the latent analysis and SR-VAE extract latent vectors from the latent space, the symbolic regression equations in SR-VAE are trained on the original airfoils passed to the encoder, rather than on the reconstructed airfoils output by the decoder. The SR-VAE uses SR with the input data to obtain equations with the same function as the decoder, while the latent analysis interprets the latent space by approximating the decoder.

4.4.2. SR loss calculation

As can be seen from Figure 4.5, SR is used for parameterization rather than airfoil features. The main reason for this is that other geometric or aerodynamic features can always be extracted from the airfoil shape. Furthermore, the parametric shape can directly be used to calculate the SR loss:

$$\mathsf{L}_{\mathsf{SR}} = \frac{1}{n}\sum_{i=1}^{n} |\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i|^2 \tag{4.12}$$

The SR loss is defined as the MSE between input vector $\boldsymbol{x}$ and SR parameterized vector $\hat{\boldsymbol{x}}$. Due to the mathematical similarities between $\mathsf{L}_{\mathsf{recon}}$ and L_{SR}, the weighting parameter γ is set to 1.

For the calculation of L_{SR}, the parameterized vector $\hat{\boldsymbol{x}}$ containing the parametric airfoil reconstructions must be obtained. This requires SR to be run within the training procedure to obtain equations for thickness $f_t(\boldsymbol{z})$ and camber $f_c(\boldsymbol{z})$. For this, input data must be generated in the same structure as previously presented in Table 4.2.

First, the input vector $\boldsymbol{x}$ is converted to airfoil coordinates to extract thickness t_n and camber c_n at n random chordwise locations x_n. The input vector is also encoded to obtain the corresponding latent vector $\boldsymbol{z}$. For every airfoil within the vector, the latent representation is duplicated and the random chord locations are concatenated to allow for the definition of multiple thicknesses and cambers along the chord. The input data can then be fed as input to the same SR class from the latent analysis. Once again, the parametric equation with the highest R^2 is chosen from the thickness and camber SR results, resulting in the parametric equations $f_t(\boldsymbol{z})$ and $f_c(\boldsymbol{z})$.

These parametric equations are used to construct the parameterized vector $\hat{\boldsymbol{x}}$. For this, once again every latent airfoil representation in $\boldsymbol{z}$ is formatted to match the training data of the SR equations, but this time all cosine spaced chord locations x_n with $n = 100$ are used. This is directly fed to the parameterization equations, from which the parametrized airfoil shapes can be extracted following the method described in subsection 4.2.5. The y-coordinates of all parameterized airfoils, which resemble $\hat{\boldsymbol{x}}$, can then be used in Equation (4.12) to calculate the SR loss.

4.4.3. PySR setting adjustments

Incorporating SR within the VAE leads to multiple PySR runs throughout the training process. With tuned PySR model parameters from the latent analysis defined in Table 4.3, a SR run from scratch for both thickness and camber distribution takes about 9 minutes on a high-performance workstation. Considering that there are 23 batches per epoch, keeping the same number of iterations at 500 would be infeasible. However, too few iterations would lead to an unstable SR loss, which is detrimental to the VAE training process.

To solve this issue, the warm start functionality from PySR is used. This ensures that the symbolic regression search automatically continues where the last call finished, instead of starting from scratch and overwriting previous results. This allows for a very low number of iterations per PySR run but a net high number over the complete SR-VAE training procedure. The number of iterations is arbitrarily set to 10. To further speed up the PySR run, only 5 random chord locations are sampled for the thickness and camber distribution, where previously 10 were used. An individual PySR run during training now takes about 5 seconds instead of the 9 minutes during latent analysis. These adjustments facilitate the integration of SR into training with minimal runtime overhead while maintaining consistent symbolic regression performance.

4.4.4. SR integration strategy

The ideal way to incorporate SR into VAE training is to run PySR every time the VAE loss is evaluated, which means that there is a symbolic regression search for every batch. This ensures that the interpretability through SR — or the ability of SR to obtain accurate equations to interpret the latent space — is properly evaluated at each optimization step. With the adjustments described in the previous section, this is feasible. However, considering that there are 23 batches, 1000 epochs would take over 24 hours.

An alternative way would be to run PySR once per epoch, subsequently using the best equation from this symbolic regression search to calculate L_{SR} for all batches. Since there are fewer stochastic symbolic regression runs, the VAE training should be more stable. Although this does result in a significant speed-up, the focus is shifted from interpretability through SR to fitting the latent space to SR equations.

Going even further, the best equations for thickness and camber can be chosen and fixed throughout the entire training process. This ensures that the SR loss is always calculated with the same equations, making the loss more consistent throughout training. However, this completely changes the focus from ensuring interpretability through SR to fitting the latent space to adhere to specified equations. Since it requires a set of SR equations, this training procedure is only used after an SR-VAE using either of the former two strategies is trained. In this way, the first SR-VAE training ensures that an accurate set of SR equations are obtained, while the latter fixes the SR equations to shift the focus to optimizing the VAE. This ensures that a more stable VAE training can be conducted in which VAE losses can potentially be improved without compromising SR loss.

4.4.5. Model evaluation

More than 100 different SR-VAE models have been trained, with variations in SR training integration strategy, weighting parameters β and γ, and initial conditions. Contrary to the latent analysis, both the performance of the VAE and the SR equations have to be assessed to find the optimal SR-VAE. To objectively assess both VAE and SR performance, all models are evaluated using the complete UIUC and AFBench datasets.

As evaluating SR equations is computationally intensive, first only the VAE reconstruction losses are evaluated. This involves encoding and decoding all airfoils in the UIUC and AFBench datasets, after which the MSE between the original and reconstructed airfoils is computed. For both datasets, the overall average MSE and the mean of the top 1% worst reconstruction losses are calculated. This results in 4 loss metrics for the decoder, for each dataset one represents the average over the overall dataset and another the average of the outliers.

The top 20 models based on VAE performance are then further evaluated for SR performance. During the training of every SR-VAE model, symbolic regression searches repeatedly produce camber and thickness equations. For equation selection, the same evaluation method and test sets from subsection 4.2.4 are used, but with UIUC and AFBench test sets that extract features from the actual datasets rather than from the reconstructed output of the decoder. This ensures that the UIUC and AFBench test sets test the equations ability to reconstruct existing airfoils, while the sampled test sets from the decoder test if the equations are continuous and generalizable. Since the latent space changes between models, new test sets are generated for each SR-VAE. The equations that exhibit the highest R^2 or lowest MSE across these test sets are then selected.

For each model, the selected SR equations are used to reconstruct every airfoil in both datasets. The average parametric MSE and the top 1% worst-case losses are computed, similar to the VAE evaluation. Airfoils with the highest parametric loss are also analyzed to assess robustness and generalizability, since the ultimate goal is universal parameterization. This once again leads to 4 loss metrics.

To objectively compare both VAE and SR performance across the 20 SR-VAE models, the 8 averaged reconstruction losses — for both UIUC and AFBench datasets, both overall and top 1% worst cases and both VAE and SR performance — are normalized using the standard score (also known as z-score):

$$s = \frac{x - \mu}{\sigma} \tag{4.13}$$

This removes any bias caused by differences in scale between metrics. The normalized losses are then averaged to yield a single ranking metric per model. The resulting model ranking is shown in Appendix C, and the best SR-VAE model is further analyzed in Chapter 5.

5. Results and Discussion

This chapter contains the main results and discussions of the research process. The structure of this chapter follows the same order as indicated in the research plan and methodology. First, the results from the latent analysis of the original β-VAE will be discussed in Section 5.1. The original VAE will be established as a baseline in Section 5.2, and a baseline for SR performance is established in section 5.3. The results of SR integration within the VAE will be outlined in Section 5.4. Finally, the results of using the VAEs and equations in an optimization study will be discussed in Section 5.5.

5.1. Latent Analysis of β-VAE

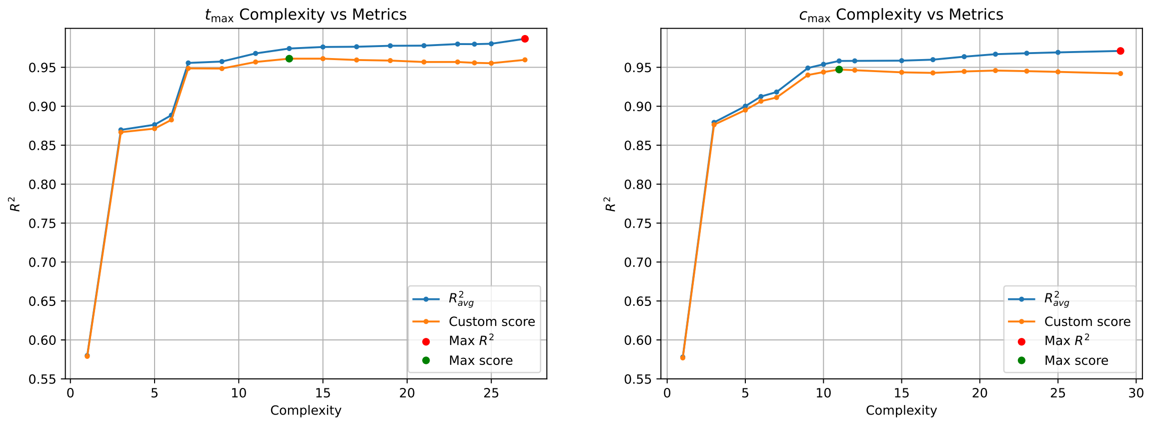
This section outlines the main results from the latent analysis of the original β-VAE from Swannet et al. [4] through SR. The equations from the algorithm allow for direct mapping from the latent space to physical space. The results are split into feature equations and parametric equations.

5.1.1. Feature equations

Throughout the tuning and equation identification process, 177 runs for t_{max} and 124 for c_{max} have been conducted with PySR. For each complexity, across all runs, the equation with the best fit has been identified. If certain equations had a worse MSE than those with lower complexity, they were filtered out. More information about all the runs conducted in PySR, including the selection of filtered equations, can be found in Appendix B. The Pearson correlation matrix of the original VAE for is used as a reference and is shown in Figure A.1. Especially Z_3 and Z_4 have a strong correlation with max thickness and camber, while Z_7 and Z_1 have the highest correlations for the associated chord locations.

Metrics

As explained in subsection 4.2.4, all equations have been tested with test sets, from which R^2_{avg} and Custom score have been identified. The performance with increasing complexity is shown in Figure 5.1.



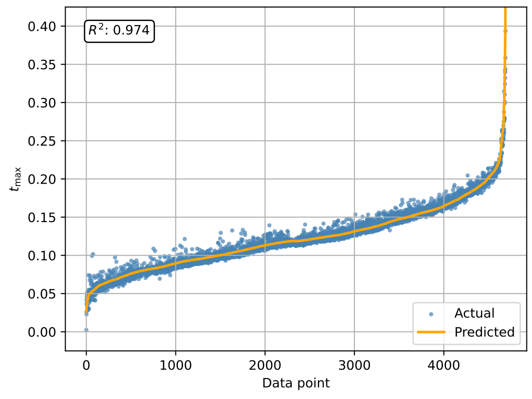
(a) t_{max} **(b)** c_{max}

Figure 5.1: R^2 and custom score of the most accurate t_{max} and c_{max} equations, with increasing complexity

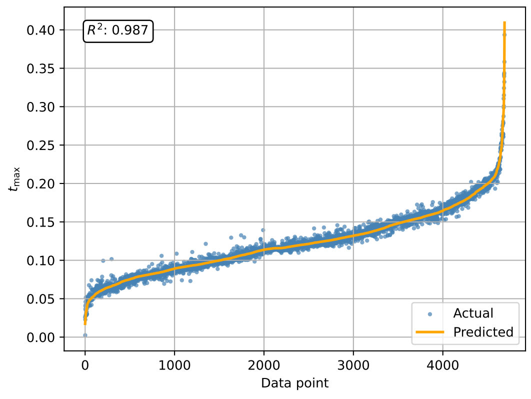
The equations with the highest score, denoted with a green marker in the graphs, represent the most accurate equations that are still adequately interpretable. As expected, the equations with the best fit to the test data have the highest complexity, denoted with a red marker. In general, the t_{max} equations perform better in both R^2 and Custom score compared to the c_{max} equations. The reason for this could be that estimating max camber and its location is inherently more complex than the thickness counterparts. Another reason could be that slightly fewer PySR runs have been conducted for c_{max}.

Analysis

The equations with the highest custom score and R^2 are presented in Equations (5.1) and (5.2) for t_{max} and Equations (5.3) and (5.4) for c_{max}, accompanied by the test data fit in Figures 5.2 and 5.3.



(a) t_{max} highest custom score



(b) t_{max} highest R^2

Figure 5.2: Predicted values of t_{max} equations with highest custom score and R^2 plotted against the actual values of all test sets, sorted by ascending predicted values

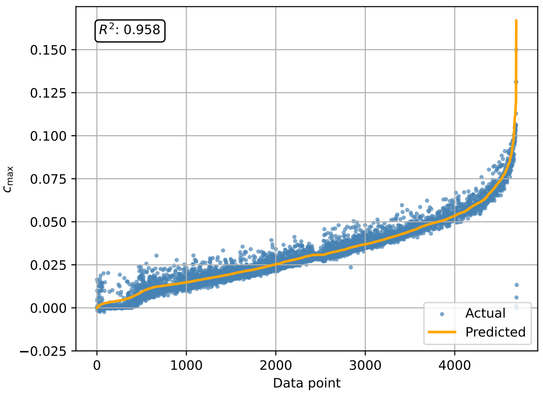
$$f_{t_{max},cr} = 0.00407 \cdot Z_7 + 0.0684 \cdot (0.223 \cdot 1.34^{Z_1} + 0.223 \cdot Z_3 + 1.0)^{1.88} + 1.0)^{1.61} \tag{5.1}$$

$$\begin{aligned} f_{t_{max},r2} = &\ 0.00307 \cdot Z_1 \cdot (Z_1 + 3.45) + 0.00307 \cdot Z_3 \\ &\cdot (-0.682 \cdot Z_0 + Z_3 - 0.682 \cdot Z_7 + 11.4) \\ &+ 0.00307 \cdot Z_7 \cdot (Z_7 + 1.51) + 0.0969 \end{aligned} \tag{5.2}$$

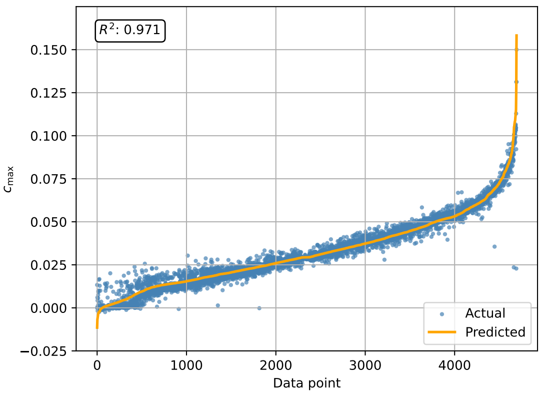
As can be seen from the equations, both contain the latent variables Z_1, Z_3 and Z_7. Considering Figure A.1, the presence of the former two variables is as expected but the latter is somewhat remarkable. While Z_7 does not show a dominant correlation for t_{max}, it does show one for $x_{t_{max}}$, which is likely the reason for its inclusion in the t_{max} equations. In the more complex SR equation from (5.2), Z_0 is included as well. There is no clear reason for this considering Figure A.1. While it does show a moderately high correlation for $x_{t_{max}}$, the inclusion of Z_4 would have been more likely for this reason.

Considering Figure 5.2, the equation with the highest custom score (Equation (5.1), shown in Figure 5.2a) achieves a test data fit comparable to the more complex Equation (5.2), whose fit is shown in Figure 5.2b. Despite the higher complexity and R^2 score of Equation (5.2), there is no visible improvement in predictive accuracy. Equation (5.1) therefore demonstrates strong performance without sacrificing interpretability.

Mathematically considering Equation (5.1), it seems that Z_1 is the most dominant variable in the equation due to its exponential growth, followed by Z_3. In comparison, the Z_7 term is insignificant unless both Z_1 and Z_3 are small. This is in contrast with the correlation matrix shown in Figure A.1, where the latent variable with the highest correlation of 0.94 was identified to be Z_3, and Z_1 as the second dominant latent variable with a correlation of 0.28 [4]. Negative t_{max} are only possible if $Z_7 < 0$.



(a) c_{max} highest custom score



(b) c_{max} highest R^2

Figure 5.3: Predicted values of c_{max} equations with highest custom score and R^2 plotted against the actual values of all test sets, sorted by ascending predicted values

$$f_{c_{max},cr} = 0.0181 \cdot (0.111 \cdot Z_3 + 0.623 \cdot Z_4 + 1.0)^{1.61} \tag{5.3}$$

$$\begin{aligned} f_{c_{max},r2} = &\ 0.0175 - 0.00179 \cdot Z_0 + (0.00237 \cdot Z_4 + 0.0181) \\ &\cdot (0.0686 \cdot Z_1 \cdot (Z_1 - 1.06 \cdot Z_7) + 0.0686 \cdot Z_3 \cdot (Z_3 + 2.05) + Z_4) \end{aligned} \tag{5.4}$$

Considering Equations (5.3) and (5.4), both contain latent variables Z_3 and Z_4, while the more complex Equation (5.4) also incorporates effects from Z_1 and Z_7. Once again, the incorporation of Z_3 and Z_4 agrees well with the highest correlations for c_{max} shown in Figure A.1. Similarly to the t_{max} equations, it can be deduced that the inclusion of Z_1 and Z_7 in Equation (5.4) is attributed to their high correlation with $x_{c_{max}}$. However, Z_0 has a low correlation for both c_{max} and $x_{c_{max}}$. In the correlation matrix, it is shown to have a moderate effect on leading edge radius R_{LE} and trailing edge wedge angle γ_{TE}, but this should not have a significant effect on the maximum thickness or camber.

This time, there is a visible smaller spread of test data points around the predicted values in Figure 5.3b compared to that of Figure 5.3a, which indicates a better fit. Nevertheless, it is remarkable how well Equation (5.3) follows the test data, while incorporating significantly fewer latent variables than (5.4).

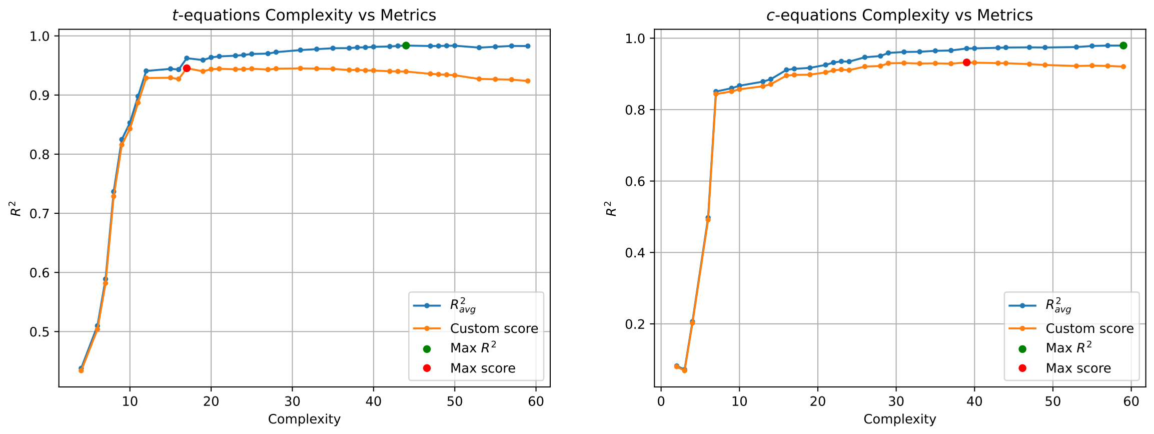
From a mathematical perspective, Z_3 and Z_4 exhibit similar behavior in Equation (5.3). Both have the same exponent, but when the variables tend to infinity, Z_4 will dominate due to a significantly higher coefficient of 0.623 compared to 0.111. If both equations are small, the sum in the parentheses will tend to 1. This agrees well with Figure A.1, where Z_4 is the most dominant latent variable for c_{max}, followed by Z_3. Further examining the equation reveals that negative maximum cambers are not possible, since a negative value within the parentheses would lead to complex numbers.

5.1.2. Parameterization equations

Since the PySR hyperparameters were already tuned to find equations for t_{max} and c_{max}, significantly fewer runs were required to get to appropriate results. In total, 60 runs for the thickness distribution and 57 for the camber distribution have been conducted with PySR. Once again, the equation with the best fit has been identified for each complexity across all runs. Equations where an incremental increase in complexity led to an increase in MSE, have again been filtered out. The final set of equations per complexity, as well as more information on all the performed PySR runs can be found in Appendix B.

Metrics

The R^2 and custom score of all the best equations per complexity can be found in Figure 5.4.



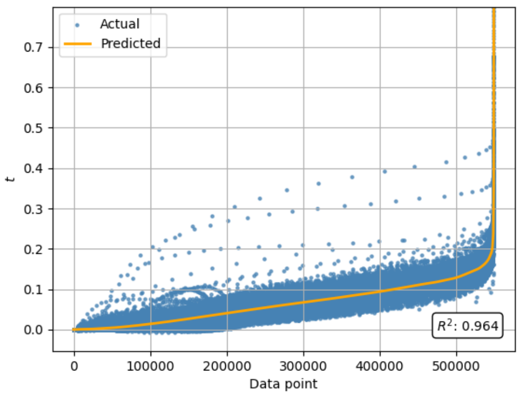
(a) Thickness distribution **(b)** Camber distribution

Figure 5.4: R^2 and custom score of the most accurate thickness and camber distribution equations, with increasing complexity

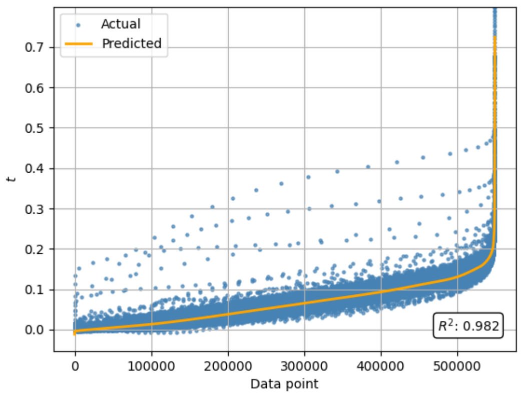
Contrary to before, Figure 5.4a shows that the equation with the highest R^2, denoted in green, is not the equation with the highest complexity. There is also a significant difference in complexity between the equations with max custom score, denoted in red, and max R^2, denoted in green. As can be seen in Figure 5.4b, the camber distribution equations behave a bit more predictably. The equation with the highest R^2 is the equation with the highest complexity, and the complexity of the equation of max custom score is much closer. However, contrary to the feature equations, the main driver of the parameterization equations is accuracy rather than complexity. Especially since the equations should use as many latent variables as possible to reflect the original decoder of Swannet et al. [4].

Analysis

Equations with the highest custom score and the highest R^2 are presented in Equation (5.5) and (5.6) for the thickness distribution and in Equation (5.7) and (5.8) for the camber distribution, accompanied with graphs of the test data fit in Figures 5.5 and 5.6. Note that there are significantly more test data points, since for every airfoil multiple thickness and camber values are defined along the chord.



(a) t-equation highest custom score



(b) t-equation highest R^2

Figure 5.5: Predicted values of t-distribution equations with highest custom score and R^2 plotted against the actual values of all test sets, sorted by ascending predicted values

$$f_{t,cr} = X^{0.505} \cdot (0.0282 - 0.0281 \cdot X) \cdot (Z_1 + 9.14 \cdot (0.229 \cdot Z_3 + 1.0)^{1.5}) \tag{5.5}$$

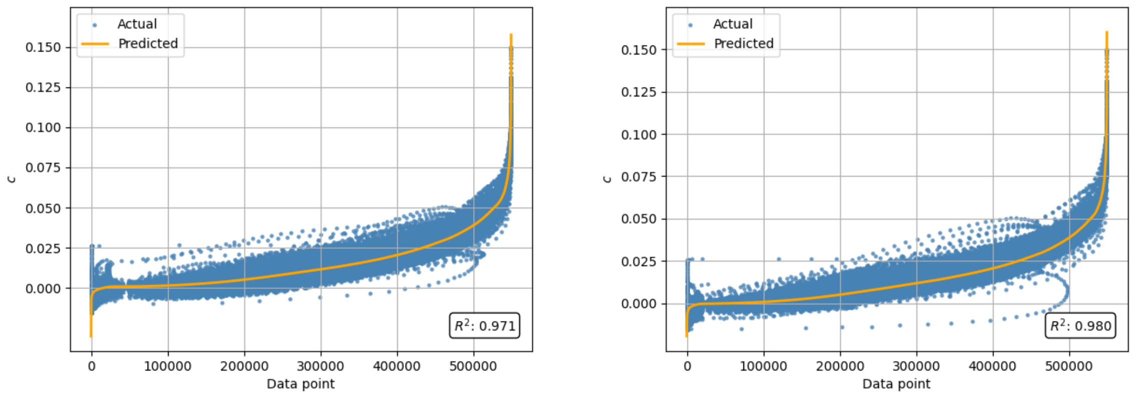
$$f_{t,r2} = 0.00713 + 0.04513(1.45871X^{0.44769}Z_3 + X + 0.31695Z_1 - 0.43911(X - 0.37690)(XZ_1 + Z_4 + 2Z_7 + \sin(Z_0)) + 1.87001) \cdot \sin(3.15397X^{0.51341}) - 0.00841 \cdot \sin(Z_1 + 0.85852) \tag{5.6}$$

Although it has a moderately high R^2, the equation with the highest custom score $f_{t,cr}$ shown in Equation (5.5) only contains latent variables Z_3 and Z_1. Although that is fine for estimating t_{max}, to capture the entire thickness distribution, all active latent variables should be present in the equation, especially those who exhibit a high correlation to $x_{t_{max}}$. Swannet et al. identified 6 active latent variables out of the 8, from which Z_5 and Z_6 are inactive [4]. This is also clearly depicted in the Pearson correlation matrix of Figure A.1. The equation with the best fit containing as many latent variables as possible is shown in Equation (5.6), and includes Z_0, Z_1, Z_3, Z_4 and Z_7. Unfortunately, the equation does not contain Z_2. As can be seen in Appendix B, none of the PySR results incorporated Z_2 into their equations.

A different equation with a complexity of 44 instead of 50 showed a marginally higher R^2 without using Z_0. The equation and plot is shown in Figure B.2. From this, it could be argued that Z_0 is not as relevant for the thickness parameterization. However, since the plots are nearly identical with similar R^2, Equation 5.6 is chosen instead as it is a function of more active latent variables.

By taking a closer look at Figure 5.5, it can clearly be seen that Figure 5.5b with highest R^2 shows a much smaller spread of test data points than Figure 5.5a for max custom score. The higher accuracy is very clear, which is likely attributed to the inclusion of more latent variables. Even if accuracy is not the main driver behind equation selection, the best choice appears to be Equation (5.6), since this is the most accurate equation that also incorporates the most latent variables. Finally, there are noticeable outliers in both graphs, which neither equation captures well.

A quick mathematical analysis of Equation (5.6), reveals that X is the most dominant, as it has linear and multiplication interactions that lead to quadratic effects. This is to be expected, since thickness varies for each airfoil with X while $\boldsymbol{Z}$ is constant. Other than that, Z_1, Z_3 and Z_4 seem most dominant, since these are multiplied or directly scaled by X. Z_0 is only included in a trigonometric term and Z_7 also appears once, making them the least dominant. The equation can become negative depending on the sine term or the factor within parentheses, while realistically this should not be possible. However, since this equation exhibits the highest R^2, it means that it can best predict the thicknesses of all airfoils in the diverse test sets. This means that negative thicknesses are likely very rare, and only occur with latent variables far from the mean.



(a) c-equation highest custom score

(b) c-equation highest R^2

Figure 5.6: Predicted values of c-distribution equations with highest custom score and R^2 plotted against the actual values of all test sets, sorted by ascending predicted values

$$\begin{aligned} f_{c,cr} &= 0.0155 \cdot X \cdot (0.8120 - 0.8123 \cdot X) \\ &\cdot ((3.4965 - 5.5194 \cdot X) \cdot (X \cdot Z_7 + Z_3 + Z_4) \\ &+ (Z_4 + 6.6944) \cdot (Z_1 \cdot (X - 0.4492) \\ &+ 0.7029 \cdot Z_4 + 0.7776)) + 0.0009 \end{aligned} \tag{5.7}$$

$$\begin{aligned} f_{c,r2} = {} & 0.01553\, X(1.16254 - 1.15956\, X) \cdot \big(- 0.90316\, X Z_7 \\ & + (2.66484 - 3.80325\, X)(X(-0.88327\, Z_0 + Z_7 - 1.85512) \\ & + Z_3(X + 0.51648) + Z_4 + 0.82437) + (Z_4 + 4.83158) \\ & (Z_1(X - 0.44830) + 0.50278\, Z_4 + 1.09133) - 1.57776\big) - 0.00017 \end{aligned} \tag{5.8}$$

The differences in test set fit between the camber equations depicted in Figure 5.6 are less extreme than those of thickness. This is expected since the R^2 of both are closer. As seen in Equations (5.7) and (5.8), both are a function of latent variables Z_1, Z_3, Z_4 and Z_7. The equation with the highest R^2 depicted in 5.6b, incorporates Z_0 as well. Like with the thickness distribution equations, technically all active latent variables should be present. Once again, there is no occurrence of Z_2 in any of the PySR results. Remarkably, exponents are absent in both equations, while they are included as operators.

From the graphs in Figure 5.6, it can be seen that the spread of test data points is quite similar around the predicted values. However, the test data fit for Equation 5.7 in Figure 5.6a appears to have less extreme outliers than that of Equation 5.6 shown in Figure 5.6b. This indicates that simpler equations might be less sensitive to outlier airfoils. Despite this, in general the points are not as concentrated near the predicted values as (5.8). The most likely reason for this is the lack of Z_0 in its equation.

From a mathematical point of view, X, Z_3, Z_4, and Z_7 all seem quite dominant due to the factorization of two large multiplicative factors. Z_1 only appears in linear terms involving X, so its impact is less dominant and only scales with other parameters. From the equation, it seems that Z_0 is the least dominant, since it does not occur in as many multiplications as the previous parameters. But again, the equation is quite hard to interpret. However, this is not as important because parametric equations are generally hard to interpret. The most important aspect would be the parameterization accuracy.

Parameterization

The parameterized shapes are obtained from the SR distribution equations following the method described in subsection 4.2.5. To investigate the parameterization potential of SR, first the latent effects of the decoder and the parameterized equations are compared. For this, the effect each latent parameter on the generated airfoil is assessed by plotting the extremes for each latent factor, as shown in Figure 5.7. As can be seen in the figure, the behavior of the SR parametrization from varying latent variables by two standard deviations very closely resembles that of the decoded airfoils.

As anticipated, the inactive latent variables Z_5 and Z_6 remain inactive. For the SR parameterization, Z_2 is inactive as well, although the latent effect for the decoded airfoil is not too prevalent either. Especially the associated plots of Z_5 and Z_6 show a very close resemblance, which indicates that the SR parameterization shows good reconstruction accuracy when latent variables are kept at their mean.

In addition to Z_2, the active latent parameter with the smallest perceptible effect appears to be Z_0. Z_0 pertains to the leading edge radius and the upper surface curvature. Both of these geometric features are not captured very well by the SR parameterization. While generally the lower surface shows a close resemblance to the decoded airfoil, the upper surface can be noticeably different. Poor performance with respect to Z_0 is expected, since for both the thickness and camber equations Z_0 was identified to be one of the least dominant latent parameters.

The effect of Z_1 is clear for both the decoded and parametric airfoil. However, similar to Z_0, there is a noticeable difference near the leading edge, but this time at the lower surface rather than the upper surface. Nevertheless, the overall latent behavior of the decoded and parametric airfoil is very similar, indicating that the SR parameterization accurately captured the geometric features influenced by this latent variable.

The same can be said for the most dominant latent variables, Z_3, Z_4 and Z_7. While there are some subtle differences, the SR parameterization accurately captures the change in maximum thickness, camber, and chordwise position of maximum thickness, respectively. It is important to note that the parameterization is compared to the decoder at its extremes and that the performance improves substantially closer to the mean μ_i of the latents. In general, SR parameterization shows a lot of promise in capturing intricacies of the multilayer decoder network in only two interpretable equations.

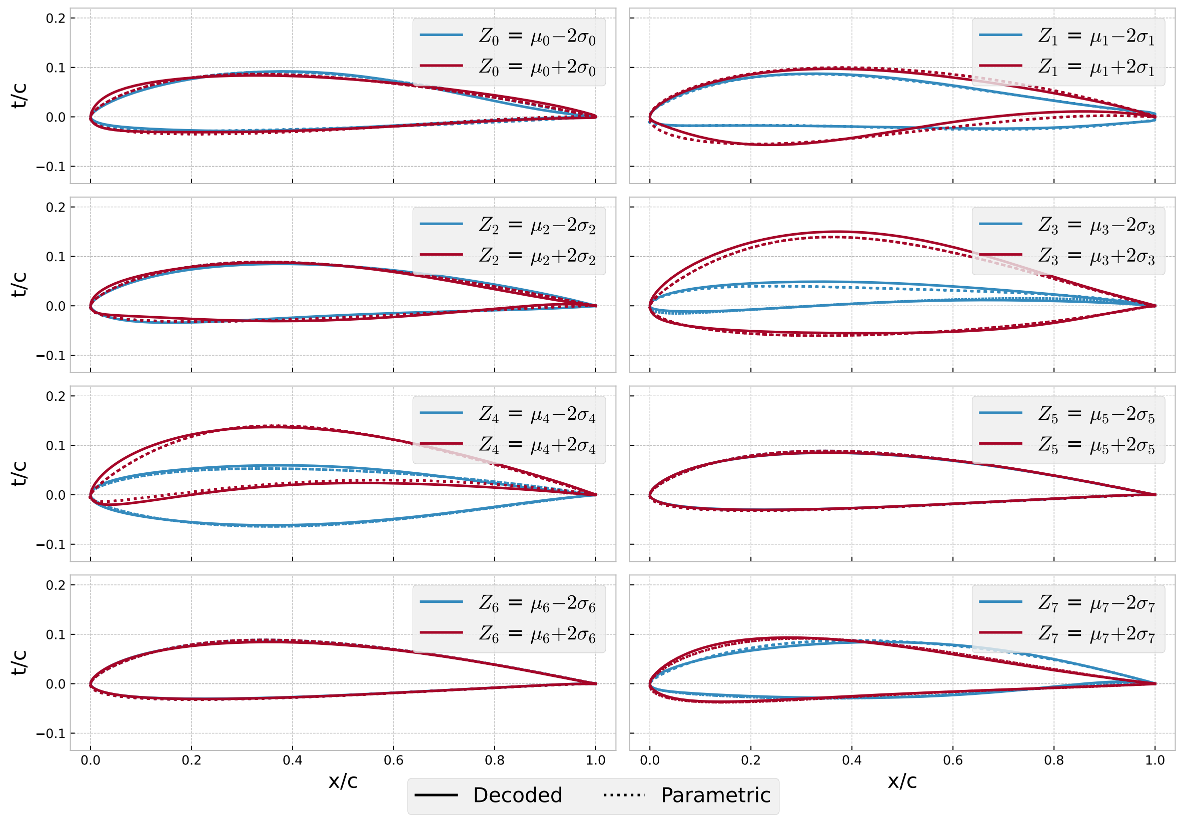


Figure 5.7: Effect of latent parameters on the generated shape in a range of $Z_i = \mu_i \pm 2\sigma_i$, while the rest are at their mean, for both decoded and parametric airfoils as a result of the latent analysis

Validation

To further assess the parameterization potential of the SR equations, validation plots are generated that include the airfoils where the parameterization equations showed the highest loss. These are shown in Figure 5.8, along with the decoder losses of the original VAE. Note that, contrary to similar plots from Swannet et al. [4], MSE is used instead of RMSE for consistency.

For some of the more uncommon shapes, the parameterization deviates significantly from the original airfoil. Especially the worst airfoils in the top row seem to have failed. Parametric equations have difficulty fitting airfoils with an unconventionally high thickness and an open trailing edge. Furthermore, the parametric airfoil always forces a sharp trailing edge, even when this is not the case. Finally, similar to what was observed with the latent parameter effects, the reconstruction inaccuracies for the better fitted airfoils seem to be located near the upper surface leading edge. This is likely attributed to Z_0 and Z_2 being insufficiently captured by the equations. For some airfoils however, like the HAM-STD-HS1-620, the fit is a lot closer, with the exception of the trailing edge.

Compared to the decoder, the parameterization equations seem to perform significantly worse. Likely, the failed airfoil reconstructions pertain to the outliers defined in Figure 5.5b. The main reason for the degraded performance is that in the end, the SR parameterization is only an approximation of the decoder, which in itself is also an approximation of the actual airfoils. The decoder itself also shows some visible discrepancies, particularly for the AH 93-W-480B where the shape is not smooth.

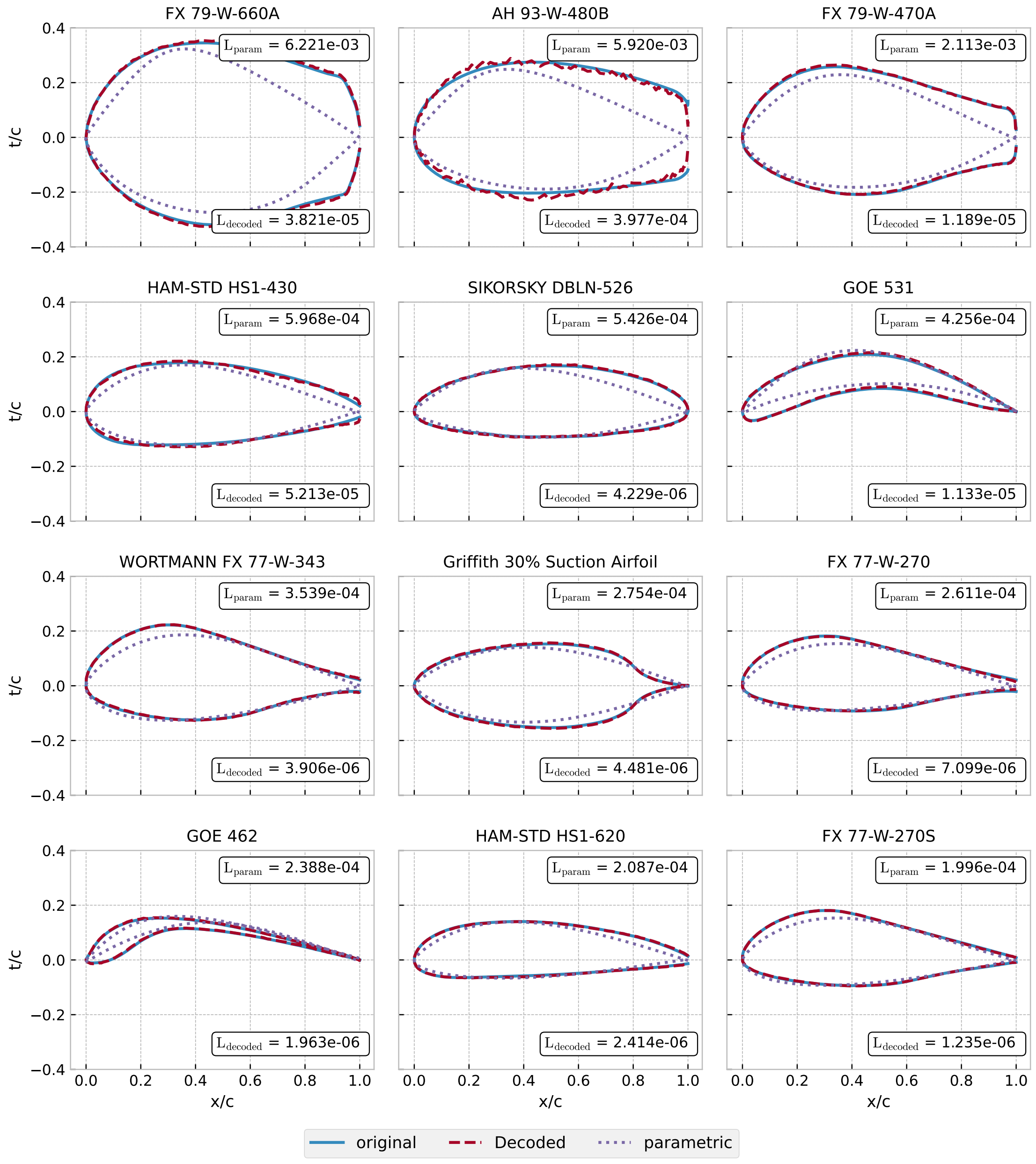


Figure 5.8: Validation plots showing parametric and decoded reconstruction of 12 airfoils with highest parametric loss as a result of the latent analysis, the loss marked at corner of each plot is the MSE

A significant advantage of the SR parameterization over the decoder is its robustness. No Bézier/B-spline smoothing is necessary for it to generate smooth airfoil shapes. Even with the use of smoothing for the decoder airfoil, the SR parameterization shows significantly higher stability in the tails of the latent distribution compared to the decoder. An example of this is shown in Figure 5.9, where Z_0, Z_1 and Z_2 and Z_7 are pushed to the tail end of the distribution. The parametric airfoil shows a much smoother reconstruction than the decoded airfoil. However, once again it should be noted that the fit of the parametric airfoil is not optimal, since it remains an approximation of the decoder.

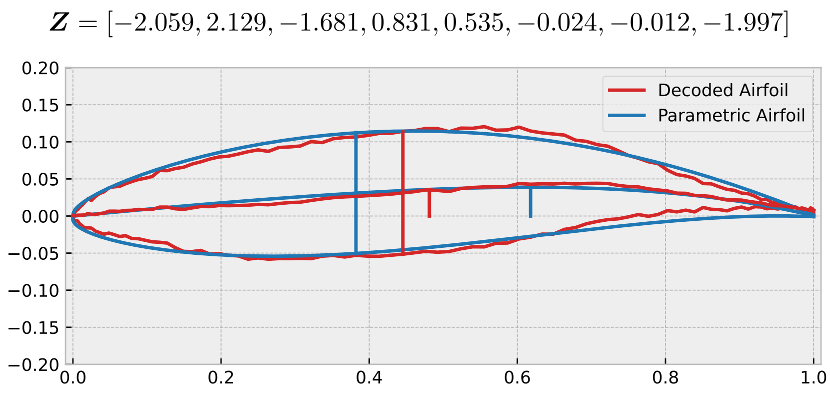
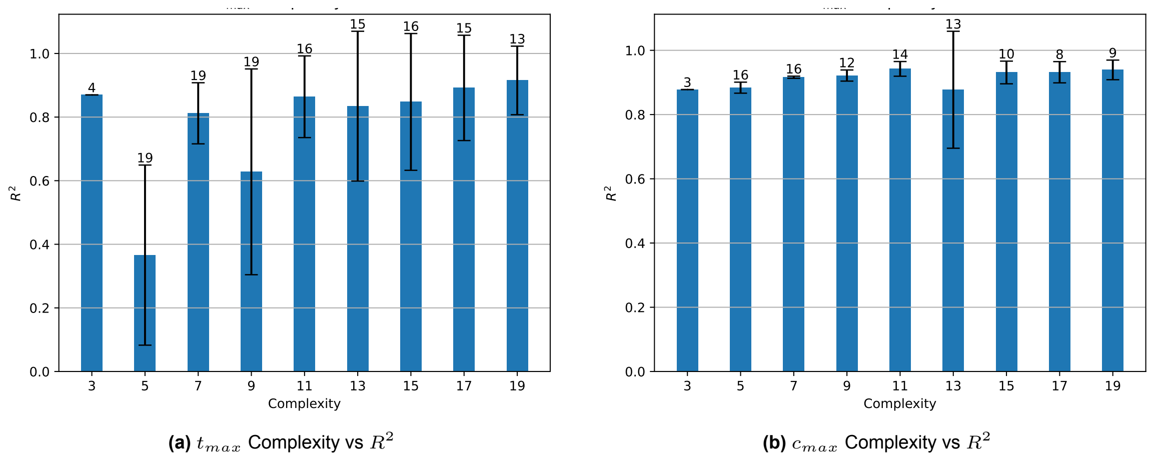


Figure 5.9: Example airfoil to show the increased robustness of the parametric airfoil over the decoded airfoil

5.1.3. Uncertainty

As mentioned, a disadvantage of GP-based methods is that it is stochastic, which means that it does not always converge to the same result. This subsection analyzes this for t_{max} and c_{max}, since these have the most runs with the tuned parameters. The uncertainty is depicted in Figure 5.10.



(a) t_{max} Complexity vs R^2 **(b)** c_{max} Complexity vs R^2

Figure 5.10: Bar chart showing R^2 and complexity of feature equations from the tuned model parameters, including uncertainty

Here, the number on top of the error bars indicates the number of equations at that specific complexity. With these tuned parameters, 21 runs have been conducted in total for t_{max} and 18 for c_{max}. Especially for t_{max} in Figure 5.10a, most complexities have quite a bit of variance. This shows that PySR does not always converge to the same equations, even though the settings are exactly the same. However, the uncertainty seems to be much smaller for c_{max} in Figure 5.10b. Apparently, PySR is more efficient in extracting the underlying relations for c_{max} compared to t_{max}. The exact reason for this is as of yet not clear, although there are notably fewer equations per complexity for max camber compared to thickness. Conducting more runs with tuned parameters should give a better insight into this.

For the distribution equations, the uncertainty is a bit harder to quantify since significantly fewer runs have been conducted. However, as can be seen from Appendix B, the specific run that yielded the most optimal equation with a complexity of 44 pertained to the tuned hyperparameters, with a max size of 45 rather than 50. Further runs with a higher max size did not yield equations with better performance. This once again shows the inherent uncertainty of PySR.

5.2. Establishing the Base VAE

To properly analyze the effect of SR on VAE performance, the original VAE of Swannet et al. [4] is established as a baseline with the new loss function definitions. This is the same VAE that was used for the latent analysis, ensuring a fair comparison of the results. As described in Section 4.3, several approaches have been investigated. This section includes results from re-tuning through the same method as the original VAE, important observations related to warm-started β-sweep, the effect of tuning with warm start, and the effect of repeated warm starts with identical hyperparameters. It is imperative to properly quantify the effect of establishing the Base VAE, and the effect that re-training has in general. Only then can the effects of integrating SR into VAE training be properly investigated.

5.2.1. Re-tuning with method from Swannet et al.

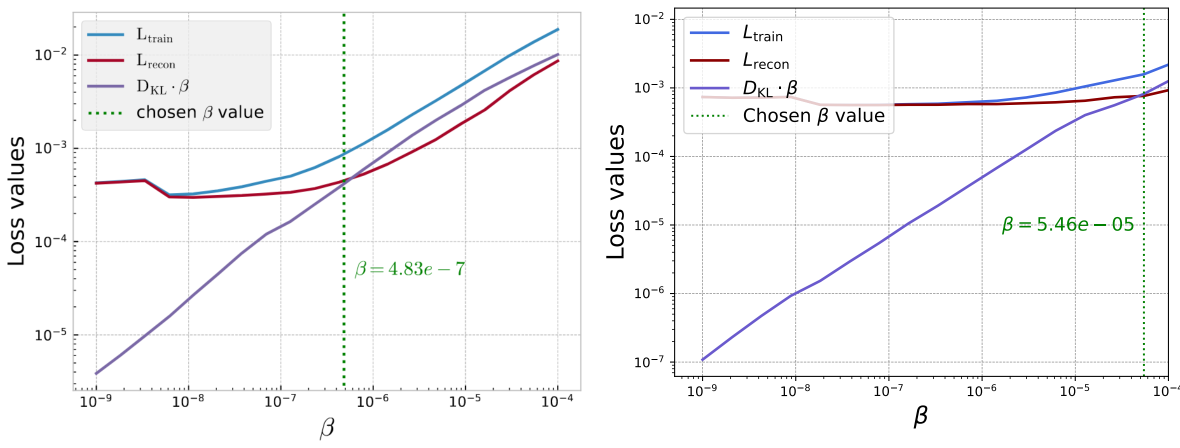
The losses of the original VAE using the old loss function definition with RMSE and the Gaussian approximation for the KL divergence are shown below [4]:

$$\mathsf{L} = 1.14302 \cdot 10^{-3} \qquad \mathsf{L}_{\text{recon}} = 7.001 \cdot 10^{-4} \qquad D_{\text{KL}} = 915.21 \tag{5.9}$$

To re-tune the original VAE with new loss functions, first the losses are re-calculated for reference. For this, the weights and biases of the original VAE are loaded, after which new losses are calculated using the functions defined in Equation (4.10). These are presented together with the original tuning plot in Figure 5.11a.

By comparing the new loss function values with the old ones, it can be seen that the reconstruction loss is of the same order of magnitude, while the KL divergence loss is significantly different. This underlines the need to re-tune β for the Base VAE consistent with the new loss function definitions, such that it can be used as a starting point for SR integration.

Using a tuning process identical to that of Swannet et al. [5], a plot is generated where the beta ranges from 10^{-9} to 10^{-3}, so that a β can be chosen that balances KL divergence and data reconstruction. The original tuning plot used for the Base VAE, as well as the re-tuned version to replicate the original VAE is shown together with the losses at the tuned β in Figure 5.11.



(a) $\mathsf{L} = ?$ $\quad \mathsf{L}_{\text{recon}} = 4.1904 \cdot 10^{-4}$ $\quad D_{\text{KL}} = 14.3987$ **(b)** $\mathsf{L} = 1.5810 \cdot 10^{-3}$ $\quad \mathsf{L}_{\text{recon}} = 7.63676 \cdot 10^{-4}$ $\quad D_{\text{KL}} = 14.982$

Figure 5.11: β tuning plots from Swannet et al. [5] and new plot following identical tuning procedure, including the losses at the chosen β value

As expected, the β-tuning plots are different. This stems from the changed scale of KL divergence. Due to the changed loss functions, the overall loss and β of the original VAE with new loss functions is not known. However, even though the same exact procedure has been used, the VAE losses at $\beta = 5.46 \cdot 10^{-5}$ does not seem to resemble the original VAE. While KL divergence is similar to the one recalculated for the original VAE, there is an 82% increase in reconstruction loss. This difference is further exemplified in the new latent effects plot of the 'tuned' VAE, as shown in Figure 5.12.

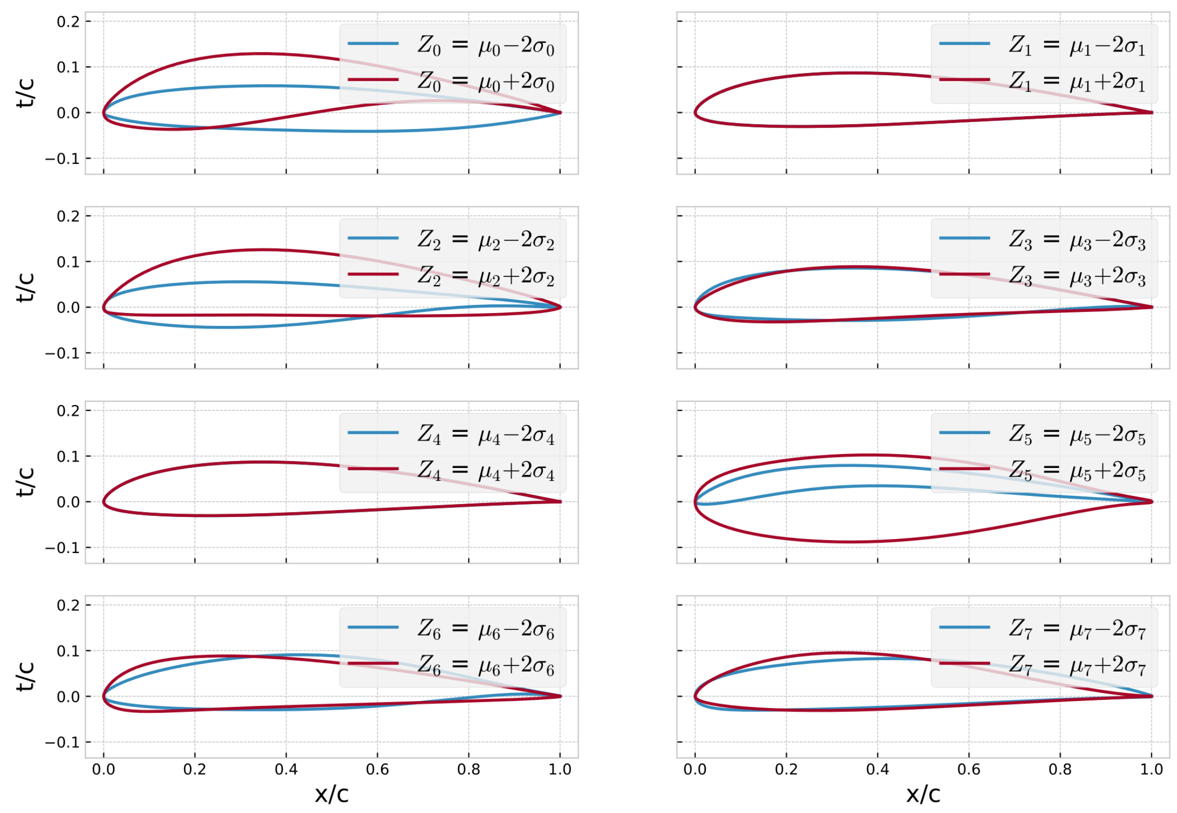
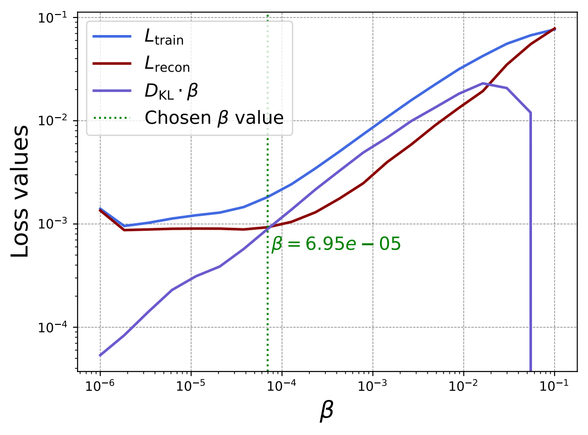
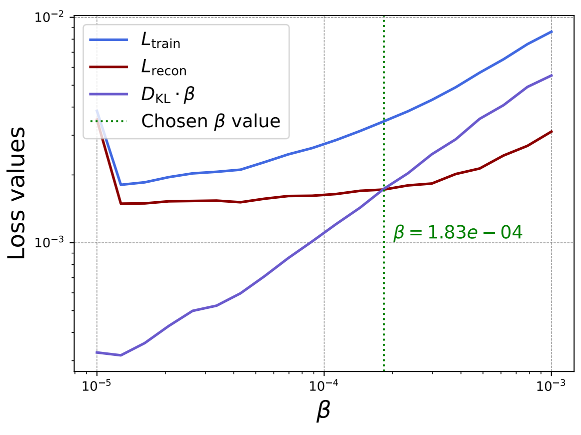


Figure 5.12: Latent effects plot of VAE with $\beta = 5.4556 \cdot 10^{-5}$, from identical Base VAE tuning procedure in new code

By comparing this with Figure 5.7, it is clear that this model does not resemble the latent effects of the original model at all. There have been many more attempts but none of these resulted in the same VAE that resembled the latent effects of Figure 5.7 or a reconstruction loss remotely close to the one listed in Figure 5.11a. The β-ranges have then been slightly shifted to better capture the losses, the resulting plots are shown in Figure 5.13.



(a) L $= 1.8255 \cdot 10^{-3}$ L$_{\text{recon}}$ $= 9.2971 \cdot 10^{-4}$ $D_{\text{KL}} = 12.88560$



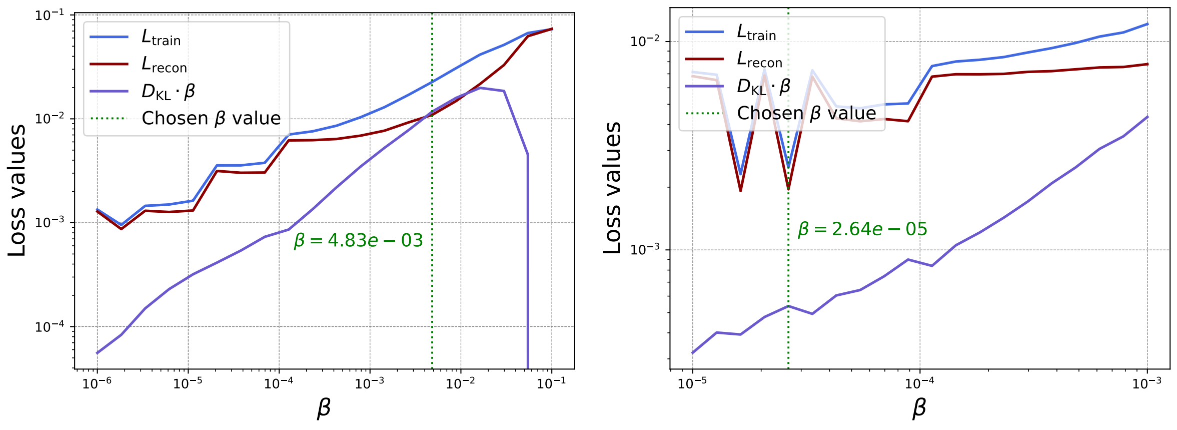
(b) L $= 3.489 \cdot 10^{-3}$ L$_{\text{recon}}$ $= 1.7226 \cdot 10^{-3}$ $D_{\text{KL}} = 9.47145$

Figure 5.13: β tuning plots using identical procedure to Base VAE, starting at higher β and at smaller ranges

There is a significant discrepancy between the chosen β values in these tuning plots; this occurs even if two β-trials are conducted back-to-back with identical conditions. However, comparing the losses found in Figure 5.13a with Figure 5.13b, does reveal that having the β-trial start at a lower β consistently leads to lower reconstruction losses. In Figure 5.13b the reconstruction loss never lowers below 10^{-3}, while this does happen in the other plots around the same range of β.

5.2.2. Implications of warm-started β-sweep

In the earlier tuning plots, the VAE models have not been properly re-initialized after every β evaluation. Consequently, the next β in the tuning procedure starts with the optimal weights and biases of the previous evaluation. The normal convention is to properly re-initialize such that there are no weights and biases defined at the start of each evaluation [81]. This ensures that later β-values are not biased by solutions optimized for a different β and that the VAE can be replicated by training a model from scratch with the same hyperparameters. Conducting such a β-trial leads to the plots defined in Figure 5.14.



(a) L $= 2.2644 \cdot 10^{-2}$ $\text{L}_{\text{recon}} = 1.092 \cdot 10^{-2}$ $D_{\text{KL}} = 2.4259$ **(b)** L $= 2.492 \cdot 10^{-3}$ $\text{L}_{\text{recon}} = 1.954 \cdot 10^{-3}$ $D_{\text{KL}} = 20.393$

Figure 5.14: β tuning plots where VAE is properly re-initialized at every evaluation, starting from higher β, with losses at chosen β value

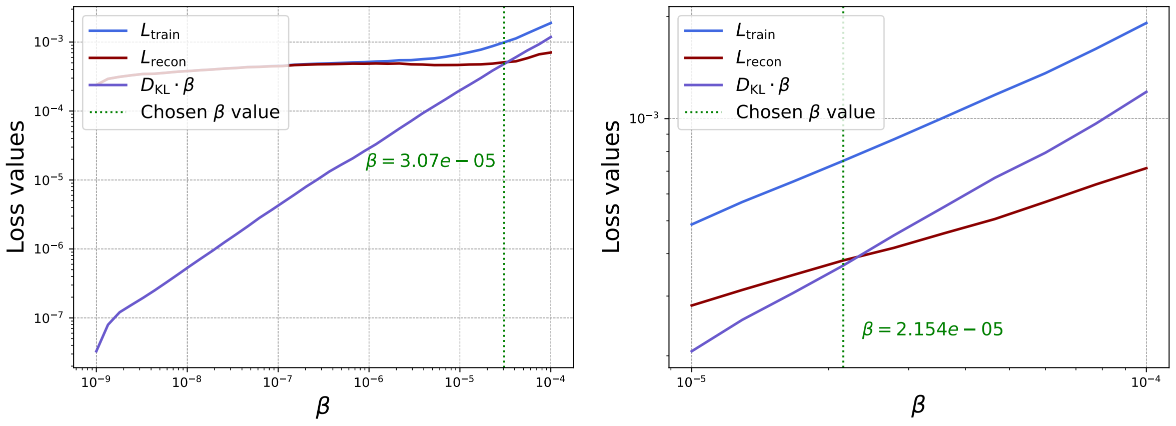
As can be seen here, the trends are much more erratic and less smooth than the plots defined in Figure 5.11 and Figure 5.13. More importantly, the intersection between reconstruction loss and KL divergence occurs at a much higher β, leading to higher 'tuned' losses.

Since Swannet et al. [5] implemented warm start in their β-sweep, the only way to obtain a similar VAE would theoretically be by doing the same. However, as established from the β-plots in Figure 5.11b and 5.13, this does not lead to a similar VAE or one with similar reconstruction performance. There are too many factors that influence the VAE training, even with identical hyperparameters and tuning procedures the probability of finding the same model, or even a similar one, is very low. The main reason for this is the inherent stochasticity involved with VAE training.

5.2.3. Re-tuning with warm start

Instead of using a warm-started β-sweep or ensuring that there are no weights and biases at the start of every iteration of the β-sweep, the Base VAE is tuned by always starting with the same weights and biases of the original VAE for every evaluation. This ensures that the tuned VAE will converge to a point similar to that of the original VAE, which increases its predictability. Once again a tuning plot is generated, for the same β-range and one with a smaller range to more accurately determine the tuned β, as shown in Figure 5.15.

As can be seen from the figure, starting the β-trial at a higher β has no effect anymore on the losses, since the effect of the warm-started β-sweep is eliminated by ensuring that every run starts with the same weights and biases. Furthermore, the reconstruction losses are now finally comparable to that of Figure 5.11a, with the Base VAE at $\beta = 2.154 \cdot 10^{-5}$ even exhibiting a better reconstruction loss. For this reason, and since this VAE also has an overall lower loss than the VAE of Figure 5.15a, this is chosen as the Base VAE for subsequent analyses.



(a) L $= 9.920 \cdot 10^{-4}$ L$_{recon}$ $= 5.097 \cdot 10^{-4}$ $D_{KL} = 15.708$ **(b)** L $= 7.512 \cdot 10^{-4}$ L$_{recon}$ $= 3.821 \cdot 10^{-4}$ $D_{KL} = 17.135$

Figure 5.15: β tuning plots which starts with weights and biases from Swannet et al. for every β-evaluation with identical conditions starting from higher β, with losses at chosen β value

Analysis of tuned Base VAE

To properly establish the Base VAE as a baseline, its performance must be assessed. For this, a latent effects plot is generated together with that of the Base VAE. This is shown in Figure 5.16.

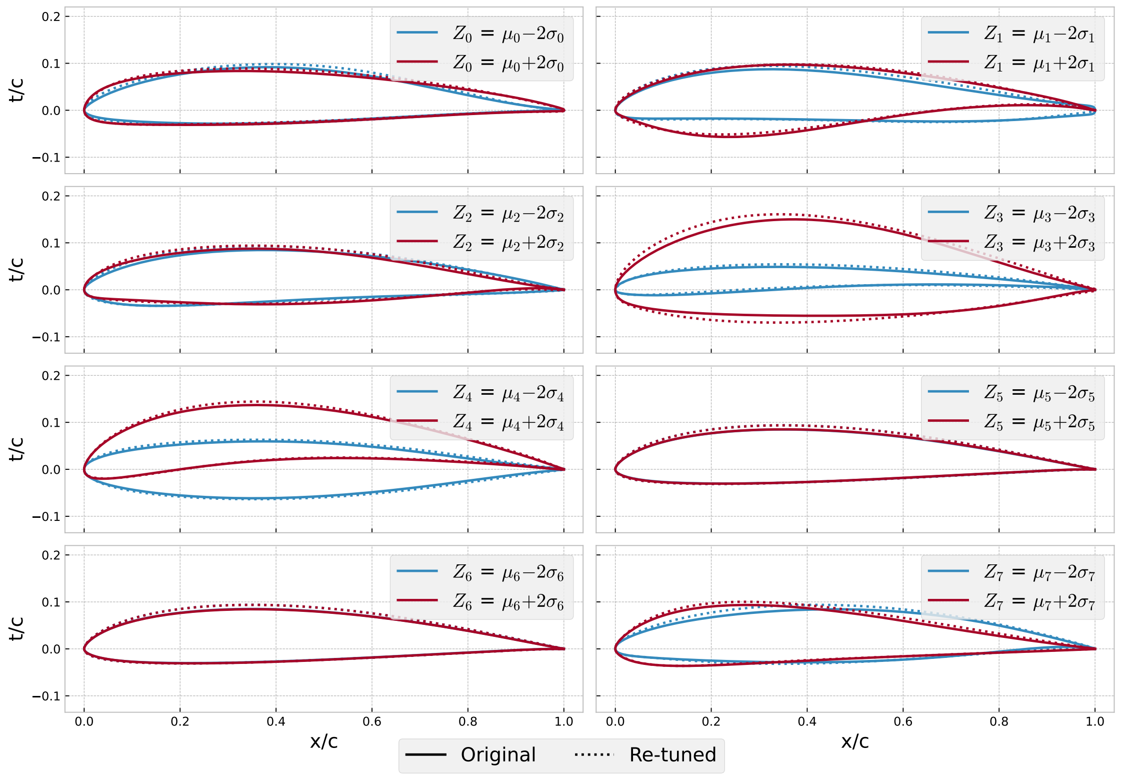


Figure 5.16: Latent effects plot of tuned Base VAE at $\beta = 2.154 \cdot 10^{-5}$, plotted together with the original from Swannet et al. [4]

Overall, the VAE now captures the same features per latent parameter. However, there are some subtle differences. Considering Z_5 and Z_6, it seems that the mean of the tuned Base VAE has a slightly increased airfoil thickness. The latent parameter for thickness, Z_3, is more pronounced for the re-tuned VAE compared to the original VAE. Overall, it seems that the latent effects are slightly more pronounced. This is confirmed in the new Pearson correlation matrix, which can be found in Appendix A.3.

As briefly discussed in Chapter 4, taking the validation losses into account is a good preliminary indicator for performance but not a definitive metric, especially for VAE models that perform very similarly. Therefore, the validation plot for this Base VAE is included below, this allows for visual inspection of the worst reconstructed airfoils.

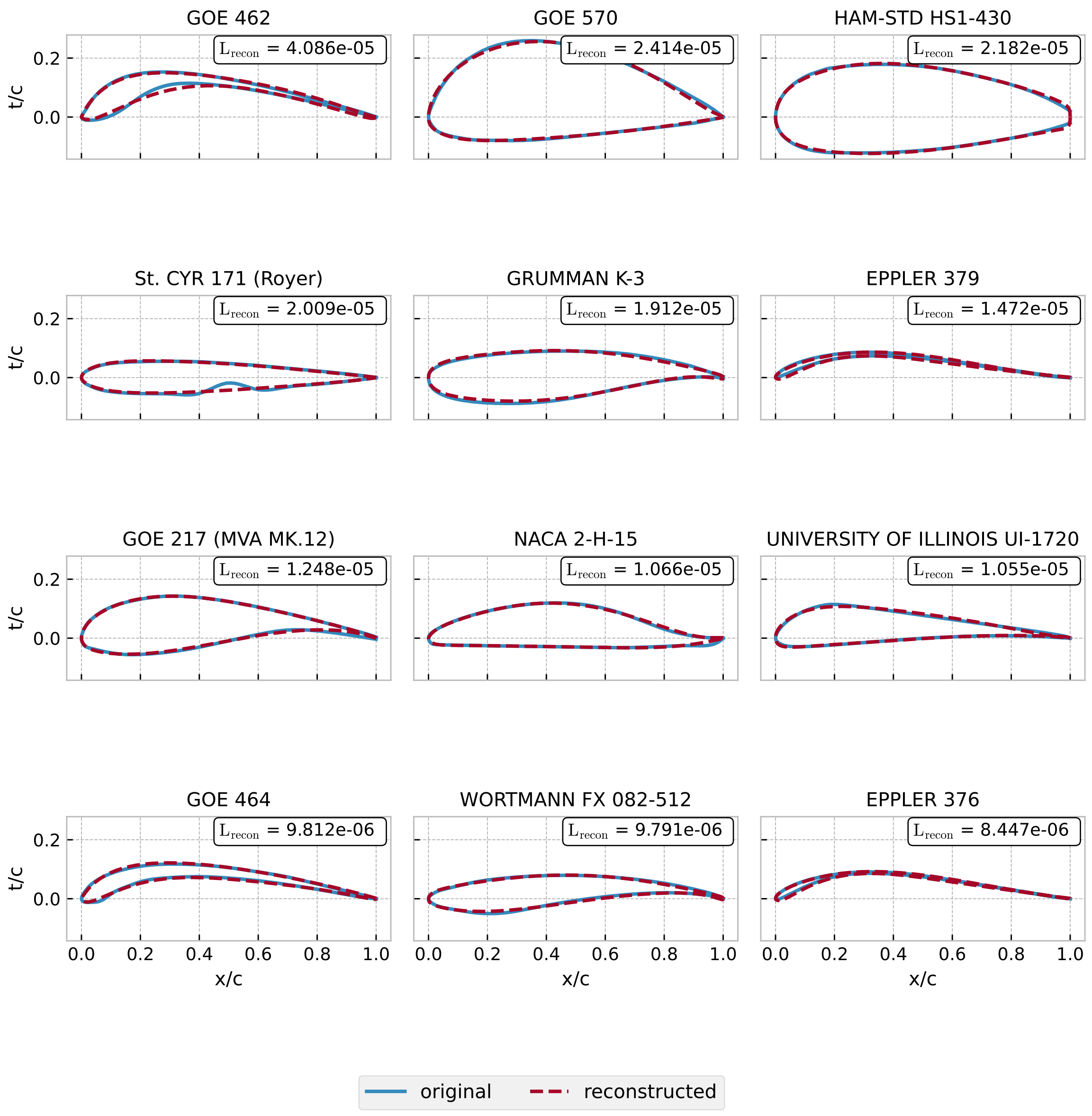


Figure 5.17: Validation plot with worst performing airfoil of the Base VAE with $\beta = 2.154 \cdot 10^{-5}$, which is the original VAE re-tuned with new loss functions

Now, directly comparing this with the validation plot of the original VAE shown in Figure A.2, shows that this VAE is significantly better at handling outliers. Remarkably, thick airfoils such as the AH 93-W-480B and FX 79-W-660A are not among the worst reconstructed anymore. There are also no visible inaccuracies, except for the GOE-462, which is not among the worst airfoils of the original VAE. The only common airfoil between both plots is the GRUMANN K-3, where the new tuned Base VAE outperforms the original. In fact, overall the highest MSE losses are quite a bit lower than those of the original VAE.

Re-tuning the original VAE into the new architecture already resulted into a significant improvement. This warrants the question of how much the VAE can be improved by retraining only, without the integration of SR into VAE training. This is important to quantify to isolate the effect of symbolic regression.

5.2.4. Effect of repeated warm starts

This effect is investigated by repeatedly retraining this tuned Base VAE and investigating the effect on its losses. For this, the Base VAE has been retrained 6 times in a trial. The trial starts with the weights and biases of the Base VAE, and then keeps using the optimal weights and biases of the last iteration. This is similar to the aforementioned warm-started β-sweep, only this time β is kept constant. The plot is shown in Figure 5.18, with losses of the optimal VAE, henceforth called Retrained VAE, below.

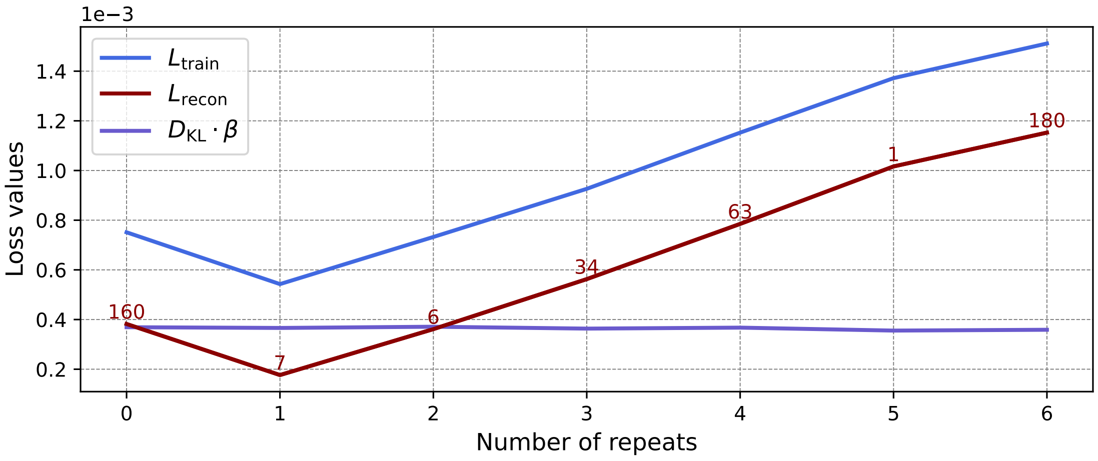


Figure 5.18: Plot of VAE losses as a result of repeated warm starts with $\beta = 2.154 \cdot 10^{-5}$, includes the epochs the model converged at

$$\text{L} = 5.4268 \cdot 10^{-4} \qquad \text{L}_{\text{recon}} = 1.7651 \cdot 10^{-4} \qquad D_{\text{KL}} = 16.999 \tag{5.10}$$

As seen here, the KL divergence is virtually constant, which is partially due to β being constant and the early convergence at low epochs. The constant β means that regularization is not changed, so the network does not have much incentive to explore different latent encodings, which is further confirmed in Figure 5.20. However, despite the early convergence, the repeated warm starts do have a significant effect on the reconstruction loss. After the first iteration, the reconstruction loss lowers significantly, but immediately sees an equally significant increase afterwards. This is likely attributed to the effect of the learning rate and the retraining with warm starts of previously converged weights. The retraining allows the optimizer to escape shallow local minima due to the favorable weights from the previous iteration. However, as the retraining continues with the same learning rate, the model likely overshoots or oscillates near the optimal solution, which prevents further improvement and even shows degradation.

5.2.5. Baseline selection

Despite the improvement in losses from a repeated warm start, the re-tuned VAE is chosen as the Base VAE as it is much closer to the original. Furthermore, repeated warm starts confirmed that retraining the tuned VAE once leads to an overall better model, but retraining it again will likely degrade VAE performance. The models are tested for the loss metrics from Subsection 4.4.5, as shown in Table 5.1.

Table 5.1: Loss metrics of Original VAE, Base VAE and Retrained VAE

MSE Averages	**Original VAE** [4]	**Base VAE**	**Retrained VAE**
Outliers UIUC	$4.7068 \cdot 10^{-5}$	$1.4684 \cdot 10^{-5}$	$7.1020 \cdot 10^{-6}$
Outliers AFBench	$4.9238 \cdot 10^{-5}$	$4.6308 \cdot 10^{-5}$	$4.1192 \cdot 10^{-5}$
Overall UIUC	$1.7394 \cdot 10^{-6}$	$5.6355 \cdot 10^{-7}$	$4.3407 \cdot 10^{-7}$
Overall AFBench	$4.2995 \cdot 10^{-6}$	$3.9272 \cdot 10^{-6}$	$3.7733 \cdot 10^{-6}$

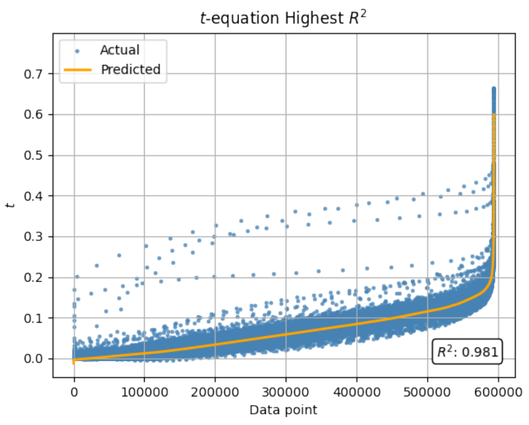
Terminology: henceforth, the term *Base VAE* refers to the Original VAE retrained once at re-tuned beta and is used as a warm-start for all SR-VAE variants. Note that this already performs much better than the original. The *Retrained VAE* refers to the same architecture retrained once more, which consistently yielded an even lower reconstruction loss. This retrained model is used as the primary baseline when comparing decoder performance, to isolate effects of SR from VAE retraining.

5.3. Establishing base SR performance

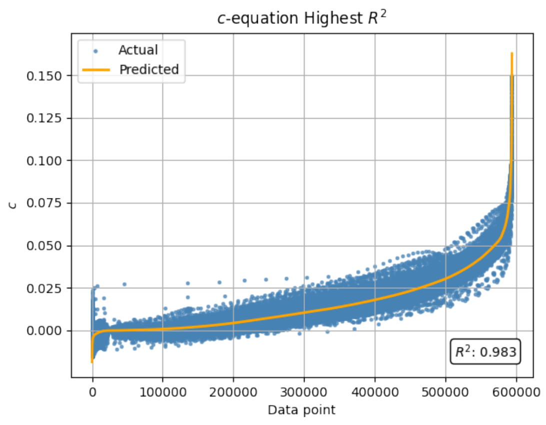
Alongside a base for VAE performance, a base for SR performance is also required. This will enable the quantification of the potential improvement in SR performance from the integration of SR into VAE training. The results of the latent analyses can directly be used to establish this baseline. However, as can be seen by comparing Figure 4.2 and 4.5, an important distinction here is that the actual airfoil properties must be used to assess the equations, rather than extracting airfoil properties from the decoded output. This ensures consistency of the test sets used for SR-VAE equation selection. More importantly, the SR equations of the latent analysis are assessed with the loss metrics defined in Subsection 4.4.5.

5.3.1. Equation plots

Due to the changed test sets, the R^2 score has to be recalculated and the test data fit plots from the latent analysis have to be generated again. The new plots corresponding to the equations with highest R^2 from the latent analysis are shown in Figure 5.19.



(a) Plot with adjusted test sets for Equation 5.6



(b) Plot with adjusted test sets for Equation 5.8

Figure 5.19: Predicted values of parameterization equations from latent analysis with highest R^2 plotted against adjusted test sets, sorted by ascending predicted values

The thickness distribution plot shown in Figure 5.19a is nearly identical to the initial plot shown in Figure 5.5b, and only sees a small decrease in overall R^2. Remarkably, compared to Figure 5.6b, the camber distribution plot in Figure 5.19b shows far fewer outliers and an improvement in R^2, even though the equations have not been explicitly trained on the actual properties of the UIUC and AFBench dataset. The most likely reason for this behavior is that the equations are not unjustly penalized here by the limited outlier performance of the decoder.

5.3.2. Evaluation

Similar to Table 5.1, the established average loss metrics from Subsection 4.4.5 are used to evaluate the SR equations with the UIUC and AFBench dataset, the results are shown in Table 5.2.

Table 5.2: Average MSE loss metrics of SR equations from latent analysis and original VAE

Outliers UIUC	Outliers AFBench	UIUC	AFBench
$1.1133 \cdot 10^{-3}$	$4.8452 \cdot 10^{-4}$	$2.6896 \cdot 10^{-5}$	$2.1726 \cdot 10^{-5}$

As expected, the MSE values are significantly worse than what is shown for the original decoder in Table 5.1. While the original VAE has similar outlier performance for the UIUC and AFBench test sets, the parametric equations are significantly better at reconstructing outliers from AFBench than from UIUC. Similarly, the overall performance of the parametric equations are better for AFBench than UIUC, whereas the opposite is true for the original VAE decoder. While the overall performance of the parametric equations is significantly worse, it does seem that there is no apparent bias towards the UIUC dataset. Understandably, there does seem to be a bias towards UIUC for the original VAE.

5.4. SR implementation in VAE training

This section describes the results of the implementation of SR in VAE training for different SR integration strategies. With the established Base VAE and SR performance results, the integration of SR in VAE training can now be further investigated and tested with the same model evaluation loss metrics. The results in this section are the culmination of more than 100 runs, from which an overview of the best SR-VAE models and associated hyperparameters can be found in Appendix C. In the end, the final SR-VAE will be presented as a result of the model selection. All SR-VAE models presented in this section use Base VAE for warm start with identical hyperparameters of $\beta = 2.154 \cdot 10^{-5}$ and $\gamma = 1$ if applicable. Table 5.1 and 5.2 will be used as a baseline for the comparison of the decoder and SR performance.

5.4.1. Per epoch

Integrating SR into the training per epoch and subsequently using the same equations to calculate the SR loss for all batches within this epoch, is a compromise between interpretability through SR and VAE training efficiency. The training of the best performing SR-VAE model with this integration strategy stopped at a relatively early epoch of 101, and displayed the following validation losses:

$$\mathsf{L} = 4.7265 \cdot 10^{-3} \qquad \mathsf{L}_{\mathsf{recon}} = 2.1487 \cdot 10^{-4} \qquad D_{\mathsf{KL}} = 17.441 \qquad \mathsf{L}_{\mathsf{SR}} = 4.1359 \cdot 10^{-3} \tag{5.11}$$

Compared to the Base VAE losses shown in Figure 5.15b, there is a clear decrease in reconstruction loss, but a slight increase in KL divergence. Furthermore, the losses are higher compared to the Retrained VAE shown in (5.10), which showed a slight decrease in D_{KL} instead of an increase. This is likely attributed to SR, which shifts the focus of VAE training from minimizing D_{KL} to minimizing SR loss. To investigate the effect on the latent space, a latent effects plot is shown in Figure 5.20, including the latent effects of the Base VAE and retrained VAE from Subsection 5.2.4 for reference.

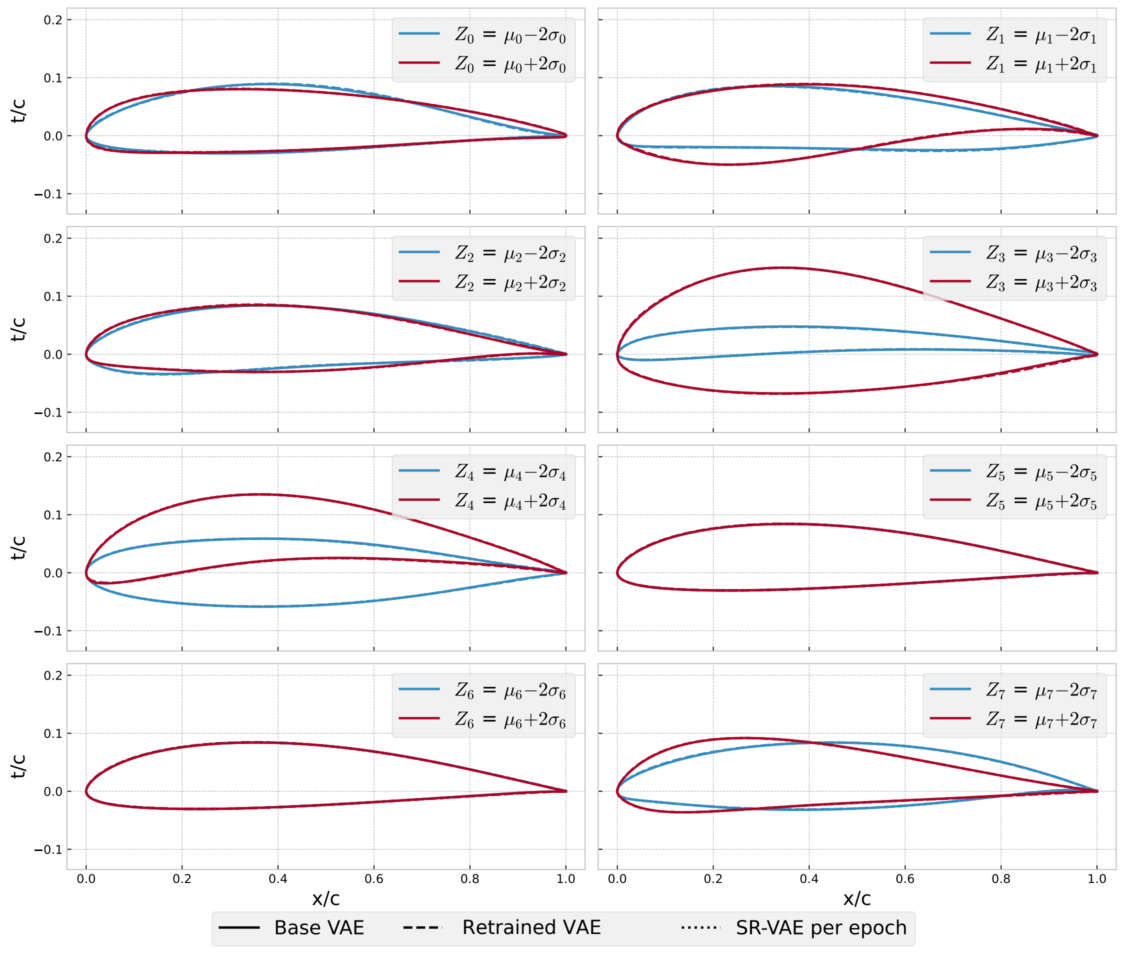


Figure 5.20: Latent effects plot of SR-VAE per epoch with tuned Base VAE and Retrained VAE for reference

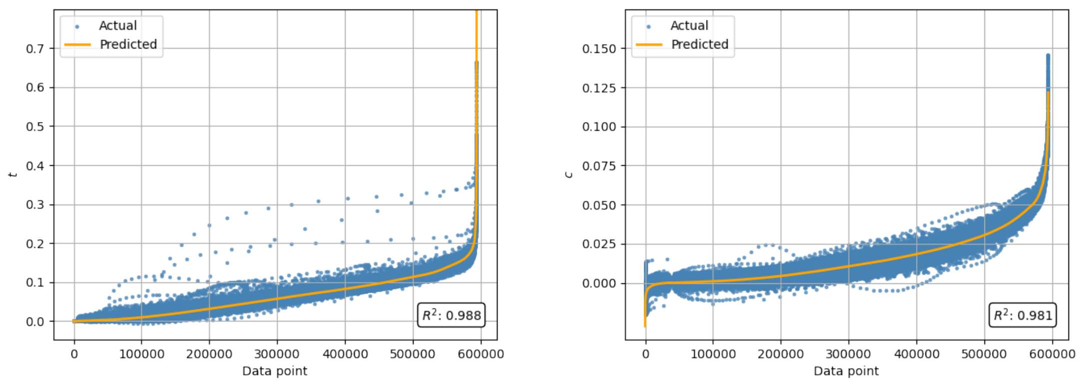
Although the losses improved significantly, neither the repeated warm start of the Base VAE nor the integration of SR-VAE has had any significant effect on the latent parameters. However, zooming in enough does show the dashed airfoils from the Retrained VAE, whereas the SR-VAE per epoch is not visible at all. The exact reason for this is unclear, but this indicates that the overall improved VAE performance of this SR-VAE is likely due to retraining rather than SR integration. To further quantify VAE performance, the model is evaluated with established loss metrics as shown in Table 5.3.

Table 5.3: Average MSE loss metrcis for SR-VAE decoder, where SR is integrated in the training per epoch

Outliers UIUC	Outliers AFBench	UIUC	AFBench
$7.7031 \cdot 10^{-6}$	$4.4010 \cdot 10^{-5}$	$4.4039 \cdot 10^{-7}$	$3.8146 \cdot 10^{-6}$

Considering Table 5.1, it can once again be seen that there is a significant improvement over the Base VAE but slightly worse performance compared to the Retrained VAE. Overall, the differences are more subtle for AFBench than for UIUC, which is likely attributed to the much larger size of the dataset.

Regardless of VAE performance, the main results for SR integration are the new thickness and camber distribution equations. These are shown below, with a plot indicating the fit with test data.



(a) t-equation highest R^2 **(b)** c-equation highest R^2

Figure 5.21: Predicted values of parameterization equations from SR-VAE with SR trained per epoch, with the lowest overall MSE plotted against the actual values of adjusted test sets, sorted by ascending predicted values

$$\begin{aligned} f_{t,epoch} &= (Z_3 \cdot -0.0073 - 0.0213) \cdot (X \cdot Z_3 + Z_1 \\ &\cdot 0.2635 + (0.5435 - 1.3395 \cdot X) \cdot (Z_1 \cdot (X + X) \\ &+ Z_7 \sin(-3.1177 \cdot X + Z_0 \cdot 0.5422)) \\ &+ 4.9740) \cdot \sin(X^{0.5533} \cdot -3.1304) \end{aligned} \tag{5.12}$$

$$\begin{aligned} f_{c,epoch} &= (Z_4 + (X - 0.4442) \cdot (Z_1 + (X \cdot (X + Z_0 \\ &+ Z_1 - 0.6650 \cdot \sin(Z_7)) \cdot -1.4590 + Z_2 + Z_3 + Z_7) \\ &\cdot -0.4464) + \sin((Z_3 + Z_4 \cdot Z_4) \cdot 0.1362) + 1.0242) \\ &\cdot -0.0199 \cdot \sin(X^{0.8020} \cdot -3.1263) \end{aligned} \tag{5.13}$$

Although the thickness distribution equation shown in Figure 5.21a has a near identical complexity of 45, it shows a much better fit with a higher R^2 compared to Figure 5.19a. The outliers are still present, but much closer to the predicted values than before. The camber distribution equation from Figure 5.21b appears to be slightly worse than what was shown in Figure 5.19b, as is evident from the outliers and the R^2 score. However, this is likely attributed to the reduced complexity of 50 as opposed to 59. SR performance can be further investigated with the loss metrics in Table 5.4.

Table 5.4: Average MSE loss metrics of parametric equations from SR-VAE where SR is integrated in the training per epoch

Outliers UIUC	Outliers AFBench	UIUC	AFBench
$6.1631 \cdot 10^{-4}$	$3.0136 \cdot 10^{-4}$	$1.7263 \cdot 10^{-5}$	$1.5961 \cdot 10^{-5}$

Compared to Table 5.2, the overall MSE values are significantly lower, which means that the SR-VAE has parametric equations that are much better at reconstructing the airfoils than the equations from the latent analysis. However, compared to VAE performance shown in Table 5.3, the overall SR performance is still significantly worse. Similarly to the decoder, the equations perform better for the AFBench test set than for the UIUC test set.

5.4.2. Per batch

Although it is the most computationally intensive, training a VAE where an SR search is conducted for every batch ensures that interpretability through SR is evaluated at each optimization step. The training of the SR-VAE analyzed in this section stopped at an epoch of 344 with a patience of 250, ensuring that the VAE properly converged. The following validation losses are observed at this epoch:

$$\mathsf{L} = 5.3455 \cdot 10^{-3} \qquad \mathsf{L}_{\mathsf{recon}} = 2.6412 \cdot 10^{-4} \qquad D_{\mathsf{KL}} = 17.324 \qquad \mathsf{L}_{\mathsf{SR}} = 4.7082 \cdot 10^{-3} \tag{5.14}$$

Once again, there is a significant decrease in reconstruction loss, but a slightly higher KL divergence compared to Figure 5.15b. The validation losses indicate that performance of the decoder is worse than that of the SR-VAE per epoch, which already performed slightly worse than the Retrained VAE. This reduced performance is likely due to a decrease in training stability, which is a consequence of the increased number of SR searches that are now conducted at every batch. This is also expected to have a more pronounced effect on the latent space than in the SR-VAE per epoch. To verify this, the latent effects plot of this SR-VAE is compared to that of the Base VAE below.

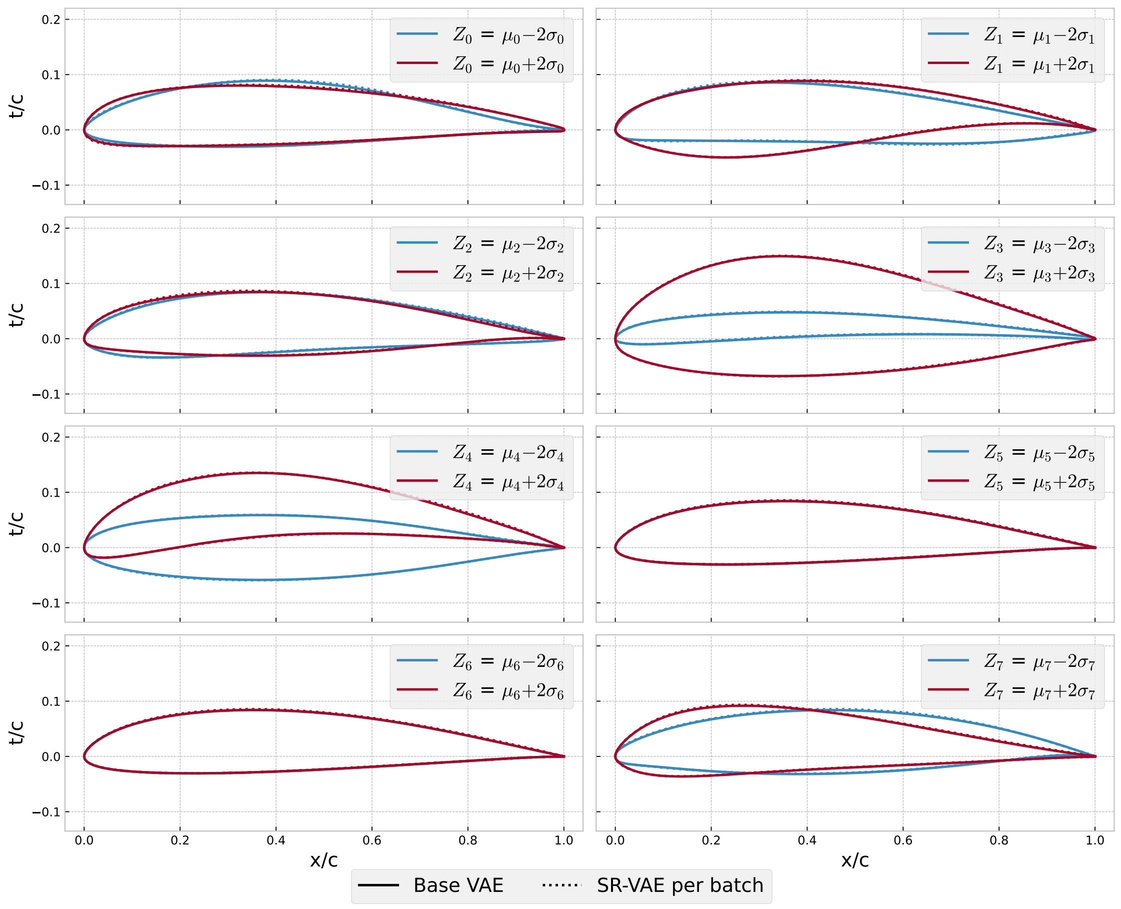


Figure 5.22: Latent effects plot of SR-VAE per batch with tuned Base VAE for reference

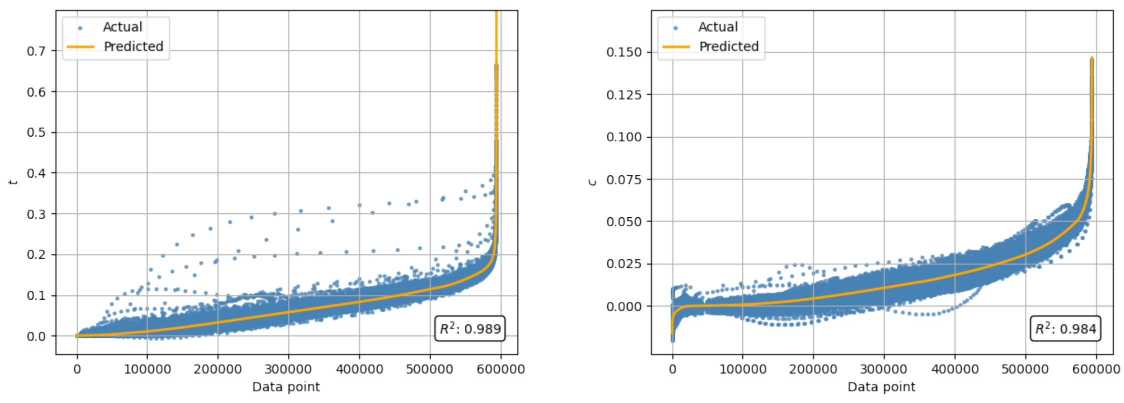
Although minor, this time there seems to be a slight change in the latent effects plot for the SR-VAE. However, this change is too small to assume that this is due to the integration of SR. It could also simply be because this model converged much later than the SR-VAE per epoch. Therefore, once again this does not prove that the lower validation losses are caused by SR integration. The performance of the decoder is further assessed with the metrics shown in Table 5.5.

Table 5.5: Average MSE loss metrics of SR-VAE decoder, where SR is integrated in the training per batch

Outliers UIUC	Outliers AFBench	UIUC	AFBench
$8.8292 \cdot 10^{-6}$	$4.2552 \cdot 10^{-5}$	$4.5790 \cdot 10^{-7}$	$3.7921 \cdot 10^{-6}$

The loss metrics confirm the observations from the validation losses. Compared to the SR-VAE per epoch variant, the average decoder losses of this SR-VAE are higher for the UIUC dataset but slightly lower for AFBench. Overall, the decoder loss metrics are still much higher than the Retrained VAE. Therefore, once again it can be deduced that the improved average losses over the Base VAE is likely not due to the integration of SR. If anything, the extra SR term in Equation (4.11) seems to slightly worsen the performance of the decoder. This is likely due to SR introducing a source of instability during training and pulling away the focus of the optimizer from minimizing VAE-specific losses.

Theoretically, the increase in SR searches during VAE training should lead to better SR equations. The equations with lowest MSE or highest R^2 are assessed with the test data in Figure 5.23.



(a) t-equation highest R^2 **(b)** c-equation highest R^2

Figure 5.23: Predicted values of parameterization equations from SR-VAE with SR trained per batch, with the lowest overall MSE plotted against the actual values of adjusted test sets, sorted by ascending predicted values

$$\begin{aligned} f_{t,batch} &= (Z_1 + Z_3 + (Z_1 \cdot (X + 0.2759) + Z_3 \\ &\cdot -1.1097 + Z_4 + Z_7 + (X \cdot Z_1 + Z_7) \cdot 2.1315) \\ &\cdot \sin((X + 1.0625) \cdot 2.1779) + 17.5325) \cdot 0.0021 \\ &\cdot \sin(X^{0.4649} \cdot 2.1339 + X) \cdot (Z_3 + 2.8507) \end{aligned} \tag{5.15}$$

$$\begin{aligned} f_{c,batch} &= X \cdot \sin(((Z_4 \cdot 0.2103 + 2.1126) \cdot (Z_3 \cdot 0.1166 \\ &+ Z_4 + (X - 0.4434) \cdot (Z_0 \cdot 0.1336 + Z_1 + (X - 0.9275) \\ &\cdot (Z_2 + 1.7101) + (Z_3 \cdot 0.4908 + Z_4 + Z_7) \cdot -0.5555)) \\ &+ 2.1619) \cdot (X - 0.9986) \cdot -0.0390) \end{aligned} \tag{5.16}$$

Considering the R^2 scores, there appears to be an improvement in test data fit over the SR-VAE per epoch. Figures 5.23a and 5.21a are nearly identical, apart from a very subtle horizontal translation to the right for the outliers in the former, which agrees with the R^2 scores. The differences in the camber plots shown in Figure 5.23b and Figure 5.21b are more visible, where the former has fewer outliers at higher cambers and a smaller overall spread around the predicted values. The reconstruction performance of the SR equations can be even better quantified using the loss metrics shown in Table 5.6.

Table 5.6: Average MSE loss metrics of parametric equations from SR-VAE where SR is integrated in the training per batch

Outliers UIUC	Outliers AFBench	UIUC	AFBench
$6.5806 \cdot 10^{-4}$	$2.9258 \cdot 10^{-4}$	$1.6976 \cdot 10^{-5}$	$1.4345 \cdot 10^{-5}$

As expected, the average MSE losses are once again lower than those of the equations from the latent analysis shown in Table 5.2. The more interesting comparison is with the SR-VAE per epoch, for which the average MSE losses are shown in Table 5.4. Although it shows slightly worse performance for UIUC outliers, the parametric equations from the SR-VAE seem to perform better for all other losses. This is contrary to what one would expect considering the validation losses presented in Equation (5.11) and Equation (5.14), where the former showed a lower L_{SR}. This reinforces the notion that validation losses are a good preliminary indication of performance but not a definitive loss metric.

Similarly to what was observed for the SR-VAE per epoch variant, the parametric equations seem to perform slightly better for AFBench compared to UIUC, indicating that they are generalizable, while the decoder shows a clear preference for the UIUC dataset. Overall, however, the decoder still performs much better than the SR equations.

5.4.3. Fixed SR

While the SR-VAE per batch yielded more accurate parametric equations, it worsened overall decoder performance. One of the causes identified in the previous section was the instability introduced by the many SR searches. A possible way to mitigate this is to take the SR-VAE that yielded accurate parametric equations, and retrain this by fixing these parametric equations for the SR loss calculations. This allows the optimizer to minimize VAE-specific losses without the instability introduced by SR. The SR loss calculations with fixed equations should also help the optimizer to more effectively train the VAE to fit the equations, or at least prevent the optimizer from degrading SR loss. The resulting model of training such a fixed SR-VAE, using the SR-VAE per batch and the corresponding equations as a warm start, converged at an epoch of 349 and showed the following validation losses:

$$\mathsf{L} = 4.1378 \cdot 10^{-3} \qquad \mathsf{L}_{\mathsf{recon}} = 2.1331 \cdot 10^{-4} \qquad D_{\mathsf{KL}} = 17.11 \qquad \mathsf{L}_{\mathsf{SR}} = 3.5560 \cdot 10^{-3} \tag{5.17}$$

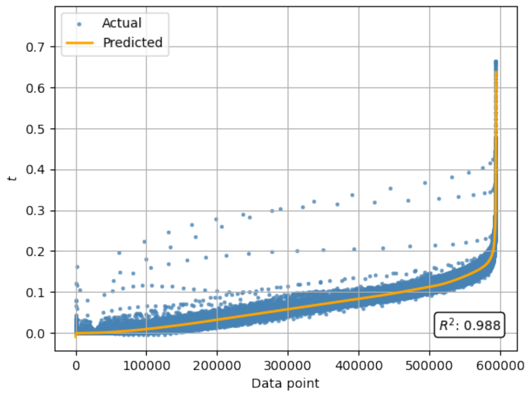
At first glance, the validation losses in Equation (5.17) already seem to indicate that both the decoder and SR performance improved over the losses of the SR-VAE per batch from (5.14). The effect on the latent space is negligible. Earlier sections already indicated that neither warm start retraining nor SR integration has any bearing on the behavior of the latent parameters, which is confirmed in Figure C.1. VAE performance can be further assessed with the MSE metrics shown in Table 5.7.

Table 5.7: Average MSE loss metrics of SR-VAE decoder, where SR-VAE per batch is retrained by fixing the best SR equations

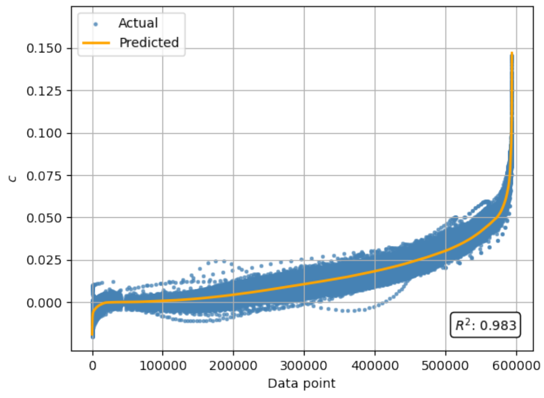
Outliers UIUC	Outliers AFBench	UIUC	AFBench
$5.6806 \cdot 10^{-6}$	$4.2636 \cdot 10^{-5}$	$3.8292 \cdot 10^{-7}$	$3.7826 \cdot 10^{-6}$

Compared to Table 5.5, there is indeed a significant improvement in decoder performance, except for a slightly higher MSE value for the AFBench metrics. The decoder is now much better in handling UIUC outliers, and shows substantial overall UIUC improvement as well. For the first time, the fixed SR-VAE also shows better performance over the Retrained VAE, with the exception of AFBench loss metrics where it shows similar performance. This is an important result, since it was identified in subsection 5.2.4 that retraining again without SR leads to degraded VAE performance.

For SR equation selection, the equations found during the training of the SR-VAE per batch are considered. This is because this model was used as a warm start for the fixed SR-VAE, which did not conduct any SR searches of its own. Theoretically, the best-performing parametric equations should correspond to those fixed during training. The optimal equations identified after training, based on R^2 and MSE, are presented in (5.18) and (5.19), with the test data fit shown in Figure 5.24.



(a) t-equation highest R^2



(b) c-equation highest R^2

Figure 5.24: Predicted values, of parameterization equations with the lowest overall MSE from retrained SR-VAE per batch with fixed equations, plotted against the actual values of adjusted test sets, sorted by ascending predicted values

$$\begin{aligned}f_{t,fixed} &= (X^{0.5297} + X \cdot \mathsf{sin}(Z_3 \cdot 0.2051)) \cdot (Z_3 \\ &+ (X - 0.4344) \cdot (Z_0 \cdot (X + (X \cdot 0.9099)^{0.3567} \\ &- 1.0805) + (1.3503^{Z_4} + Z_1 + Z_3 \cdot 0.0971 + Z_7) \\ &\cdot -1.0801) + 3.2726) \cdot (X - 0.9973) \cdot -0.0821\end{aligned} \tag{5.18}$$

$$\begin{aligned}f_{c,fixed} &= X \cdot \mathsf{sin}(((Z_4 \cdot 0.2103 + 2.1126) \cdot (Z_3 \cdot 0.1166 \\ &+ Z_4 + (X - 0.4434) \cdot (Z_0 \cdot 0.1336 + Z_1 + (X - 0.9275) \\ &\cdot (Z_2 + 1.7101) + (Z_3 \cdot 0.4908 + Z_4 + Z_7) \cdot -0.5555)) \\ &+ 2.1619) \cdot (X - 0.9986) \cdot -0.0390)\end{aligned} \tag{5.19}$$

The thickness distribution equation shown in (5.18) is different from the equation that was fixed for training. The rationale for this choice is further elaborated in section C.4. The camber distribution equation shown in (5.19) does correspond to the one that was fixed, and matches the equation presented in (5.16) for the per batch variant of the SR-VAE. For both thickness and camber, the R^2 decreased slightly compared to before. Aside from the different R^2 scores, there are no visible differences in the plots. SR performance is further quantified in Table 5.8.

Table 5.8: Loss metrics of parametric equations from SR-VAE where SR-VAE per batch is retrained with fixed equations

Outliers UIUC	Outliers AFBench	UIUC	AFBench
$5.9367 \cdot 10^{-4}$	$2.3784 \cdot 10^{-4}$	$1.6312 \cdot 10^{-5}$	$1.4659 \cdot 10^{-5}$

Despite the lower R^2 scores, the SR performance of the fixed SR-VAE is better in almost every way, with the exception of the overall AFBench test set. This means that in the end, retraining the per batch SR-VAE with fixed equations improved both the performance of the decoder and SR equations. Compared to the baseline SR performance, the new equations exhibit MSE losses that are 30% to 50% lower.

5.4.4. β=0

Since neither of the previous SR training procedures revealed any effect on the latent space, SR-VAE training is attempted where $\beta = 0$. This means that there is no regularization from minimizing the KL divergence anymore. This ensures that any regularization effects visible after training is from the integration of SR. This also allows the optimizer to focus more on reconstruction and SR loss. Similar to previous section, once again the Base VAE is used as a warm start and SR is integrated per batch. This led to a VAE that converged at epoch 811 with the following associated validation losses:

$$\mathsf{L} = 3.7091 \cdot 10^{-3} \qquad \mathsf{L}_{\mathsf{recon}} = 1.70373 \cdot 10^{-4} \qquad D_{\mathsf{KL}} = 79.0117 \qquad \mathsf{L}_{\mathsf{SR}} = 3.5387 \cdot 10^{-3} \tag{5.20}$$

As expected, the KL-divergence shows a substantial increase. Furthermore, both the reconstruction loss and symbolic regression loss are lower than any of the past SR integration strategies. However, as mentioned in subsection 2.2.5, in VAE models there is often a tradeoff between disentanglement and reconstruction. The seemingly improved reconstruction could be at the cost of disentanglement [22]. To confirm this, the latent effects plot is shown in Figure 5.25

As shown, the changes in the latent effects of SR-VAE with $\beta = 0$ are once again negligible. The training converged at a relatively high epoch, so there was plenty of time for the optimizer to change the latent structure. The SR-VAE is probably stuck at a narrow basin near the optimum of the initial weights and biases, which prevents it from exploring different latent encodings. To confirm this, an identical model is trained without the integration of SR, this is shown in Figure 5.26.

Once again, no change in the latent effects is observed. This particular model also converged at a very early epoch of 21. This confirms that the lack of change within the latent space does not necessarily mean that SR has no regularization effects. This means that the lack of change for previous SR-VAE models is also attributed to the warm start, which prevents the optimizer from exploring a different latent space structure. Regardless, since SR-VAE with $\beta = 0$ does show promising validation losses, VAE performance is once again further investigated with loss metrics in Table 5.9.

Compared to all previously trained SR-VAE models, this decoder performs the worst in all categories except the overall UIUC dataset, where it shows the lowest MSE value so far. This is a perfect example of an SR-VAE that is overfit to the UIUC dataset, so much so that the outlier performance and AFBench performance degrades significantly. This underlines the importance of disentanglement through the minimization of the KL divergence.

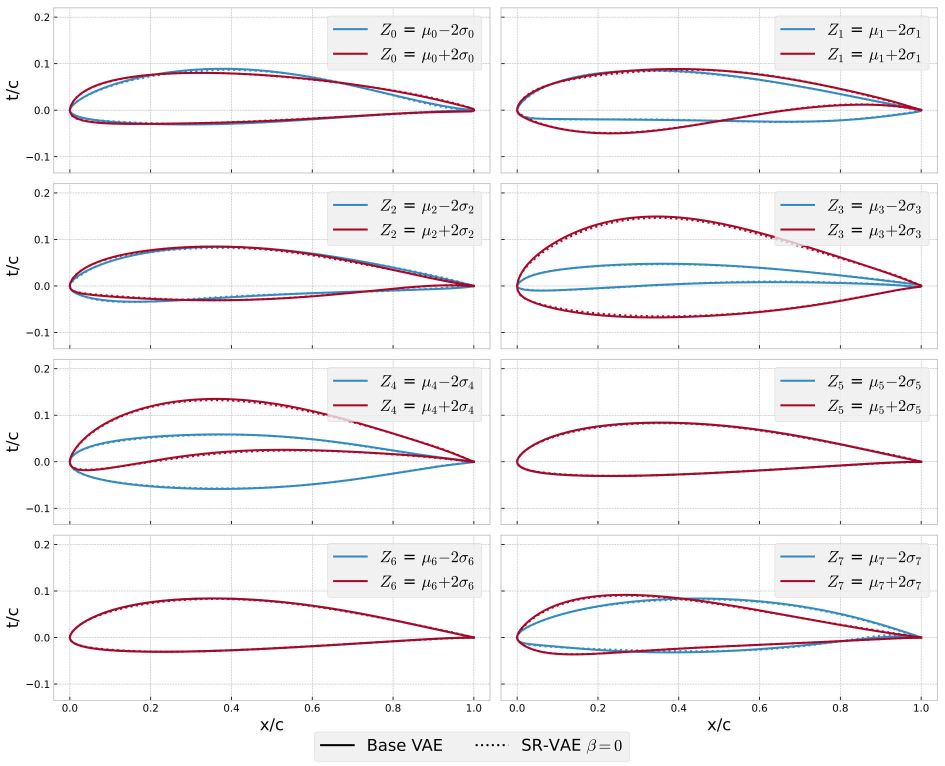


Figure 5.25: Latent effects plot of SR-VAE per batch where $\beta = 0$ with tuned Base VAE for reference

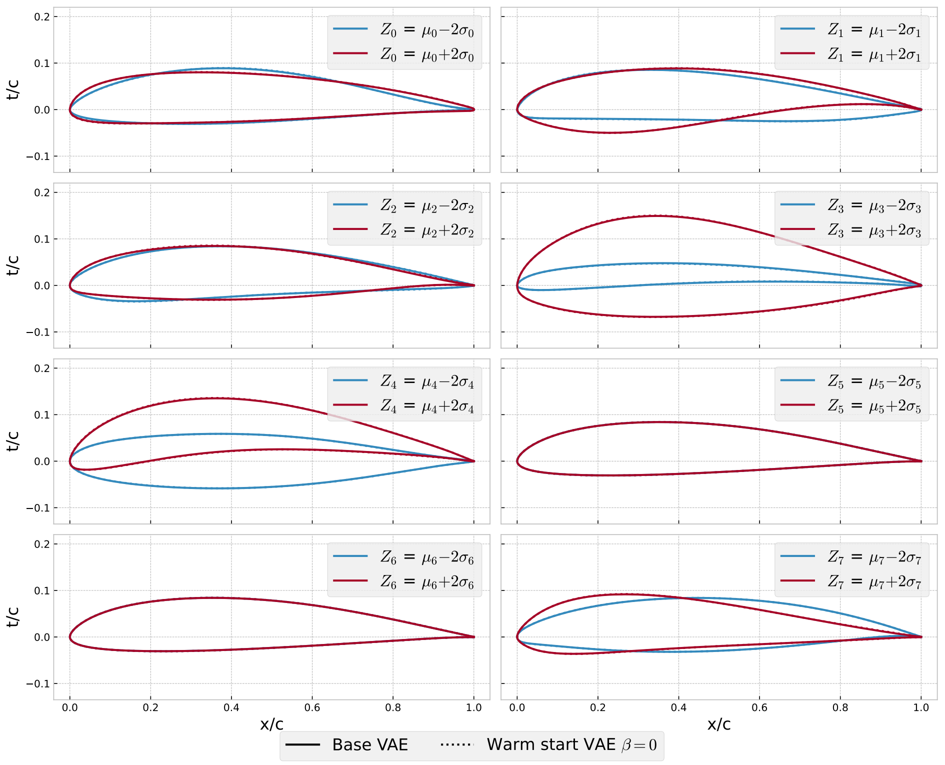
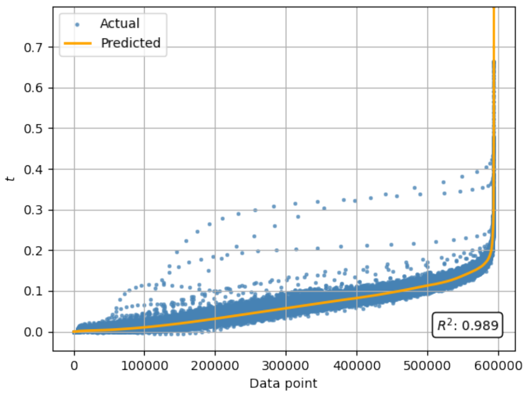


Figure 5.26: Latent effects plot of warm start VAE where $\beta = 0$ with tuned Base VAE for reference

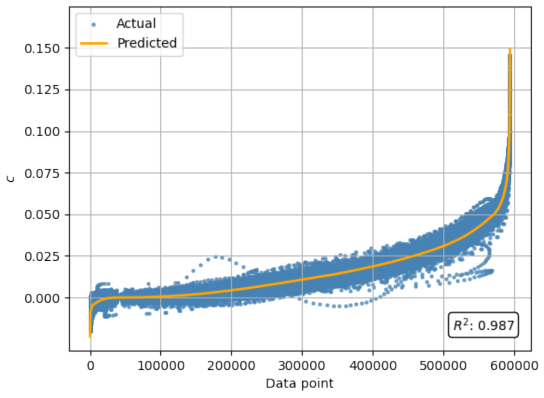
Table 5.9: Average MSE loss metrics of SR-VAE decoder, trained per batch with $\beta = 0$

Outliers UIUC	Outliers AFBench	UIUC	AFBench
$9.5714 \cdot 10^{-6}$	$4.8328 \cdot 10^{-5}$	$3.3964 \cdot 10^{-7}$	$4.0877 \cdot 10^{-6}$

While from the VAE performance it seems that the model is overfit, it could still be interesting to look at the SR performance since the SR validation loss is lower than that of all previous models. For this, the equations are analyzed and tested again against the test data in Figure 5.27.



(a) t-equation highest R^2



(b) c-equation highest R^2

Figure 5.27: Predicted values of parameterization equations from SR-VAE with SR trained per batch with $\beta = 0$, with the lowest overall MSE plotted against the actual values of adjusted test sets, sorted by ascending predicted values

$$\begin{aligned} f_{t,\beta=0} &= (Z_1 \cdot 0.0098 + Z_3 \cdot 0.0599 \\ &+ (X - 0.3757) \cdot (X \cdot Z_1 + Z_4 \cdot 0.3788 \\ &+ (Z_3 \cdot -0.6460 + Z_7 - 0.7787) \cdot \sin(3.1351^X)) \\ &\cdot -0.0711 + 0.2506)^{1.6238} \cdot \sin(X^{0.5334} \cdot 3.1223) \end{aligned} \tag{5.21}$$

$$\begin{aligned} f_{c,\beta=0} &= X \cdot (0.3226^X \cdot 0.1604 - 0.0517) \\ &\cdot (1.0772^{Z_4} \cdot (Z_4 + (X - 0.4617) \cdot (Z_1 + (X \\ &- 0.3485) \cdot (X \cdot (Z_0 + Z_1) + Z_2 + \sin(0.5459)^{Z_7}) \\ &- 0.4473 \cdot (Z_2 + Z_3 + Z_7))) + 1.1182^{Z_3}) \end{aligned} \tag{5.22}$$

Solely considering the R^2 scores, these parametric equations seem to be the best so far. However, closely examining and comparing the plots with previous SR-VAE models, it can be seen that although the spread of actual values is more concentrated around the predicted values, there are more outliers overall. For the camber distribution plot in Figure 5.27b, it can clearly be seen that a new region of outliers appeared near higher camber values. The differences in outliers for the thickness distribution plot in Figure 5.27a are more subtle, but a close look reveals that there are data points just outside the concentrated region near the predicted values. To confirm these observations, the SR performance can once again be quantified with the loss metrics as shown in Table 5.10.

Table 5.10: Average MSE loss metrics of equations from SR-VAE where SR is integrated in training per batch, with $\beta = 0$

Outliers UIUC	Outliers AFBench	UIUC	AFBench
$7.0308 \cdot 10^{-4}$	$4.2608 \cdot 10^{-4}$	$1.6182 \cdot 10^{-5}$	$1.4197 \cdot 10^{-5}$

As seen here, the SR-VAE with $\beta = 0$ outperforms all previous SR-VAE models with respect to overall UIUC and AFBench, which agrees with the high R^2-values. However, the equations exhibit the highest MSE values for the outliers so far. Since the equations perform well for both AFBench and UIUC test sets, the equations cannot exactly be called overfit. Nevertheless, parametric equations that perform well for both outliers and the overall test set are desired, which makes these particular set of equations less attractive. Furthermore, overall, a good SR-VAE model balances both VAE and SR performance.

5.4.5. γ=0

Finally, to confirm that the improved parametric equations are a consequence of SR integration into VAE training, and not the new way of conducting SR by doing multiple repeated searches with PySR's warm start, a new 'SR-VAE' is trained where SR is incorporated per batch but with $\gamma = 0$. In this way, there are still SR searches for every optimization step throughout training, but it will not have an influence on the loss function of the VAE. In case the improved parametric equations are purely achieved through this new way of conducting SR, this should lead to SR performance similar to the SR-VAE per batch. Since this is strictly speaking not an SR-VAE, an in depth analysis is not included. Nevertheless, this model is evalauted with the average loss metrics as shown in Table 5.11.

Table 5.11: Combined loss metrics of VAE decoder and SR equations from “SR-VAE” where $\gamma = 0$

	Outliers UIUC	**Outliers AFBench**	**UIUC**	**AFBench**
VAE	$7.7458 \cdot 10^{-6}$	$4.2404 \cdot 10^{-5}$	$4.1366 \cdot 10^{-7}$	$3.7495 \cdot 10^{-6}$
SR	$6.9050 \cdot 10^{-4}$	$6.3099 \cdot 10^{-4}$	$1.7768 \cdot 10^{-5}$	$1.8307 \cdot 10^{-5}$

Comparing this to previous SR-VAE models, the VAE decoder performance seems better than most and similar to the average MSE losses of the Retrained VAE. This is to be expected, since this ‘SR-VAE’ is equivalent to the Retrained VAE due to $\gamma = 0$, which enabled the optimizer to focus on minimizing VAE specific losses. The main point of interest here is the SR performance. The parametric equations from this ‘SR-VAE’ perform the worst out of all of the discussed SR-VAE models, with the exception of the UIUC outlier performance, where the SR-VAE with $\beta = 0$ performed even worse. This confirms that, while there does not seem to be a physical effect on the latent space, SR does have an influence on the VAE training, allowing for more accurate parametric equations.

However, comparing this to base SR metrics from Table 5.2, shows a slight improvement, except for UIUC outliers. This shows that part of the improved SR performance is from the new SR search strategy.

5.4.6. Analysis of final SR-VAE

With all SR-VAE model results presented, an optimal SR-VAE can be selected and further analyzed.

Model Selection

This section contains a qualitative evaluation of the best SR-VAE model of each variant, summarizing the findings of the previous SR-VAE sections. An overview of the presented decoder and SR equation loss metrics is shown in Table 5.12 and 5.13.

The decoder performance in Table 5.12 shows that the Retrained VAE, SR-VAE ($\gamma = 0$) and SR-VAE (fixed) yield the best average MSE losses. The former two can be purely attributed to retraining with warm start, explained in Subsection 5.2.3. While most SR-VAE models show degraded decoder performance in comparison, the SR-VAE (fixed) shows that SR integration is possible while retaining comparable decoder performance with even better performance for the UIUC averaged losses. The SR-VAE ($\beta = 0$) has the lowest overall UIUC loss, but has the worst performance in all other categories. As mentioned, this indicates that removing the regularization of D_{KL} leads to an overfitted model.

Table 5.13 highlights the performance of the equations derived in each SR-VAE training strategy. As expected, SR performance of the latent analysis and ‘SR-VAE’ ($\gamma = 0$) is significantly worse than that of any SR-integrated model. The SR-VAE ($\beta = 0$) variant has the lowest MSE losses for the overall datasets, but performs poorly on the outliers. Among the SR-VAE variants, the fixed SR-VAE demonstrates the lowest error in two of the four metrics, with above average performance for overall AFBench and SR-VAE ($\beta = 0$) slightly outperforming it in overall UIUC. These results reinforce the benefit of incorporating SR into the training process, especially when L_{SR} is actively minimized.

Table 5.12: Loss metrics of VAE decoder across different SR integration strategies, color shading reflects relative MSE

Model	**Outliers UIUC**	**Outliers AFBench**	**UIUC**	**AFBench**
Retrained VAE	$7.1020 \cdot 10^{-6}$	$4.1192 \cdot 10^{-5}$	$4.3407 \cdot 10^{-7}$	$3.7733 \cdot 10^{-6}$
SR-VAE (per epoch)	$7.7031 \cdot 10^{-6}$	$4.4010 \cdot 10^{-5}$	$4.4039 \cdot 10^{-7}$	$3.8146 \cdot 10^{-6}$
SR-VAE (per batch)	$8.8292 \cdot 10^{-6}$	$4.2552 \cdot 10^{-5}$	$4.5790 \cdot 10^{-7}$	$3.7921 \cdot 10^{-6}$
SR-VAE (fixed)	$5.6806 \cdot 10^{-6}$	$4.2636 \cdot 10^{-5}$	$3.8292 \cdot 10^{-7}$	$3.7826 \cdot 10^{-6}$
SR-VAE ($\beta = 0$)	$9.5714 \cdot 10^{-6}$	$4.8328 \cdot 10^{-5}$	$3.3964 \cdot 10^{-7}$	$4.0877 \cdot 10^{-6}$
‘SR-VAE’ ($\gamma = 0$)	$7.7458 \cdot 10^{-6}$	$4.2404 \cdot 10^{-5}$	$4.1366 \cdot 10^{-7}$	$3.7495 \cdot 10^{-6}$

🟩 Best 🟩 Better 🟩 Above Average 🟥 Below Average 🟥 Worse 🟥 Worst

Table 5.13: Loss metrics of parametric equations across different SR integration strategies, color shading reflects relative MSE

Model	Outliers UIUC	Outliers AFBench	UIUC	AFBench
Latent Analysis β-VAE	$1.1133 \cdot 10^{-3}$	$4.8452 \cdot 10^{-4}$	$2.6896 \cdot 10^{-5}$	$2.1726 \cdot 10^{-5}$
SR-VAE (per epoch)	$6.1631 \cdot 10^{-4}$	$3.0136 \cdot 10^{-4}$	$1.7263 \cdot 10^{-5}$	$1.5961 \cdot 10^{-5}$
SR-VAE (per batch)	$6.5806 \cdot 10^{-4}$	$2.9258 \cdot 10^{-4}$	$1.6976 \cdot 10^{-5}$	$1.4345 \cdot 10^{-5}$
SR-VAE (fixed)	$5.9367 \cdot 10^{-4}$	$2.3784 \cdot 10^{-4}$	$1.6312 \cdot 10^{-5}$	$1.4659 \cdot 10^{-5}$
SR-VAE ($\beta=0$)	$7.0308 \cdot 10^{-4}$	$4.2608 \cdot 10^{-4}$	$1.6182 \cdot 10^{-5}$	$1.4197 \cdot 10^{-5}$
'SR-VAE' ($\gamma=0$)	$6.9050 \cdot 10^{-4}$	$6.3099 \cdot 10^{-4}$	$1.7768 \cdot 10^{-5}$	$1.8307 \cdot 10^{-5}$

🟩 Best 🟩 Better 🟩 Above Average 🟥 Below Average 🟥 Worse 🟥 Worst

Looking at both SR and decoder performance, SR-VAE (fixed) is the optimal choice. For a more formal comparison, all loss metrics were standardized and averaged to produce a single scalar score for each model. This formal selection procedure is included in Appendix C, where both decoder and SR equation performance are jointly evaluated, incorporating many more trained SR-VAE models.

The rest of this section contains additional results of the fixed SR-VAE, on top of those presented in subsection 5.4.3, such that the model can be better compared to the results of the latent analysis. Similarly to before, the additional results will mainly present results from UIUC. Additional results using the AFBench dataset are included in Appendix D. The optimal SR equations have been presented for the SR-VAE (fixed), and can be found in Equations (5.18) and (5.19).

Latent effects plot

Similar to the latent analysis, the latent effects of the improved VAE and the parametric SR equations are plotted in Figure 5.28 to investigate the similarities between the decoder and the equations.

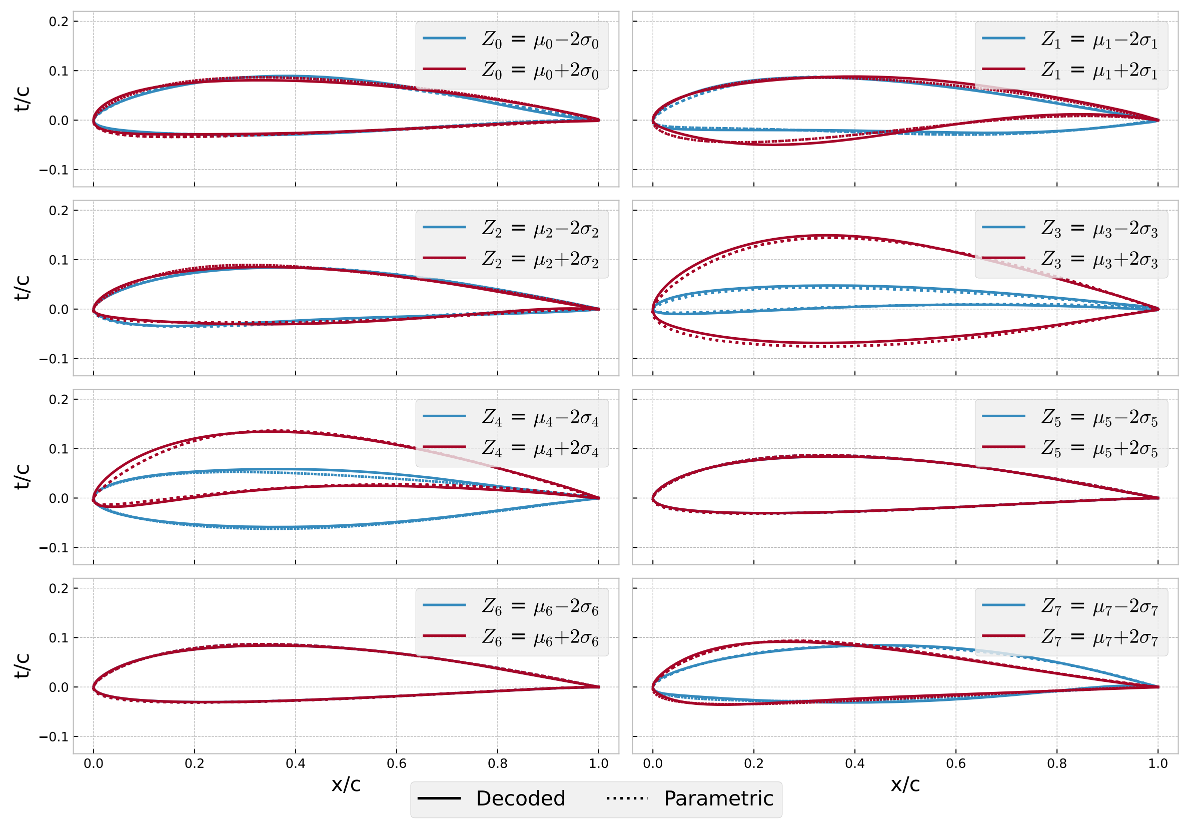


Figure 5.28: Effect of latent parameters on the generated shape in a range of $Z_i = \mu_i \pm 2\sigma_i$, while the rest are at their mean, for both decoded and parametric airfoils as a result of the final SR-VAE

Compared directly to Figure 5.7, the figures show that there is no visible difference in latent effects. The improved overall losses of the parametric equations are not reflected in the behavior of the latent parameters. This shows that the role of the latent variables in the equation is consistent.

Validation plots worst decoded airfoils

The new SR-VAE model can be validated by considering the reconstructed airfoils with the highest MSE losses, for both the decoder and the parametric equations. The reconstructed airfoils with the highest decoder loss are plotted in Figure 5.29.

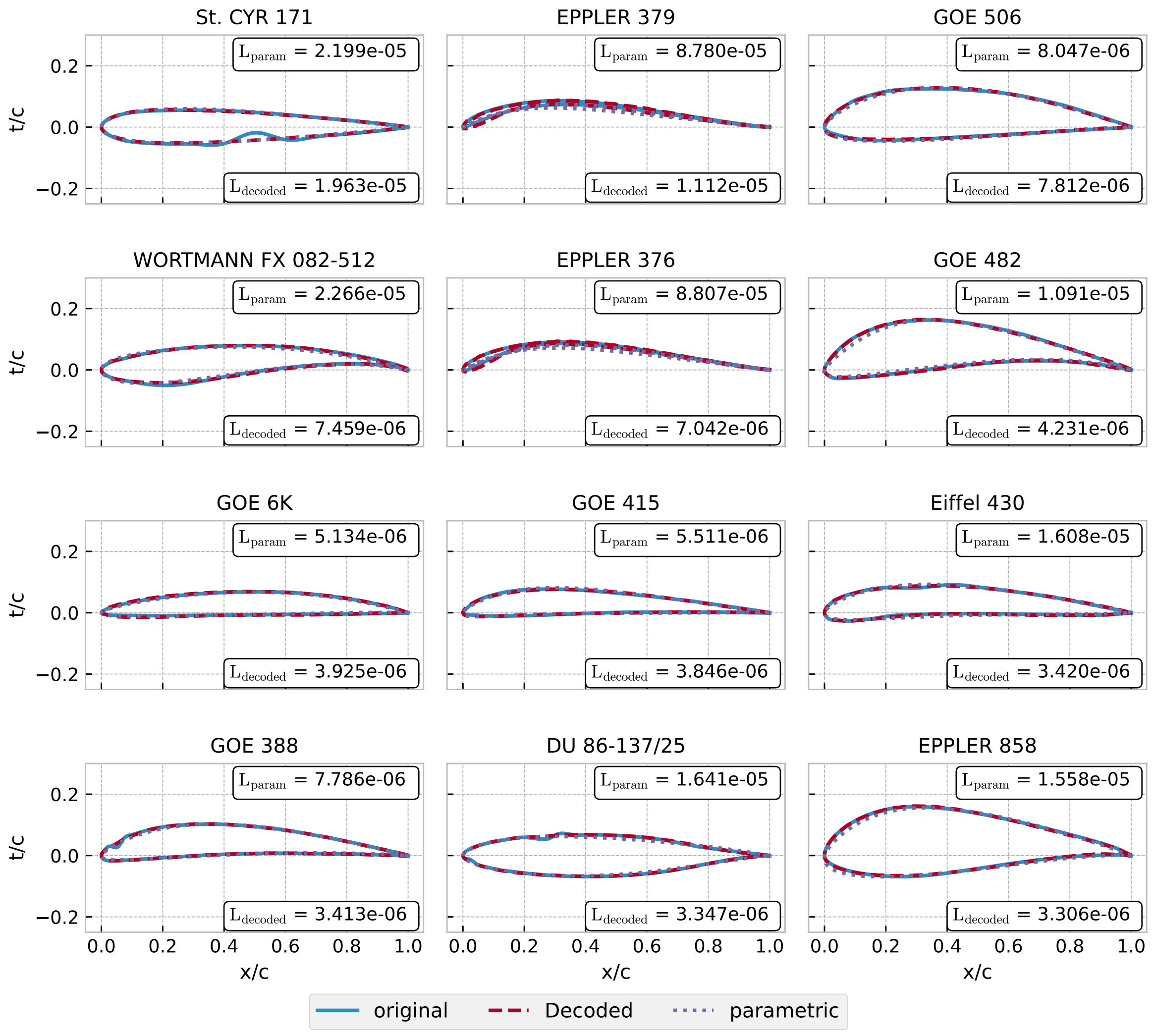


Figure 5.29: Validation plots of final SR-VAE showing parametric and decoded reconstruction of 12 airfoils with highest decoder loss, the losses marked at the corners of each plot is the MSE

While previous analyses have shown that the SR equations perform worse than the decoder, in this particular plot the parametric equations seem to perform remarkably similar. For most airfoils, the decoder loss L_{decod} and the parametric loss L_{param} are of the same order of magnitude, with the St. CYR 171, GOE 506 and EPPLER 376 even being almost equivalent.

Considering the decoder performance and comparing it with the original validation plot by Swannet et al. [4] included in Figure A.2, shows that overall the highest reconstruction losses are much lower than before. This outlier performance difference was already confirmed in Table 5.1 and Table 5.8. The airfoils with an extremely high thickness are not among the worst reconstructed airfoils anymore, while the very thin EPPLER airfoils seem to be added. For a direct comparison with the original VAE, the same airfoils are selected from Figure A.2 and plotted in Figure 5.30.

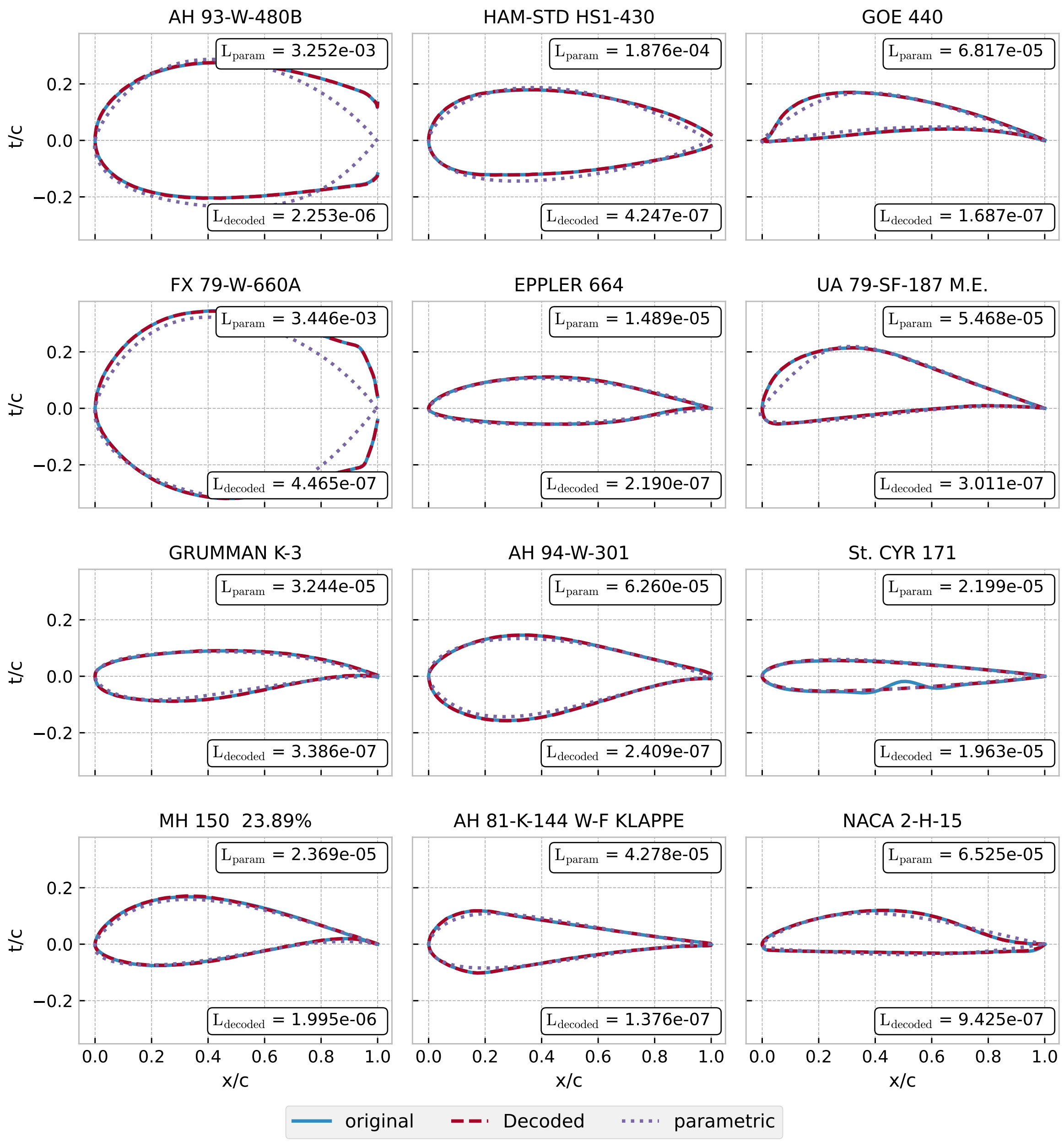


Figure 5.30: Validation plots of final SR-VAE showing parametric and decoded reconstruction of 12 airfoils that showed the highest decoder loss for Swannet et al. [4], the losses marked at the corners of each plot is the MSE

By comparing the parametric losses here with the decoded losses from Figure A.2, it can be seen that the new SR equations perform similarly to the original decoder for its outliers, even outperforming it for the EPPLER 664. The exception to this are thick airfoils like the AH 93-W-480B or the FX 79-W-660A. This shows that the parametric equations are not as sensitive to some of the outlier airfoils as the original VAE. More interestingly, the SR-VAE decoder performs far better than the original VAE, with all the MSE values even being two orders of magnitude smaller than the original. The only exception is the St. CYR 171, where the MSE values are nearly equivalent.

Validation plots worst parametric airfoils

Finally, to properly validate the parametric equations, the airfoils with the worst parametric losses can be plotted, similar to what was done for Figure 5.8. The resulting figure is shown in Figure 5.31.

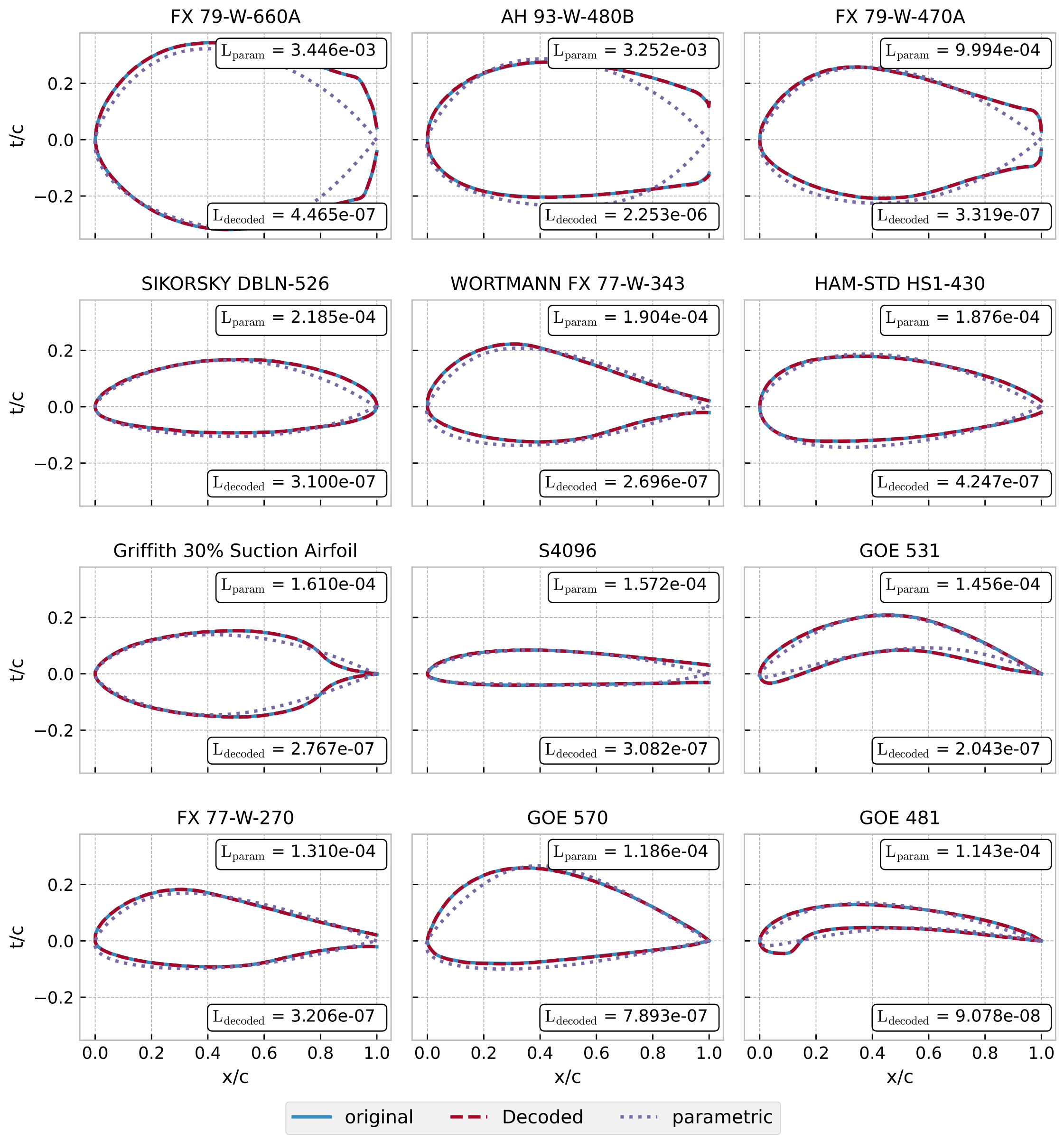


Figure 5.31: Validation plots of final SR-VAE showing parametric and decoded reconstruction of 12 airfoils with highest parametric loss, the loss marked at corner of each plot is the MSE

Once again comparing it with the MSE losses of the latent analysis equations shown in Figure 5.8, shows that the parametric losses are significantly lower, with most even being more than twice as low. This agrees with the earlier identified loss metrics from Table 5.1 and Table 5.8. Contrary to before, the max thickness of bigger airfoils like the FX 79-W-660A, AH 93-W-480B and FX 79-W-470A are not underestimated anymore. While from visual inspection the parametric reconstructions seemed to have improved a lot, the limitations identified before during the latent analysis remain. The unconventional airfoils with extreme thickness and/or an unconventional trailing edge still show poor reconstructions. Furthermore, the parametric reconstructions are still significantly worse compared to the decoder.

The most likely reason for the improved parametric equations is that actual airfoil properties have been used. The SR algorithm no longer depends on the decoder. While before the SR model represented a surrogate of a surrogate, it now directly tries to fit physical airfoils with no possible distortions from the decoder. Considering all the SR-VAE models that have been trained and the tens of thousands of SR run, it seems that the capability limit of SR has been reached in fitting parameterization equations with the current tuned PySR settings.

5.5. Applicability to optimization

Now that SR has been used to interpret the original VAE model and that novel SR-VAEs have been trained in attempt to improve upon the original, the optimization performance of the VAEs with the accompanying SR equations can be assessed and compared with other parameterization methods. For this, first the inverse design capability is tested to confirm if it can accurately reconstruct all the airfoils in the UIUC dataset. This is then followed by applications to two optimization cases, one real-life case with a known solution for validation, and another to test the capability to explore the design space. The performance of the original VAE, the SR-VAE and SR equations will be tested. To also compare it with a more traditional parameterization, CST is tested as well, since that is widely regarded as the most popular and versatile airfoil parameterization method, especially in optimization contexts.

5.5.1. Inverse design

The test for inverse design capability draws from results by Kang et al. [7], where parsimony and flexibility are assessed through a cumulative distribution plot of the MSE of reconstructed airfoils. However, the methodology for obtaining these plots is vastly different.

The MSE values from the VAE and the accompanying SR equations are obtained in a similar fashion to the calculation of reconstruction loss L_{recon} and symbolic regression loss L_{SR}. For the former, the target airfoils are encoded and decoded, subsequently calculating the reconstruction error between the target airfoil and the reconstructed airfoil. For the latter, the target airfoils are encoded to obtain the latent variables, which are then fed to the SR equations to obtain the thickness and camber distribution. Subsequently, these are used to reconstruct the target airfoil and calculate the MSE. For CST parameterization, the AeroSandBox library [82, 83] is used. This library contains functions that can approximate CST coefficients from airfoil coordinates, where the user can set the amount of weights per side for the CST parameterization. This is set to 6 to be consistent with the SR-VAEs and SR equations. This is then converted back to coordinates to calculate the MSE of the reconstructed airfoil.

Contrary to Kang et al., who only used a subset of the UIUC dataset, the complete UIUC dataset is used to construct the cumulative distribution plot. The resulting plot is shown in Figure 5.32.

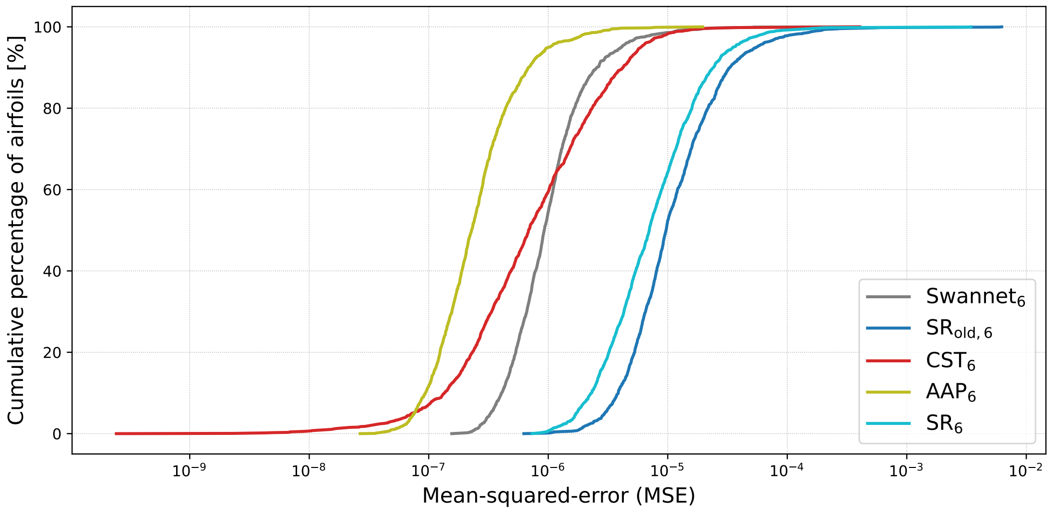


Figure 5.32: Cumulative distribution plot of all airfoil MSE losses from the UIUC dataset, for SR-VAE model and its equations, the original model from Swannet et al., the SR equations from the latent analysis and CST parameterization

Here, the x axis represents the MSE losses and the y-axis the cumulative percentage of airfoils that can be fit within this error bound. SR-VAE can represent all airfoils within an MSE error bound of $2 \cdot 10^{-5}$, while CST parameterization can do this within a error bound of $3 \cdot 10^{-4}$. These results confirm that the SR-VAE substantially improved compared to the original VAE. The improvement in the SR equation is also apparent, although subtle. Furthermore, SR-VAE can consistently represent airfoils within a smaller error bound than the traditional CST parameterization, which was not true for the original VAE.

5.5.2. Daedalus case

To test the performance in an optimization, a real-life design case is chosen where an airfoil was expertly designed for the MIT Daedalus, a human-powered aircraft [84]. The final DAE-11 is a second generation airfoil that is obtained using a traditional inverse-design method and direct geometry manipulation techniques, where the performance of the predecessor airfoils has been verified in flight tests [85]. This optimization problem is commonly used in literature as a benchmark for parameterization methods and aerodynamic solvers [86–88]. For reference, a drawing of the aircraft along with the DAE-11 airfoil is shown in Figure 5.33.

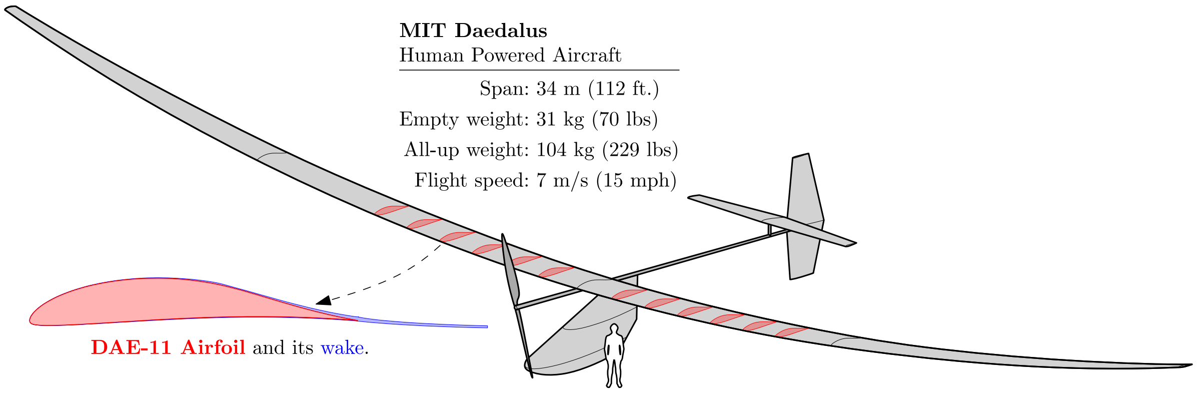

MIT Daedalus
Human Powered Aircraft
Span: 34 m (112 ft.)
Empty weight: 31 kg (70 lbs)
All-up weight: 104 kg (229 lbs)
Flight speed: 7 m/s (15 mph)
DAE-11 Airfoil and its wake.

Figure 5.33: Drawing of the MIT Daedalus human-powered aircraft with its centerline DAE-11 airfoil [87]

Problem definition

Drela's optimization problem [85] is shown below, where following the implementation by Sharpe [87], the equality constraints have been replaced by inequality constraints to reflect the design goals of the original engineering problem and trailing edge angle has been adjusted to that of the DAE-11 airfoil:

$$\begin{aligned}\text{Minimize:} \quad F(\{S_i\},\alpha) \equiv {} & \frac{5}{45}C_D|_{C_L=0.8} + \frac{6}{45}C_D|_{C_L=1.0} + \frac{7}{45}C_D|_{C_L=1.2} \\ & + \frac{8}{45}C_D|_{C_L=1.4} + \frac{9}{45}C_D|_{C_L=1.5} + \frac{10}{45}C_D|_{C_L=1.6}\end{aligned} \tag{5.23}$$

$$\text{Subject to:} \quad \begin{aligned} & C_M \geq -0.133 \quad (t/c)_{0.33c} \geq 0.128 \\ & \theta_{\text{TE}} \geq 6.25° \quad (t/c)_{0.90c} \geq 0.014 \\ & C_L = [0.8,\ 1.0,\ 1.2,\ 1.4,\ 1.5,\ 1.6] \end{aligned} \tag{5.24}$$

$$\text{Design variables:} \quad S_i,\ \alpha \tag{5.25}$$

As can be seen here, a 6-point weighted optimization problem is defined, where drag C_D is optimized at 6 different fixed lift coefficients C_L with both the shape parameters S_i and angle of attack α as design variables, so $F(\{S_i\},\alpha) = \bar{C}_D$. The lift constraint essentially eliminates α as a design variable [85]. For aerodynamics, NeuralFoil is used, with the following flow conditions:

$$Re = 500000 \cdot (\tfrac{C_L}{1.25})^{-0.5} \qquad M = 0.03 \tag{5.26}$$

The initial guess of the shape parameters are set to the NACA 0012 and for α it is defined as follows:

$$\alpha = \frac{C_L}{2\pi} \tag{5.27}$$

The bounds of this design variable are slightly tightened, to shift the focus of the optimization to the parameterization. Instead of the original bounds of $[-5, 18]$, the bounds are set to $[0, 10]$ instead.

This exact optimization problem has been used by Sharpe to validate NeuralFoil with CST [87]. This ensures that any difference in the results are specific to the change in parameterization method. For reference, an adjusted version of this problem without α as a design variable is included in Appendix E.

Implementation

The original code by Sharpe uses IPOPT with `opti.solve()` and 16 CST coefficients for airfoil parameterization. This optimizer relies on gradients and Hessians, which means that it requires symbolic expressions instead of numpy arrays. While this would work for the symbolic SR equations, this would be cumbersome to implement for the VAE decoder. Therefore, instead of using IPOPT, the scipy optimization toolbox with `scipy.minimize()` is used instead. The default SLSQP optimization method is used, with a function tolerance of $1 \cdot 10^{-7}$ and a maximum number of iterations of 500.

For the VAE and SR equations, the NACA 0012 from the UIUC dataset is encoded to define an initial guess for the latent parameters Z. The corresponding bounds are defined such that the design space does not restrict the converged result. Since the prior is Gaussian, the starting point of the bounds, before relaxing them if required, is set to $[-2, 2]$, which corresponds to $\mu = 0$ and $2\sigma = 2$. For the CST case, the native NACA 0012 coordinate file from AeroSandBox is used, with the same custom bounds as used by Sharpe [87], where upper surface coefficient bounds are set to $[-0.5, 0.25]$, lower surface coefficient bounds to $[-0.25, 0.5]$ and the leading edge coefficient to $[-1, 1]$.

The original validation case from NeuralFoil used 16 CST coefficients, while the new parameterizations from SR equations and SR-VAE model only have 6 parameters. Therefore, the CST parameterization is established as a baseline with both 16 and 6 coefficients.

Across the three different optimization cases, all optimization options are kept identical if not mentioned otherwise. This ensures a fair comparison in the final shapes and run time. As an additional check, the DAE-11 airfoil is established as a reference airfoil to test how much the drag has improved or worsened.

Results

First, as a baseline for the optimization results, the traditional CST parameterization is applied to the subsonic Daedalus optimization case. The result is shown in Figure 5.34.

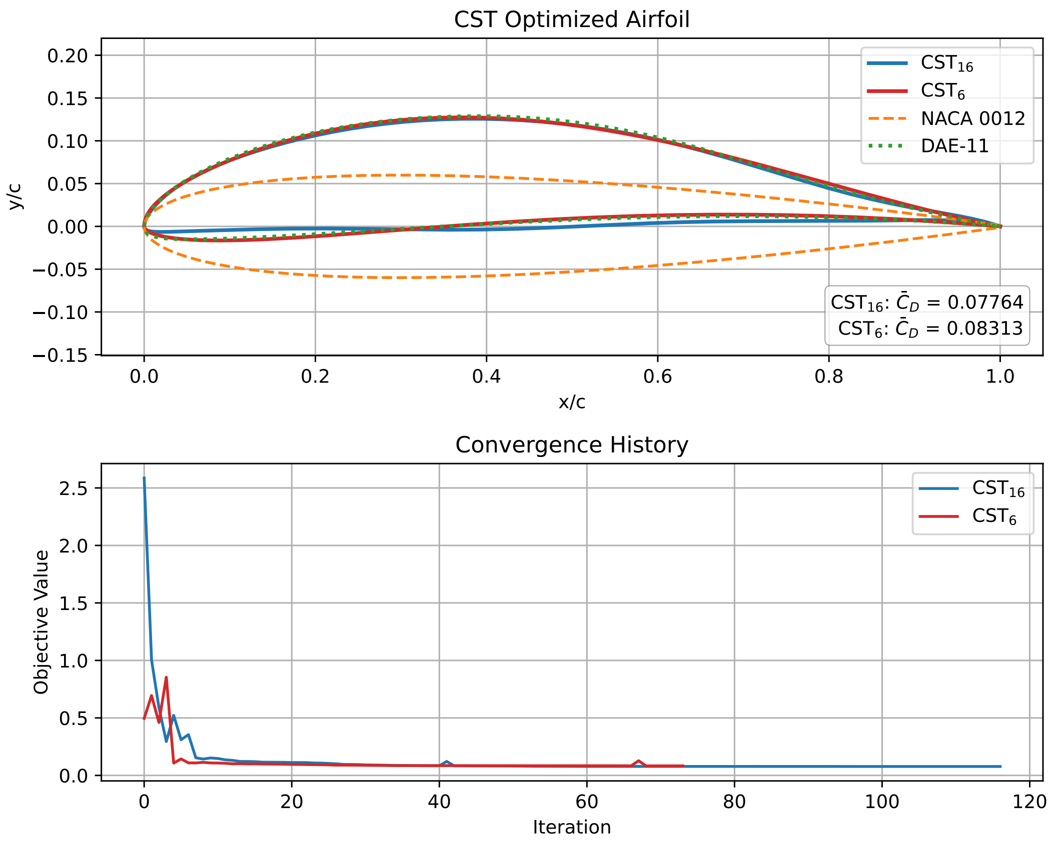


Figure 5.34: Airfoil shape from CST_6 and CST_{16} airfoil shape optimization for the subsonic Daedalus optimization case, including the mean drag objective value $\bar{C}_D$

The CST$_{16}$ optimization had a runtime of 30.51 seconds and converged after 117 iterations, corresponding to a rate of 3.835 iterations per second. It achieved a mean drag coefficient of 0.07764, which is 3.376% lower than the DAE-11 baseline. The strength of the CST parameterization is evident in the airfoil shape, particularly near the trailing edge, where subtle perturbations are visible.

Using 6 coefficients with CST$_6$ resulted in a runtime of 12.42 seconds and convergence after 74 iterations, or 5.96 iterations per second. This optimization appeared more unstable at the start, with overshoots, but converged faster than CST$_{16}$. The final shape is smoother and more similar to DAE-11, although it has a higher $\bar{C}_D$ of 0.08313, representing a 3.19% increase over DAE-11 and only a 0.126% difference from the CST$_6$-parameterized DAE-11.

The exact same optimization setup was used to test the SR equations, as shown in Figure 5.35.

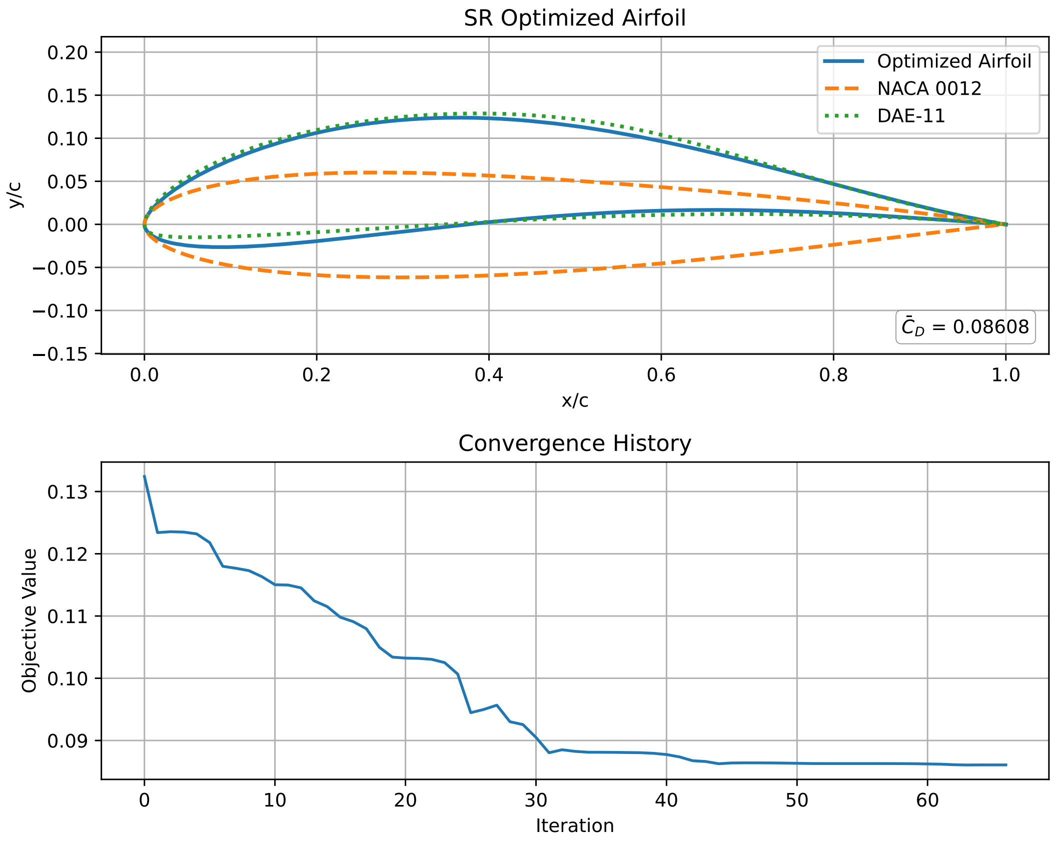


Figure 5.35: Airfoil shape from SR airfoil shape optimization of the subsonic Daedalus optimization case, including the mean drag objective value $\bar{C}_D$

This optimization had a runtime of 10.65 seconds and converged after 67 iterations, or 6.29 iterations per second, which is faster than both CST optimizations. This can be attributed to the compact design space and reduced number of design variables from the use of parametric SR equations. The convergence was also more stable, with no visible overshoots, although it appeared to reach the optimal value slightly later than CST$_6$.

The SR-optimized airfoil has a different optimum, with an objective of 0.08608. Similar to CST$_6$, it is smoother and lacks the nuanced shape perturbations seen in CST$_{16}$, demonstrating the reduced flexibility of the SR equations. This is further illustrated in the parametric reconstruction of the DAE-11 airfoil shown in Figure 5.36.

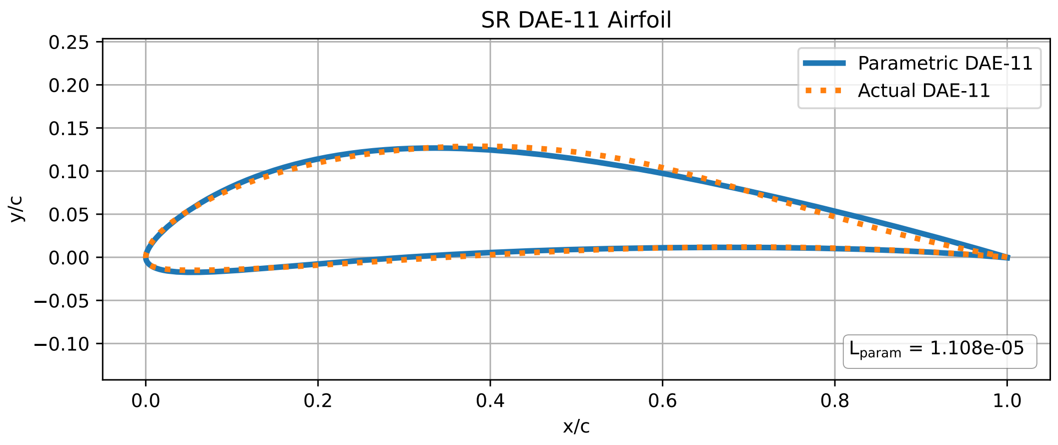


Figure 5.36: Parametric reconstruction of the DAE-11 airfoil using final SR equations from the SR-VAE

The reconstructed shape is close, but not identical. Although the optimized $\bar{C}_D$ in Figure 5.35 is 6.85% higher than the original DAE-11, it is 2.93% lower than the mean drag of the SR-parameterized DAE-11 shown in Figure 5.36. Clearly, more efficient parametric equations come at the cost of flexibility. Nevertheless, it converges rapidly and shows promise for preliminary airfoil design. However, for this specific case, CST parameterizations perform better.

The SR-VAE model should offer more accuracy and flexibility than both the SR equations and CST$_6$, while maintaining a similar level of parsimony. The result is shown in Figure 5.37.

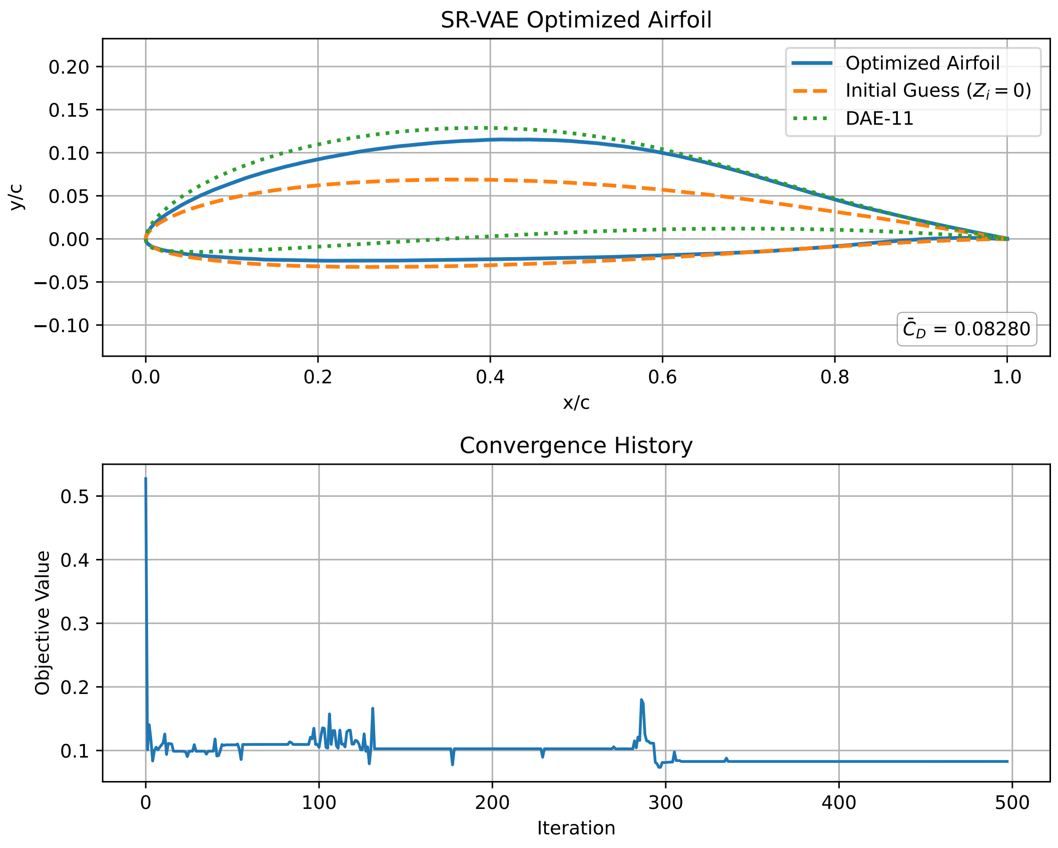


Figure 5.37: Airfoil shape from SR-VAE airfoil shape optimization of the subsonic Daedalus optimization case, including the mean drag objective value $\bar{C}_D$

Here, NACA 0012 could not be used as the initial guess due to a positive directional derivative in the line search, which prevented progress without violating constraints. Instead, the latent variables Z_i were initialized to 0, roughly corresponding to the mean of the training set. Additionally, a regularization constraint was added to enforce a NeuralFoil "analysis confidence" of at least 0.90, ensuring reliable aerodynamic predictions. The result of the original case without changes is shown in Figure E.5.

Despite these adjustments, the optimization did not fully converge, terminating after the maximum 500 iterations. It took 152.94 seconds, or 3.27 iterations per second, which is slower than SR and CST$_6$, but comparable to CST$_{16}$. This slowdown is likely due to the decoder's computational overhead. The optimization initially finds a promising shape and stabilizes after about 310 iterations. However, it fails to meet all constraints, specifically the C_L equality and trailing edge angle θ_{TE}, as shown in Figure 5.38.

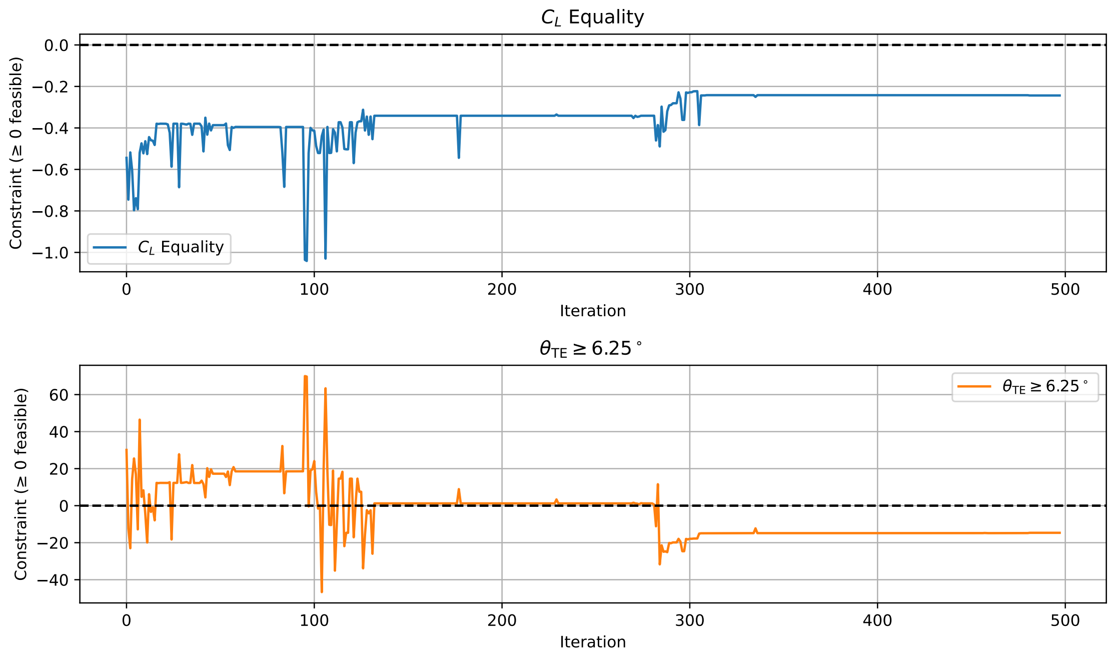


Figure 5.38: Constraint violations during SR-VAE airfoil shape optimization, showing worst-case violation at each iteration

As seen here, at no point was the C_L constraint satisfied, which means that the optimization did not achieve the target lift values at all points. The trailing edge constraint was met initially but eventually violated in favor of a better objective. This is visible in the final shape, where it clearly has a different trailing edge angle than the DAE-11. The resulting $\bar{C}_D$ is 0.08280, outperforming the SR and CST$_6$ optimizations, and only 2.78% worse than the expertly designed DAE-11. Compared to the SR airfoil in Figure 5.35, the SR-VAE shape shows greater nuance, which is confirmed in Figure 5.39.

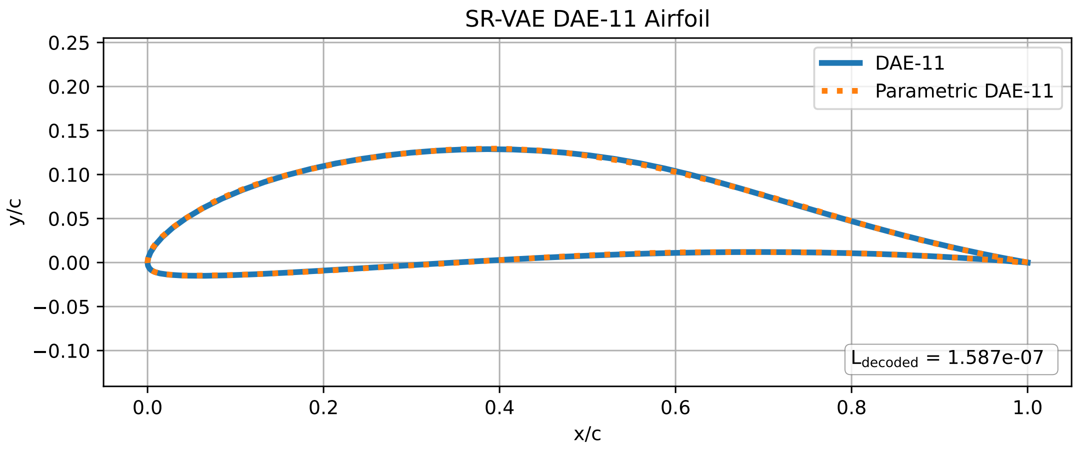


Figure 5.39: SR-VAE decoder reconstruction of the DAE-11 airfoil

However, the SR-VAE decoder's sensitivity to the initial guess and reliance on a NeuralFoil constraint make it less robust for aerodynamic optimization. This likely stems from its vulnerability to outlier cases far from the training set mean. The decoder appears too unstable for reliable use in shape optimization.

A summary of the optimization results for the subsonic Daedalus airfoil optimization case using the different parameterizations is shown in Table 5.14.

Table 5.14: Summary of optimization results for the subsonic Daedalus case using different parameterizations

Method	**Runtime [s]**	**Iters**	**Iters/s**	$\bar{C}_D$	**%** $\Delta\bar{C}_D$ **vs DAE-11**
CST$_6$	12.42	74	5.96	0.08313	$+3.19\%$
CST$_{16}$	30.51	117	3.84	0.07764	-3.38%
SR Equations	10.65	67	6.29	0.08608	$+6.85\%$
SR-VAE	152.94	500*	3.27	0.08280	$+2.78\%$

*Optimization did not converge; terminated at max iterations.

Overall, these results demonstrate a trade-off between model complexity, flexibility, and aerodynamic performance. The CST$_{16}$ parameterization achieves the best mean drag reduction relative to the expert-designed DAE-11, reducing drag by 3.38% at a moderate computational cost. In contrast, the more compact CST$_6$ and SR equations offer faster convergence and shorter runtimes but at the expense of increased drag, highlighting their value for rapid preliminary design.

The SR-VAE decoder, while capable of generating more nuanced shapes than the SR equations and outperforming CST$_6$ in terms of drag, suffers from slower runtimes and did not fully converge within the maximum iteration limit. Its sensitivity to the initial guess and reliance on solver constraints indicate challenges with stability and constraint satisfaction, particularly for shapes deviating far from the UIUC mean. These limitations currently make it less robust for aerodynamic optimization compared to CST$_{16}$.

5.5.3. NACA 0012 Optimization

The DAE-11 optimization primarily tested the ability of the parameterizations to converge to a known solution. To properly test the design space of the parameterizations, a new optimization case is selected with no set solution, that aims to optimize the NACA 0012 shape. Two variants of this optimization case are considered, a constrained case for more physically feasible shapes and an unconstrained case to investigate the flexibility of each parameterization by pushing it to an extreme. This specific optimization case draws from work by Kang et al. and Chen et al. [7, 31]. The optimization problems for both cases are defined in Table 5.15.

Table 5.15: Problem definition of constrained and unconstrained NACA 0012 optimization [7]

	Unconstrained	Constrained
Maximize	C_L/C_D	C_L/C_D
Subject to	$t_{\text{min}} > 0$	$t_{\text{min}} > 0$
		$c_{\text{max}} < 0.03$
		$0.1 < t_{\text{max}} < 0.12$
		$R_{\text{LE}} < 0.005$
Design variables	S_i	S_i

As can be seen here, the objective is a single-point maximization of the lift-to-drag ratio, or C_L/C_D, with parameterization S_i as the only design variable. The flow conditions are defined as follows [7, 31]:

$$Re = 1.8 \cdot 10^6 \qquad M = 0.01 \qquad \alpha = 0^\circ \tag{5.28}$$

The rest of the NACA 0012 optimization and its implementation is mostly identical to the DAE-11 case. The Scipy optimization toolbox is used with the SLSQP optimization method, with a function tolerance of $1 \cdot 10^{-7}$ and maximum iterations of 500. The NACA 0012 is once again parameterized with the appropriate method to define the initial guess, with the same corresponding bounds depending on the parameterization method. The CST bounds are relaxed as well if required. NeuralFoil is used as the aerodynamic solver, this time with a constant angle of attack under the defined flow conditions.

The reduced complexity of the objective function should lead to more variability in the optimized shapes. Furthermore, the absence of aerodynamic constraints and angle of attack as design variable means that the optimization should be less affected by possible limitations of NeuralFoil.

Constrained

First, the CST parameterization is used for the optimization. The result, once again using 16 and 6 CST coefficients, is shown in Figure 5.40.

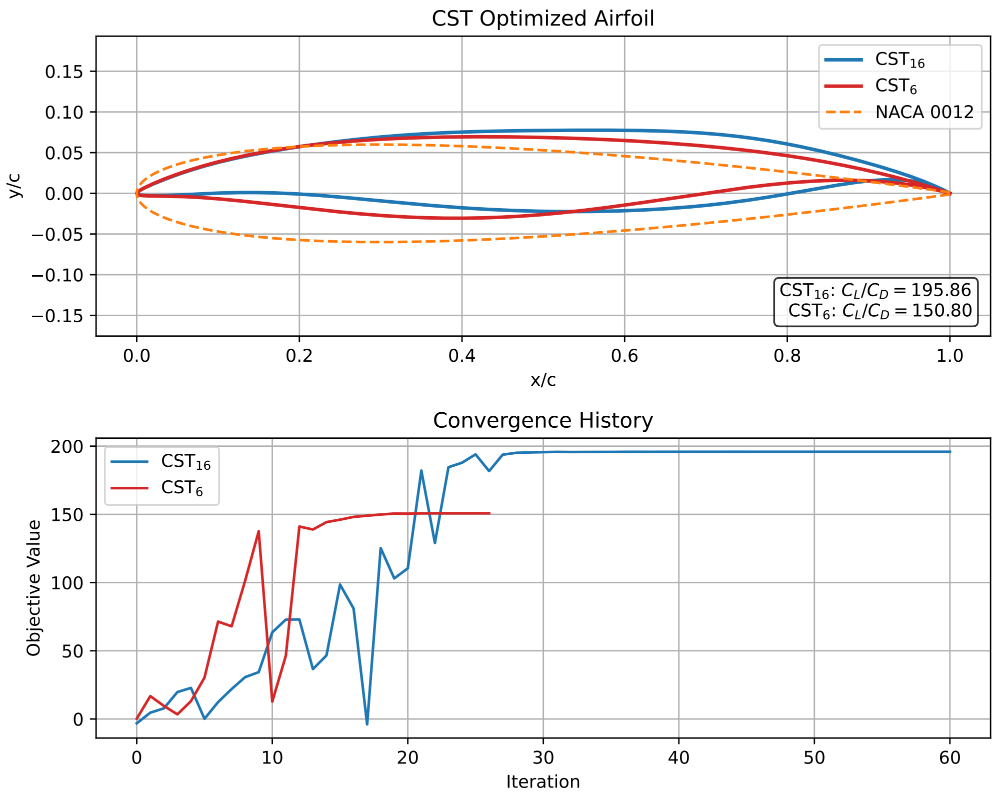


Figure 5.40: Airfoil shape and convergence history of CST$_6$ and CST$_{16}$ airfoil shape optimization for the constrained NACA 0012 optimization case, including lift over drag objective value C_L/C_D

As anticipated, CST$_{16}$ converged to an optimized airfoil shape with a noticeably higher C_L/C_D of 195.86, compared to the objective value of 150.80 for CST$_6$. The increased flexibility from the higher number of CST coefficients allowed the optimizer to find a more optimal shape. Interestingly, both shapes show a similar degree of detail, each exhibiting nuanced perturbations in their optimized forms.

The CST$_{16}$ optimization converged in 16.261 seconds after 61 iterations, resulting in a convergence speed of 3.751 iterations per second. In contrast, CST$_6$ took only 6.01 seconds and 27 iterations, or 4.493 iterations per second. The difference in convergence speed is less pronounced than in the Daedalus case, suggesting that the number of design variables has a less pronounced impact on this relatively less constrained problem. This is also reflected in the convergence behavior, where CST$_6$ converges with fewer overshoots than CST$_{16}$.

The results of applying the SR equations and SR-VAE decoder to the same optimization case are shown in Figure 5.41 and 5.42.

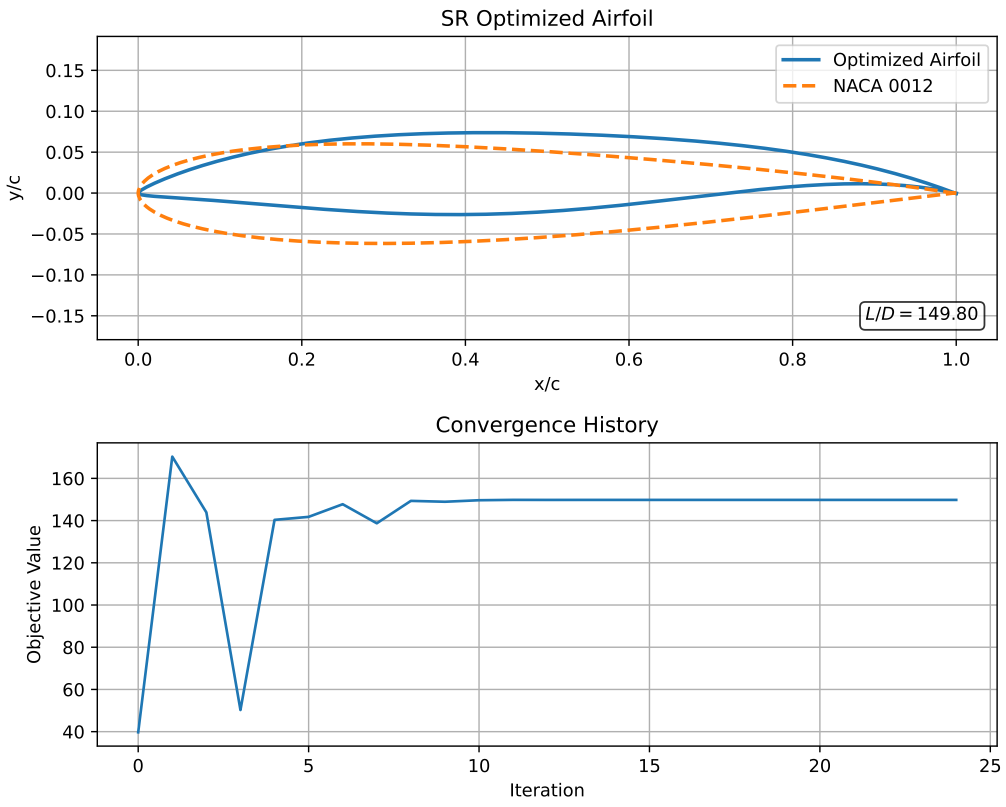


Figure 5.41: Airfoil shape and convergence history of SR airfoil shape optimization for the constrained NACA 0012 optimization case, including the lift over drag objective value C_L/C_D

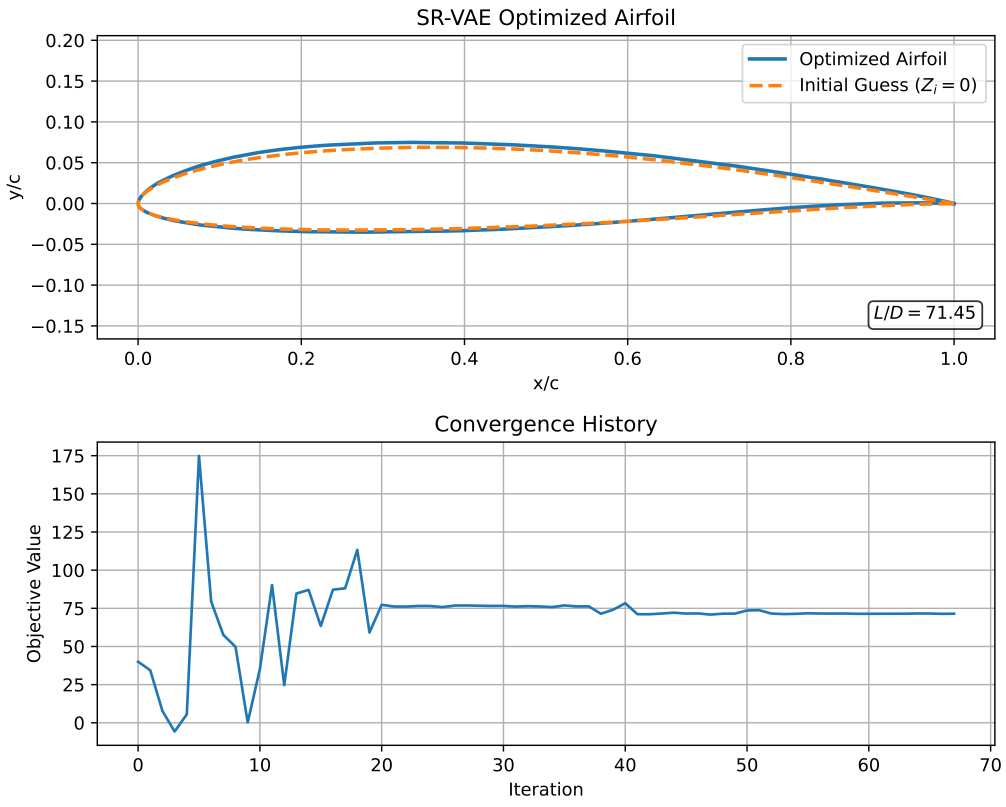


Figure 5.42: Airfoil shape and convergence history of VAE airfoil shape optimization for the constrained NACA 0012 optimization case, including the lift over drag objective value C_L/C_D

Figure 5.41 shows that the optimal shape from the SR equations is nearly identical to that of the CST$_6$ parameterization, albeit at a slightly lower C_L/C_D. This is consistent with Figure 5.32, which indicated that CST$_6$ can reconstruct airfoils with higher fidelity than SR$_6$. The optimization result where both parameterizations are combined can be found in Figure E.6. The convergence behavior appears slightly less stable than that of CST$_6$, which is reflected in a higher runtime of 3.111 seconds over 25 iterations. However, the convergence speed remains consistently faster at 8.074 iterations per second.

The optimization with the SR-VAE decoder is shown in Figure 5.42, and once again required a different initial guess and an additional confidence constraint for NeuralFoil. The case without this constraint is included in Figure E.7. Despite these changes, the optimization failed to converge because it could not identify a feasible direction to improve the objective value while satisfying all constraints. Unlike the Daedalus case, the final shape does not violate any constraints, but the airfoil barely changed with an objective value of only 71.45. The optimization terminated after 72 iterations in 16.603 seconds, or 4.337 iterations per second, which is once again slower than CST$_6$ and SR equations, but faster than CST$_{16}$. This reinforces the observation that using a decoder during optimization is a computationally intensive approach and significantly restricts the optimizer's ability to explore the design space.

A summary of the NACA 0012 optimization for the tested parameterizations is shown in Table 5.16.

Table 5.16: Summary of the constrained NACA 0012 airfoil optimization results using different parameterizations

Method	Runtime [s]	Iters	Iters/s	C_L/C_D
CST$_6$	6.01	27	4.493	150.80
CST$_{16}$	16.261	61	3.751	195.86
SR Equations	3.11	25	8.074	149.80
SR-VAE	16.60	72*	4.337	71.45

*Optimization did not converge; terminated due to positive directional derivative for linesearch.

This table clearly shows that the CST$_{16}$ parameterization yields the best aerodynamic results due to its increased fine geometric control, but that the SR equations are far quicker while retaining similar aerodynamic performance to CST$_6$. Unfortunately, the SR-VAE decoder lacks in all aspects since it restricts the optimizer from effectively searching the design space.

Unconstrained

The unconstrained NACA 0012 optimization case pushes the parameterizations to their extremes, since the design space is nearly unrestricted. The lack of constraints should allow the decoder to converge.

Figure 5.43 shows the results of the unconstrained optimization using CST parameterization with 6 and 16 coefficients. Remarkably, both parameterizations required the additional NeuralFoil confidence constraint for a feasible optimization result; the result without the use of this constraint is included in Figure E.8. The unrestricted design space in combination with the very flexible CST parameterization, likely resulted in very unrealistic airfoil shapes for which the aerodynamic solver fails. With this constraint, once more the CST$_{16}$ parameterization outperforms CST$_6$, showing a higher C_L/C_D of 314.14 compared to 284.37. Both shapes are very thin, with the max thickness location located near the leading edge. The CST$_{16}$ shape transitions from near-zero thickness to a nonzero value before returning to near-zero thickness, which is unrealistic.

As anticipated, CST$_6$ converged faster, reaching a solution in 7.115 seconds after 33 iterations, with a convergence speed of 4.638. In comparison, CST$_{16}$ required 29.354 seconds and 124 iterations, with a slower convergence speed of 4.224. The difference in convergence speed is even less pronounced than the constrained case, which itself is already less pronounced than the Daedalus case. This confirms the observation that the number of design variables have a less pronounced impact on less constrained optimization problems. In contrast to the constrained cases, the convergence behavior of both is very erratic, which is likely attributed to the unrestricted design space.

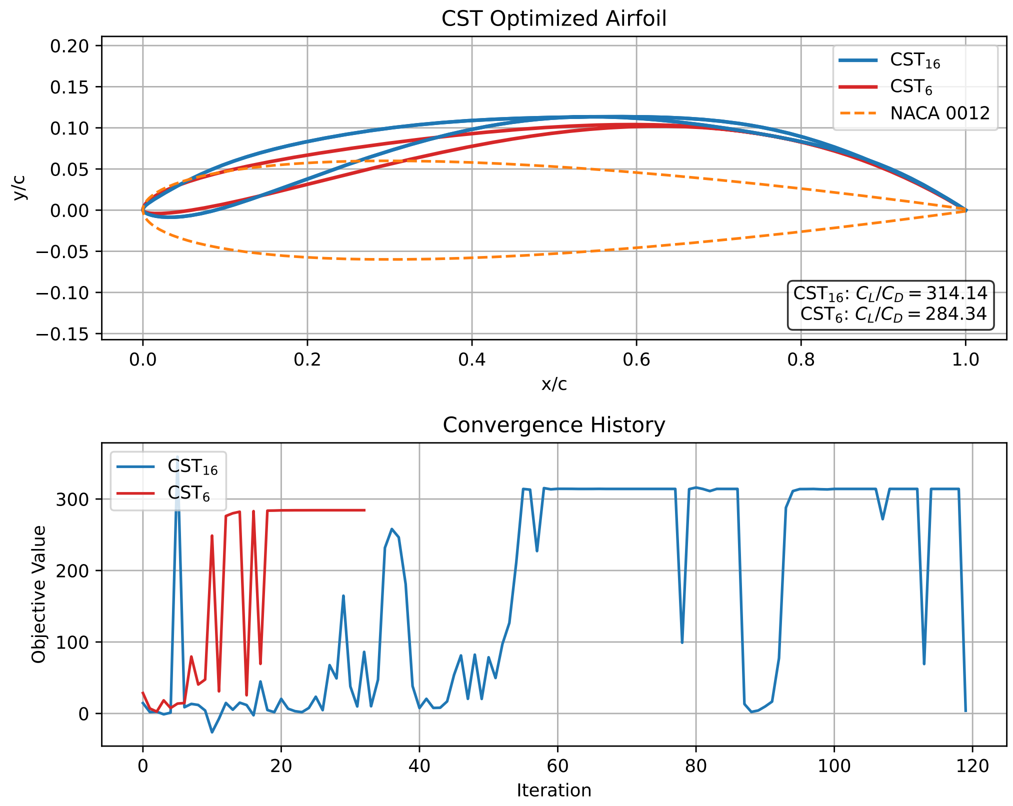


Figure 5.43: Airfoil shape and convergence history of CST_6 and CST_{16} airfoil shape optimization for the unconstrained NACA 0012 optimization case, including lift over drag objective value C_L/C_D

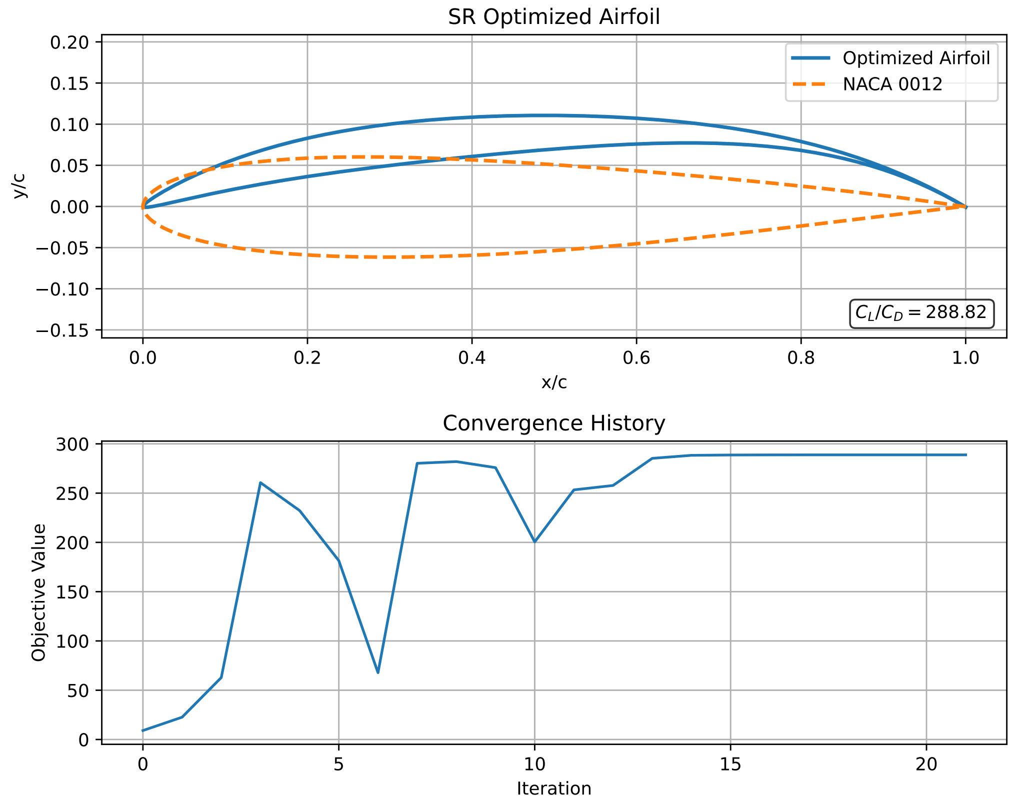


Figure 5.44: Airfoil shape and convergence history of SR airfoil shape optimization for the unconstrained NACA 0012 optimization case, including the lift over drag objective value C_L/C_D

Figure 5.44 shows the unconstrained optimization result for the SR equations, which converged in 2.937 seconds after 22 iterations, or 7.491 iterations per second. For the first time, the SR equations converged to a more aerodynamically efficient shape than the CST$_6$ optimization, with a C_L/C_D of 288.82. Despite the lack of constraints, the convergence behavior is very stable, and the optimizer quickly finds the optimal shape. This is due to the compactness and efficiency of the design space: the equations are specifically trained to generate feasible airfoil shapes, making the method more robust to the absence of constraints. As a result, it still converges to a realistic optimum without the need for a NeuralFoil constraint. Finally, the optimized latent representation of the airfoil is shown below.

$$\boldsymbol{Z_{opt}} = [-2.2869, 2.8057, 4.3556, -1.3744, 2.7651, -4.0366] \tag{5.29}$$

As can be seen here, most optimized latent parameters far exceed the inferred latent distributions from the UIUC dataset shown in Figure 4.3. The SR-VAE decoder is incapable of producing a smooth airfoil using these latent parameters, as shown in Figure 5.45.

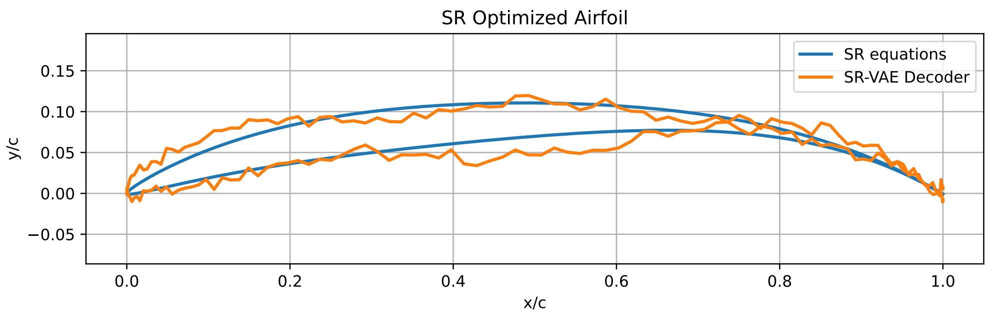


Figure 5.45: SR optimized airfoil (Equation (5.29)) constructed with SR equations and SR-VAE decoder

This indicates that while trained in the same way, the SR equations are capable of generating many more shapes than the decoder. The optimization result for the decoder is shown in Figure 5.46.

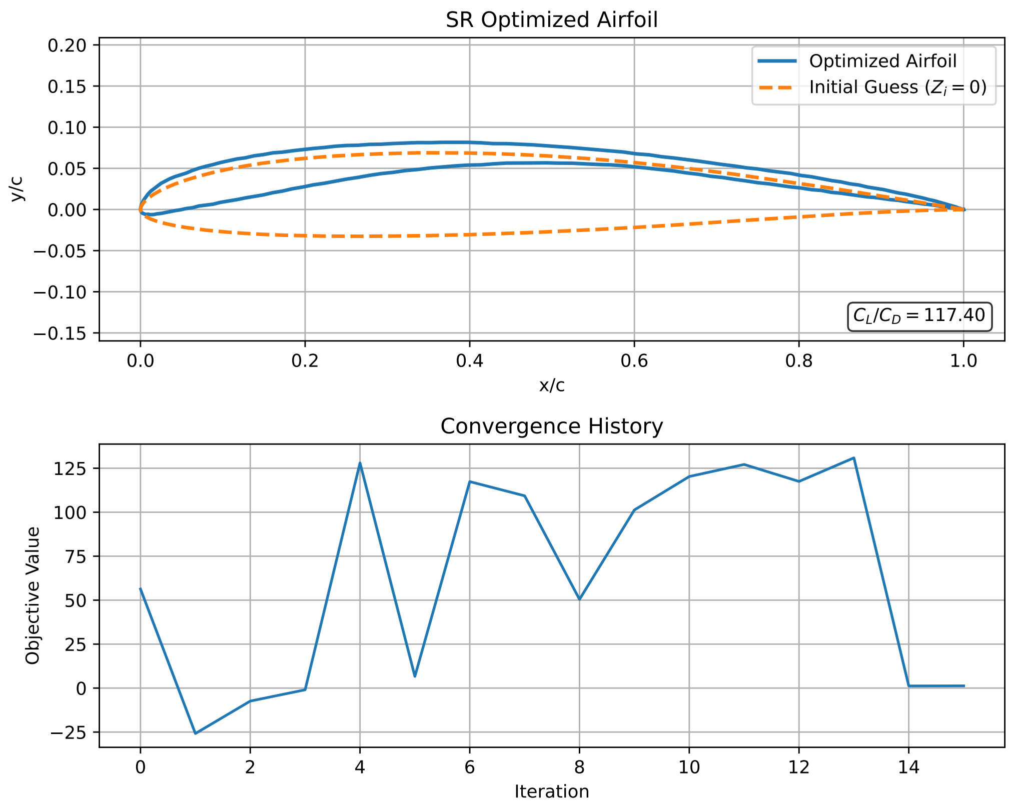


Figure 5.46: Airfoil shape and convergence history of VAE airfoil shape optimization for the unconstrained NACA 0012 optimization case, including the lift over drag objective value C_L/C_D

Once again, the optimization with the SR-VAE decoder required a different initial guess and the addition of a NeuralFoil confidence constraint. This time, however, the optimization converged without issue. This took 4.015 seconds and 17 iterations, which corresponds to 4.234 iterations per second. Unlike the constrained case, there is now visible aerodynamic improvement over the initial guess. Still, the aerodynamic performance of 117.40 remains significantly worse than that of the CST and SR optimizations. The successful convergence confirms that constraints were never the issue. the main limitation is the optimizer's inability to use the decoder to effectively explore the design space.

An overview of the results of this unconstrained optimization for different parameterizations is shown in Table 5.17.

Table 5.17: Summary of the constrained NACA 0012 airfoil optimization results using different parameterizations

Method	**Runtime [s]**	**Iters**	**Iters/s**	C_L/C_D
CST$_6$	7.115	33	4.638	284.37
CST$_{16}$	29.35	124	4.22	314.14
SR Equations	2.94	22	7.49	288.82
SR-VAE	4.02	17	4.234	117.40

The SR equations are shown to outperform the CST$_6$, in both aerodynamic and optimization performance. The use of CST$_{16}$ still leads to the best aerodynamic performance and the SR-VAE is still the worst in almost all categories. Considering the need of additional constraints for the CST and SR-VAE parameterizations, the SR equations seem most suited for unconstrained aerodynamic optimizations. The compact and efficient design space prevents the optimizer from generating unfeasible airfoil shapes.

5.6. Limitations

Throughout this thesis, there have been multiple limitations and possible biases introduced through certain decisions and assumptions. This section will highlight some of them along with the steps taken to minimize their implications.

5.6.1. Computation power

One of the main limitations in this work is the computation power. The latent analysis has been performed on a personal laptop with limited processing power. This influenced the choices during model tuning, such as the maximum complexity of the equations or the max number of iterations. With more computation power, the bias this introduces for the tuning process can be minimized.

Even for SR integration in VAE training, where a high performance workstation was used, the computation power remained a significant limitation. The thousands of symbolic regression searches per training require tremendous processing power. Even after optimizing the training to work with the GPU, memory limits were often exceeded. More importantly, the computational intensity of the SR-VAE training significantly increased the overall runtime per training, limiting the number of times a model can be run. A regular VAE takes less than 10 minutes to train, whereas despite best efforts, an SR-VAE can take more than 24 hours if SR is performed for every batch. Even running SR only once per epoch takes at least 8 hours. This severely limits the capability for tuning, which is why only very few γ values have been used. Ideally, a complete tuning plot should be generated across 20 different γ-values, where the optimal γ can be chosen that balances SR loss and reconstruction loss. Or perhaps β can be tuned while SR is already evaluated during training.

However, it should be noted that once the SR equations have been derived, time can be saved in subsequent optimizations. The equations retain some of the advantages of the VAE, such as a reduced design space and a more parsimonious representation, without the need to use the decoder for every optimization step.

5.6.2. Feature selection

Throughout this thesis, a distinct focus was put on airfoil thickness and camber. The rationale for this is three-fold. First and foremost, the max thickness and max camber were already features that were best captured by the original VAE, which means it should be the easiest to interpret with symbolic regression. Secondly, estimating these features at multiple locations allows for airfoil parameterization, from which all other airfoil characteristics can be extracted. Finally, since parameterization directly pertains to airfoil reconstruction, it naturally fits in VAE training, with a loss calculation similar to the reconstruction loss.

Choosing different features will lead to significant changes for latent analysis and SR-VAE integration. For the former, the tuned PySR settings will completely change, likely leading to different operators and max complexities. For the latter, the SR integration would start to resemble a CVAE, where certain airfoil characteristics have to be extracted from the input and passed to the VAE such that they can be used in the loss calculation.

5.6.3. Stochasticity

A big asterisk throughout this work has been the inherent stochasticity involved in the genetic programming of SR and in VAE training. This means that it is difficult to reproduce results, which was a big issue during the establishment of the original VAE as a new Base VAE. Even two repeated runs with the exact same parameters do not guarantee the same result. For this reason, the work in this thesis has been made as transparent as possible. The reader can find a lot of information about intermediary results and model configuration in the appendix.

5.6.4. Choice of metrics

The method used in this thesis involves a lot of selection processes, from tuning metrics to choosing equations, SR models or VAE models.

The model evaluation for the latent analysis is as of yet very subjective. It is difficult to define a metric that works intuitively well, where an equation with high accuracy but reasonable complexity is chosen. Established metrics that balance complexity and accuracy, such as the Bayesian Information Criterion (BIC) and the Akaike Information Criterion (AIC), have been considered as well but exhibit mixed results. These metrics are also significantly more difficult to interpret.

The main metrics used in this thesis have been the coefficient of determination R^2 and the MSE. The reason for this choice is their robustness, ease of use and interpretability. However, neither is perfect and come with their own limitations.

Although R^2 is inherently normalized, making it easy to compare, it is dependent on the variance in the data. A comparison is only valid if the variance is the same in both cases. This is why R^2 has purposefully been avoided to consider airfoil fit. For example, the FX 79-W-660A has been identified as a notoriously difficult airfoil to reconstruct, due to its extreme thickness and unconventional trailing edge. However, evaluating all separate R^2 scores per airfoil revealed that this airfoil is consistently among the airfoils with the highest R^2, despite the poor fit. The reason for this is the extreme thickness, which leads to a high variance in the dataset and, therefore, to a high R^2.

MSE is more robust to variances, but the values can differ greatly in order of magnitude. This means that in taking the average, high values will dominate. This is not necessarily wrong, since outlier performance is important for individual airfoil reconstructions. However, as has been addressed in the SR-VAE model selection, it can dominate across different datasets and put too much weight on outliers in some instances. This issue has only been alleviated through the use of the standard score for model selection, while there might be other situations where this is required.

To mitigate the disadvantages of both metrics, both have been considered and used in different instances. For the same dataset, both are mathematically equivalent.

5.6.5. Cosine spacing

The use of cosine spacing in this thesis has been inherited from the original VAE. The main reason behind the use of cosine spacing is to concentrate more points towards the trailing edge and leading edge. While this important for training, it does form a bias for the model selection and reconstruction evaluation. Airfoils with a visually worse fit in most parts of the airfoil can still have a lower MSE loss if sections close to the leading edge and trailing edge are reconstructed well. Therefore, a uniform spacing would be more appropriate for model selection.

5.6.6. Choice of test data

The test sets have been kept as consistent as possible, to allow for comparison throughout the multiple research phases. The test data comprises of test sets sampled from the decoder, the UIUC dataset and the AFBench dataset. The AFBench dataset has been used to test with airfoils that have never been used before in training. However, of the 200,000 airfoils in the dataset, many airfoils have been generated by introducing random CST based mutations on existing airfoils. This leads to some very unrealistic airfoils, which might not be relevant. However, to minimize the overall bias, all the UIUC and AFBench airfoils have been used. Taking out difficult airfoils would result in an unfair evaluation of the models.

5.6.7. VAE training sensitivity

In the process of establishing a Base VAE for SR integration in training, important insights related to VAE training were discovered. In particular, it was found that warm-started β-sweeps, intentional or not, can significantly impact the convergence and performance of the VAE during β tuning. The warm starts between β-trials lead to increased VAE performance, at the cost of reproducibility. It was found that taking such a tuned VAE and retraining with repeated warm starts with a constant β, will lead to an improvement after the first instance of retraining but will degrade the VAE performance afterwards. These findings suggest that VAE training dynamics are more sensitive to initialization and scheduling strategies than often assumed, and that care must be taken when iteratively retraining VAEs with warm starts. The use of a warm start for the SR-VAE models limit the reproducibility of this work.

6. Conclusion

This thesis set out to investigate how Symbolic Regression (SR) can be used to improve the latent space interpretability of Variational Autoencoders (VAEs) for airfoil shape optimization. This addresses a key challenge in using deep generative models in engineering: its black box nature. By combining the dimensionality reduction capabilities of VAEs with the interpretability potential of SR, this research aimed to bridge a gap between deep learning and physical understanding, to facilitate its use in practical engineering applications.

To this end, the research was structured around three main questions. To conclude this work, each of these questions will be reflected on below.

Subquestion 1

What role can SR play in deriving physically meaningful relationships from the latent space of VAEs?

SR has been successfully used to interpret the latent space of an existing β-VAE, for geometric airfoil features such as maximum thickness and camber as well as for complete airfoil parameterizations. The resulting equations are shown to effectively capture the latent variables that exhibit the highest correlation for the predicted feature, although the occurrence of some latent variables are less clear. Nevertheless, these interpretable equations allow for valuable insights into role of every latent variable, thereby increasing the overall reliability of the model. This latent analysis also showed a lot of potential for generating parametric equations that show similar generative capability to the decoder. While these equations offer key insights about the effect of every latent variable on the airfoil shape, they did not capture all the active latents and showed limited reconstruction capability for airfoils with extreme thickness and an unconventional TE. Finally, the stochastic nature of genetic programming based SR limits the reproducibility and consistency of the tuned model to generate accurate equations.

Subquestion 2

How does incorporating Symbolic Regression (SR) into the training process influence the interpretability of the latent space in Variational Autoencoders (VAEs)?

The integration of SR into VAE training consistently led to more accurate parametric equations than those obtained from the latent analysis. A new model architecture was introduced, where an SR-based term was added to the traditional VAE loss function: the SR-VAE. Different SR integration strategies were evaluated, and the most effective approach was found to be training while integrating SR per batch with a tuned VAE as warm start, and then retraining once more with the best SR equations fixed throughout. The equations of this final SR-VAE better captured the latent variables while preserving, and some aspects even improving, the generative capability of the VAE decoder. However, the structure of the latent space itself showed limited change due to the use of warm starts, suggesting that SR primarily enhanced interpretability at the output level rather than driving disentanglement internally. This highlights the potential of SR as a tool for training to extract meaning from VAEs, without significantly affecting the structure of the latent space.

Subquestion 3

How does a VAE framework incorporating SR perform in airfoil shape optimization compared to other airfoil representations?

To test the applicability of the parameterizations in airfoil optimizations, first the inverse design capability of the models was tested. The novel SR-VAE and corresponding parametric equations were tested against the original VAE, the SR equations from the latent analysis, and the commonly used CST parameterization. For the same number of variables, the SR-VAE consistently reconstructed airfoils within the smallest MSE error bound. The original VAE performed similarly to CST, while both sets of parametric SR equations lagged behind; however, the new set clearly outperformed those from the latent analysis.

Afterwards, the SR-VAE, SR equations, and CST parameterization were used to solve two aerodynamic optimization problems: a real-world Daedalus-inspired case and a benchmark based on the NACA 0012. In both cases, the CST parameterization achieved the best aerodynamic performance due to its ability to make fine geometric adjustments, while the SR equations obtained comparable results at a significantly higher speed. A key insight here was that optimizations with SR equations showed more stable convergence behavior and resulted in smoother airfoils. However, this smoothness also limited their ability to represent subtle geometric perturbations. In contrast, the SR-VAE was capable of reconstructing a wide range of shapes with very low error, but consistently failed to converge to aerodynamically optimal solutions. This was primarily due to its sensitivity to shapes that deviate far from the training set mean, which restricted the optimizer's ability to explore the design space. Furthermore, it remained the slowest method, largely due to the repeated use of the decoder during optimization. Although the SR equations are trained in the same way as the decoder, the unconstrained optimization showed that they were still capable of representing shapes well outside the original training distribution.

This thesis presents a novel and effective approach to improve the latent space interpretability VAE models using SR. The results prove that accurate, interpretable equations describing the latent space are not only possible but to an extent also practical for actual airfoil design optimizations. This work lays the foundation of latent space interpretability through SR, for engineering and beyond.

7. Recommendations for Future Work

To the knowledge of the author, this is one of the first works that uses Symbolic Regression (SR) to interpret the latent space of deep generative models, particularly for engineering applications such as airfoil shape optimization. While the results are promising, certain limitations, such as computational resources and time constraints, restricted the scope of this research. Several key directions for future work are identified below, with a focus on those that could offer the most meaningful improvements.

Train an SR-VAE from scratch
The current SR-VAE architecture builds on a pre-trained VAE using warm starts, which likely constrained the optimizer's ability to explore alternative latent encodings. Training an SR-VAE from scratch would allow for more flexibility in shaping the latent space and could promote more natural disentanglement and interpretability, especially if guided by SR throughout training.

Extend the method to airfoil features other than thickness and camber
While this thesis primarily focused on camber and thickness, many other geometric or aerodynamic properties could be targeted using SR. Applying symbolic regression to features with lower correlation to the latent space would test the limits of its interpretability, and could uncover whether certain features require more targeted architectural or loss function changes in the generative model.

Integrate SR in parameterization-based deep generative models
In this thesis, SR was applied to VAEs trained on raw airfoil coordinates, which means that SR had to approximate a high-dimensional output, which likely constrained their accuracy and generalization. It could therefore be interesting to investigate the application of SR to deep generative models that restrict the output space by using parameterizations, such as Bézier-GAN, CST-GAN or a VAE based on B-splines. This would make it much more feasible for SR to generate accurate equations.

Additional directions worth exploring:

- Investigate larger max complexities and use fixed seeds for more robust PySR performance.
- Define a custom loss function within PySR to better reflect domain-relevant characteristics
- Test SR methods other than GP-based, such as symbolic metamodelling.
- Limit amount of latent variables PySR can use to encourage disentanglement.
- Actively tune γ during SR-VAE training to better balance VAE losses with SR loss.
- Compare with additional parameterization methods like PARSEC or Bézier in optimization.
- Develop a custom metric that balances interpretability and accuracy.
- Extend SR-based interpretability to other generative models or 3D geometries

References

[1] Christopher M. Bishop and Hugh Bishop, *Deep Learning: Foundations and Concepts*, en. Cham: Springer International Publishing, 2024.

[2] Koosha Sharifani and Mahyar Amini, Machine Learning and Deep Learning: A Review of Methods and Applications, en, SSRN Scholarly Paper, Rochester, NY, 2023.

[3] S.L. Brunton, J.N. Kutz, K. Manohar, A.Y. Aravkin, K. Morgansen, J. Klemisch, N. Goebel, J. Buttrick, J. Poskin, A.W. Blom-Schieber, T. Hogan, and D. McDonald, "Data-driven aerospace engineering: Reframing the industry with machine learning," *AIAA Journal*, Vol. 59, No. 8, pp. 2820–2847, 2021.

[4] Kilian Swannet, Carmine Varriale, and Nguyen Anh Khoa Doan, "Latent Space Correlation for Interpretable Airfoil Parameterization Using Variational Autoencoders," en, in International Council of the Aeronautical Sciences (ICAS), Sep. 2024.

[5] Kilian Swannet, Carmine Varriale, and (Nguyen) Anh Khoa Doan, "Towards Universal Parameterization: Using Variational Autoencoders to Parameterize Airfoils," en, in *AIAA SCITECH 2024 Forum*, Orlando, FL: American Institute of Aeronautics and Astronautics, Jan. 2024.

[6] Diederik P. Kingma and Max Welling, Auto-Encoding Variational Bayes, en, arXiv:1312.6114 [cs, stat], 2013.

[7] Yu-Eop Kang, Dawoon Lee, and Kwanjung Yee, Compact and Intuitive Airfoil Parameterization Method through Physics-aware Variational Autoencoder, en, arXiv:2311.10921 [cs], Nov. 2023.

[8] Xin Wang, Hong Chen, Si'ao Tang, Zihao Wu, and Wenwu Zhu, "Disentangled Representation Learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 46, No. 12, pp. 9677–9696, Dec. 2024, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[9] Emrullah ŞAHiN, Naciye Nur Arslan, and Durmuş Özdemir, "Unlocking the black box: An in-depth review on interpretability, explainability, and reliability in deep learning," en, *Neural Computing and Applications*, Nov. 2024.

[10] Michael A Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, USA, 2015, Vol. 25.

[11] Yuhan Bai, "RELU-Function and Derived Function Review," en, *SHS Web of Conferences*, Vol. 144, A. Luqman, Q. Zhang, and W. Liu, Eds., p. 02 006, 2022.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning, Adaptive Computation and Machine Learning*. The MIT Press, 2016.

[13] D. Randall Wilson and Tony R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Networks*, Vol. 16, No. 10, pp. 1429–1451, Dec. 2003.

[14] Jakub M. Tomczak, *Deep Generative Modeling*, en. Cham: Springer International Publishing, 2022.

[15] Diederik P. Kingma and Max Welling, "An Introduction to Variational Autoencoders," en, *Foundations and Trends® in Machine Learning*, Vol. 12, No. 4, pp. 307–392, 2019.

[16] Junhai Zhai, Sufang Zhang, Junfen Chen, and Qiang He, "Autoencoder and Its Various Variants," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, ISSN: 2577-1655, Oct. 2018, pp. 415–419.

[17] David Foster and Karl J. Friston, *Generative deep learning: Teaching machines to paint, write, compose, and play*, en, Second edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2023.

[18] David Charte, Francisco Charte, María J. Del Jesus, and Francisco Herrera, "An analysis on the use of autoencoders for representation learning: Fundamentals, learning task case studies, explainability and challenges," en, *Neurocomputing*, Vol. 404, pp. 93–107, Sep. 2020.

[19] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe, "Variational Inference: A Review for Statisticians," en, *Journal of the American Statistical Association*, Vol. 112, No. 518, pp. 859–877, Apr. 2017.

[20] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester, *A Modern Introduction to Probability and Statistics: Understanding Why and How*, en. Springer Science & Business Media, Mar. 2006, Google-Books-ID: TEcmHJX67coC.

[21] Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner, Understanding disentangling in \$β\$-VAE, arXiv:1804.03599, Apr. 2018.

[22] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner, "B-VAE: LEARNING BASIC VISUAL CONCEPTS WITH A CONSTRAINED VARIATIONAL FRAMEWORK," en, 2017.

[23] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, "Generative adversarial networks," en, *Communications of the ACM*, Vol. 63, No. 11, pp. 139–144, Oct. 2020.

[24] ——, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, Vol. 27, Curran Associates, Inc., 2014.

[25] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath, "Generative Adversarial Networks: An Overview," *IEEE Signal Processing Magazine*, Vol. 35, No. 1, pp. 53–65, Jan. 2018, Conference Name: IEEE Signal Processing Magazine.

[26] Yu-Eop Kang, Dawoon Lee, and Kwanjung Yee, "Physically interpretable airfoil parameterization using variational autoencoder-based generative modeling," en, in *AIAA SCITECH 2024 Forum*, Orlando, FL: American Institute of Aeronautics and Astronautics, Jan. 2024.

[27] Gabriel Achour, Woong Je Sung, Olivia J. Pinon-Fischer, and Dimitri N. Mavris, "Development of a Conditional Generative Adversarial Network for Airfoil Shape Optimization," en, in *AIAA Scitech 2020 Forum*, Orlando, FL: American Institute of Aeronautics and Astronautics, Jan. 2020.

[28] Emre Yilmaz and Brian German, "Conditional Generative Adversarial Network Framework for Airfoil Inverse Design," en, in *AIAA AVIATION 2020 FORUM*, VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, Jun. 2020.

[29] Qiuyi Chen, Jun Wang, Phillip Pope, Wei (Wayne) Chen, and Mark Fuge, "Inverse Design of Two-Dimensional Airfoils Using Conditional Generative Models and Surrogate Log-Likelihoods," *Journal of Mechanical Design*, Vol. 144, No. 021712, Dec. 2021.

[30] Kazunari Wada, Katsuyuki Suzuki, and Kazuo Yonekura, "Physics-guided training of GAN to improve accuracy in airfoil design synthesis," *Computer Methods in Applied Mechanics and Engineering*, Vol. 421, p. 116 746, Mar. 2024.

[31] Wei Chen, Kevin Chiu, and Mark D. Fuge, "Airfoil Design Parameterization and Optimization Using Bézier Generative Adversarial Networks," en, *AIAA Journal*, Vol. 58, No. 11, pp. 4723–4735, Nov. 2020.

[32] Xiaosong Du, Ping He, and Joaquim R. R. A. Martins, "A B-Spline-based Generative Adversarial Network Model for Fast Interactive Airfoil Aerodynamic Optimization," en, in *AIAA Scitech 2020 Forum*, Orlando, FL: American Institute of Aeronautics and Astronautics, Jan. 2020.

[33] Yuyang Wang, Kenji Shimada, and Amir Barati Farimani, "Airfoil GAN: Encoding and synthesizing airfoils for aerodynamic shape optimization," en, *Journal of Computational Design and Engineering*, Vol. 10, No. 4, pp. 1350–1362, Jul. 2023.

[34] Xiaosong Du, Ping He, and Joaquim R.R.A. Martins, "Rapid airfoil design optimization via neural networks-based parameterization and surrogate modeling," en, *Aerospace Science and Technology*, Vol. 113, p. 106 701, Jun. 2021.

[35] Jinxing Lin, Chenliang Zhang, Xiaoye Xie, Xingyu Shi, Xiaoyu Xu, and Yanhui Duan, "CST-GANs: A Generative Adversarial Network Based on CST Parameterization for the Generation of Smooth Airfoils," in *2022 IEEE International Conference on Unmanned Systems (ICUS)*, ISSN: 2771-7372, Oct. 2022, pp. 600–605.

[36] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen, “Improved Techniques for Training GANs,” in *Advances in Neural Information Processing Systems*, Vol. 29, Curran Associates, Inc., 2016.

[37] Jiaqing Kou, Laura Botero-Bolívar, Román Ballano, Oscar Marino, Leandro De Santana, Eusebio Valero, and Esteban Ferrer, “Aeroacoustic airfoil shape optimization enhanced by autoencoders,” en, *Expert Systems with Applications*, Vol. 217, p. 119 513, May 2023.

[38] Jing Wang, Runze Li, Cheng He, Haixin Chen, Ran Cheng, Chen Zhai, and Miao Zhang, “An inverse design method for supercritical airfoil based on conditional generative models,” *Chinese Journal of Aeronautics*, Vol. 35, No. 3, pp. 62–74, Mar. 2022.

[39] Xu Wang, Weiqi Qian, Tun Zhao, Hai Chen, Lei He, Haisheng Sun, and Yuan Tian, “A generative design method of airfoil based on conditional variational autoencoder,” *Engineering Applications of Artificial Intelligence*, Vol. 139, p. 109 461, Jan. 2025.

[40] Kazuo Yonekura and Katsuyuki Suzuki, “Data-driven design exploration method using conditional variational autoencoder for airfoil design,” en, *Structural and Multidisciplinary Optimization*, Vol. 64, No. 2, pp. 613–624, Aug. 2021.

[41] Kazuo Yonekura, Kazunari Wada, and Katsuyuki Suzuki, “Generating various airfoils with required lift coefficients by combining NACA and Joukowski airfoils using conditional variational autoencoders,” en, *Engineering Applications of Artificial Intelligence*, Vol. 108, p. 104 560, Feb. 2022.

[42] Hairun Xie, Jing Wang, and Miao Zhang, “Parametric generative schemes with geometric constraints for encoding and synthesizing airfoils,” *Engineering Applications of Artificial Intelligence*, Vol. 128, p. 107 505, Feb. 2024.

[43] Runze Li, Yufei Zhang, and Haixin Chen, “Physically Interpretable Feature Learning of Supercritical Airfoils Based on Variational Autoencoders,” en, *AIAA Journal*, Vol. 60, No. 11, pp. 6168–6182, Nov. 2022.

[44] Michael Selig, UIUC Airfoil Data Site, 1996.

[45] Vaishnavi Patil, Matthew Evanusa, and Joseph JaJa, “DOT-VAE: Disentangling One Factor at a Time,” en, in *Artificial Neural Networks and Machine Learning – ICANN 2022*, Elias Pimenidis, Plamen Angelov, Chrisina Jayne, Antonios Papaleonidas, and Mehmet Aydin, Eds., Vol. 13529, Series Title: Lecture Notes in Computer Science, Cham: Springer International Publishing, 2022, pp. 109–120.

[46] Ricky T. Q. Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud, “Isolating Sources of Disentanglement in Variational Autoencoders,” in *Advances in Neural Information Processing Systems*, Vol. 31, Curran Associates, Inc., 2018.

[47] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan, Variational Inference of Disentangled Latent Concepts from Unlabeled Observations, arXiv:1711.00848, Dec. 2018.

[48] Hyunjik Kim and Andriy Mnih, “Disentangling by Factorising,” en, in *Proceedings of the 35th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, Jul. 2018, pp. 2649–2658.

[49] Shuyang Gao, Rob Brekelmans, Greg Ver Steeg, and Aram Galstyan, “Auto-Encoding Total Correlation Explanation,” en, in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, ISSN: 2640-3498, PMLR, Apr. 2019, pp. 1157–1166.

[50] Minyoung Kim, Yuting Wang, Pritish Sahu, and Vladimir Pavlovic, Relevance Factor VAE: Learning and Identifying Disentangled Factors, arXiv:1902.01568, Feb. 2019.

[51] Satosi Watanabe, “Information Theoretical Analysis of Multivariate Correlation,” *IBM Journal of Research and Development*, Vol. 4, No. 1, pp. 66–82, Jan. 1960, Conference Name: IBM Journal of Research and Development.

[52] Zheng Ding, Yifan Xu, Weijian Xu, Gaurav Parmar, Yang Yang, Max Welling, and Zhuowen Tu, “Guided Variational Autoencoder for Disentanglement Learning,” en, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Jun. 2020, pp. 7917–7926.

[53] Jack Klys, Jake Snell, and Richard Zemel, “Learning Latent Subspaces in Variational Autoencoders,” in *Advances in Neural Information Processing Systems*, Vol. 31, Curran Associates, Inc., 2018.

[54] Zhilin Zheng and Li Sun, “Disentangling Latent Space for VAE by Label Relevant/Irrelevant Dimensions,” en, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA: IEEE, Jun. 2019, pp. 12 184–12 193.

[55] Dimitrios Angelis, Filippos Sofos, and Theodoros E. Karakasidis, “Artificial Intelligence in Physical Sciences: Symbolic Regression Trends and Perspectives,” en, *Archives of Computational Methods in Engineering*, Vol. 30, No. 6, pp. 3845–3865, Jul. 2023.

[56] Nour Makke and Sanjay Chawla, “Interpretable scientific discovery with symbolic regression: A review,” en, *Artificial Intelligence Review*, Vol. 57, No. 1, p. 2, Jan. 2024.

[57] Miles Cranmer, Interpretable Machine Learning for Science with PySR and SymbolicRegression.jl, en, arXiv:2305.01582 [astro-ph, physics:physics], May 2023.

[58] Steven L. Brunton, Steven L. Brunton, Joshua L. Proctor, Joshua L. Proctor, J. Nathan Kutz, and J. Nathan Kutz, “Discovering governing equations from data by sparse identification of nonlinear dynamical systems,” *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 113, No. 15, pp. 3932–3937, Apr. 2016, MAG ID: 2239232218.

[59] Kathleen Champion, Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton, “Data-driven discovery of coordinates and governing equations,” en, *Proceedings of the National Academy of Sciences*, Vol. 116, No. 45, pp. 22 445–22 451, Nov. 2019.

[60] Georg Martius and Christoph H. Lampert, Extrapolation and learning equations, arXiv:1610.02995, Oct. 2016.

[61] Subham Sahoo, Christoph Lampert, and Georg Martius, “Learning Equations for Extrapolation and Control,” en, in *Proceedings of the 35th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, Jul. 2018, pp. 4442–4450.

[62] Trent McConaghy, “Ffx: Fast, Scalable, Deterministic Symbolic Regression Technology,” en, in *Genetic Programming Theory and Practice IX*, Rick Riolo, Ekaterina Vladislavleva, and Jason H. Moore, Eds., New York, NY: Springer, 2011, pp. 235–260.

[63] Konstantinos Papastamatiou, Filippos Sofos, and Theodoros E. Karakasidis, “Machine learning symbolic equations for diffusion with physics-based descriptions,” en, *AIP Advances*, Vol. 12, No. 2, pp. 025 004–10, Feb. 2022, Publisher: AIP Publishing LLC.

[64] Brenden K. Petersen, Mikel Landajuela, T. Nathan Mundhenk, Claudio P. Santiago, Soo K. Kim, and Joanne T. Kim, Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients, arXiv:1912.04871, Apr. 2021.

[65] Michael Schmidt and Hod Lipson, “Distilling Free-Form Natural Laws from Experimental Data,” *Science*, Vol. 324, No. 5923, pp. 81–85, Apr. 2009, Publisher: American Association for the Advancement of Science.

[66] Silviu-Marian Udrescu and Max Tegmark, “AI Feynman: A physics-inspired method for symbolic regression,” en, *Science Advances*, Vol. 6, No. 16, eaay2631, Apr. 2020.

[67] Arthur Grundner, Tom Beucler, Pierre Gentine, and Veronika Eyring, “Data-Driven Equation Discovery of a Cloud Cover Parameterization,” en, *Journal of Advances in Modeling Earth Systems*, Vol. 16, No. 3, e2023MS003763, 2024, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1029/2023MS003763.

[68] Kaze W. K. Wong and Miles Cranmer, Automated discovery of interpretable gravitational-wave population models, arXiv:2207.12409 [astro-ph], Jul. 2022.

[69] Bogdan Burlacu, Gabriel Kronberger, and Michael Kommenda, “Operon C++: An efficient genetic programming framework for symbolic regression,” in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, ser. GECCO ’20, New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 1562–1570.

[70] Trevor Stephens, GPlearn, original-date: 2015-03-26T01:01:14Z, Nov. 2024.

[71] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, Vol. 30, Curran Associates, Inc., 2017.

[72] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo, “Neural Symbolic Regression that scales,” en, in *Proceedings of the 38th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, Jul. 2021, pp. 936–945.

[73] Mojtaba Valipour, Bowen You, Maysum Panju, and Ali Ghodsi, SymbolicGPT: A Generative Transformer Model for Symbolic Regression, arXiv:2106.14131, Jun. 2021.

[74] Pierre-alexandre Kamienny, Stéphane d’Ascoli, Guillaume Lample, and Francois Charton, “End-to-end Symbolic Regression with Transformers,” en, *Advances in Neural Information Processing Systems*, Vol. 35, pp. 10 269–10 281, Dec. 2022.

[75] Ahmed M Alaa and Mihaela van der Schaar, “Demystifying Black-box Models with Symbolic Metamodels,” en, *Advances in Neural Information Processing Systems*, Vol. 32, 2019.

[76] Cole Miles, Matthew R. Carbone, Erica J. Sturm, Deyu Lu, Andreas Weichselbaum, Kipton Barros, and Robert M. Konik, “Machine learning of Kondo physics using variational autoencoders and symbolic regression,” en, *Physical Review B*, Vol. 104, No. 23, p. 235 111, Dec. 2021.

[77] Zakaria Patel and Sebastian J. Wetzel, Closed-Form Interpretation of Neural Network Latent Spaces with Symbolic Gradients, arXiv:2409.05305, Sep. 2024.

[78] Tameem Adel, Zoubin Ghahramani, and Adrian Weller, “Discovering Interpretable Representations for Both Deep Generative and Discriminative Models,” en, in *Proceedings of the 35th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, Jul. 2018, pp. 50–59.

[79] Charles L. Ladson, Cuyler W. Brooks, Acquilla S. Hill, and Darrell W. Sproles, “Computer Program to Obtain Ordinates for NACA Airfoils,” L-17509, NTRS Author Affiliations: NASA Langley Research Center, Computer Sciences Corp. NTRS Document ID: 19970008124 NTRS Research Center: Langley Research Center (LaRC), Dec. 1996.

[80] Jian Liu, Jianyu Wu, Hairun Xie, Guoqing Zhang, Jing Wang, Wei Liu, Wanli Ouyang, Junjun Jiang, Xianming Liu, Shixiang Tang, and Miao Zhang, AFBench: A Large-scale Benchmark for Airfoil Design, arXiv:2406.18846 [cs], Oct. 2024.

[81] Irina Higgins, Le Chang, Victoria Langston, Demis Hassabis, Christopher Summerfield, Doris Tsao, and Matthew Botvinick, “Unsupervised deep learning identifies semantic disentanglement in single inferotemporal face patch neurons,” en, *Nature Communications*, Vol. 12, No. 1, p. 6456, Nov. 2021, Publisher: Nature Publishing Group.

[82] Peter D. Sharpe, “Accelerating Practical Engineering Design Optimization with Computational Graph Transformations,” PhD Thesis, Massachusetts Institute of Technology, 2024.

[83] ——, “AeroSandbox: A Differentiable Framework for Aircraft Design Optimization,” Master’s thesis, Massachusetts Institute of Technology, 2021.

[84] Mark Drela, “Low-Reynolds-number airfoil design for the M.I.T. Daedalus prototype- A case study,” *Journal of Aircraft*, Vol. 25, No. 8, pp. 724–732, Aug. 1988, Publisher: American Institute of Aeronautics and Astronautics.

[85] ——, “Pros & Cons of Airfoil Optimization,” in *Frontiers of Computational Fluid Dynamics 1998*, WORLD SCIENTIFIC, Nov. 1998, pp. 363–381.

[86] Feng Deng, Cheng Xue, and Ning Qin, “Parameterizing Airfoil Shape Using Aerodynamic Performance Parameters,” *AIAA Journal*, Vol. 60, No. 7, pp. 4399–4412, 2022, Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.J061464.

[87] Peter Sharpe and R. John Hansman, NeuralFoil: An Airfoil Aerodynamics Analysis Tool Using Physics-Informed Machine Learning, arXiv:2503.16323 [physics], Mar. 2025.

[88] Pablo Rouco, Pedro Orgeira-Crespo, Guillermo David Rey González, and Fernando Aguado-Agelet, “Airfoil Optimization and Analysis Using Global Sensitivity Analysis and Generative Design,” en, *Aerospace*, Vol. 12, No. 3, p. 180, Mar. 2025, Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

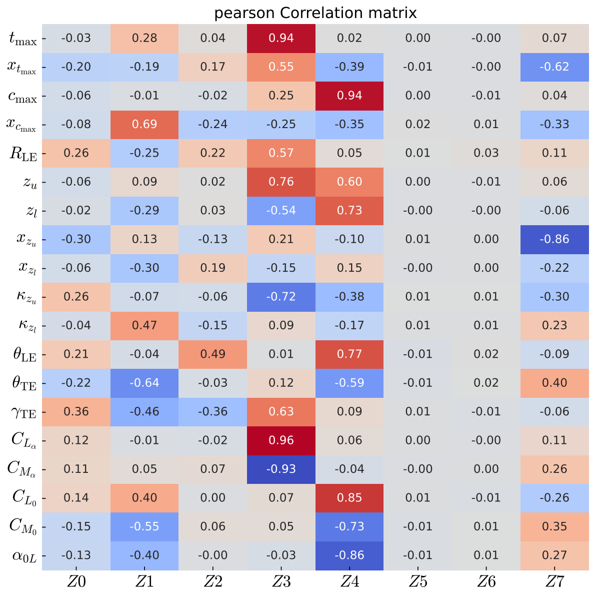
A. β-VAE by Swannet et al.

This chapter repeats some of the results from the β-VAE for reference. For more details regarding results and the VAE architecture, the reader is referred to work by Swannet et al.[4, 5]. At the end of this appendix, a custom tuned version of this VAE is included to be used as a basis for SR training integration.

As briefly described at the start of Subsection 2.4.2, the β-VAE by Swannet al. [4, 5] retains the basic VAE architecture, where the KL-divergence term is weighted by a factor β to promote disentanglement. The VAE is trained on the UIUC dataset, with fixed cosine spacing to force a higher density of points to the leading edge and trailing edge. The conscious decision was made here not to impose any conditions on the latent space to minimize user-imposed bias. This appendix outlines the most important results, with a short reflection.

A.1. Pearson Correlation Matrix

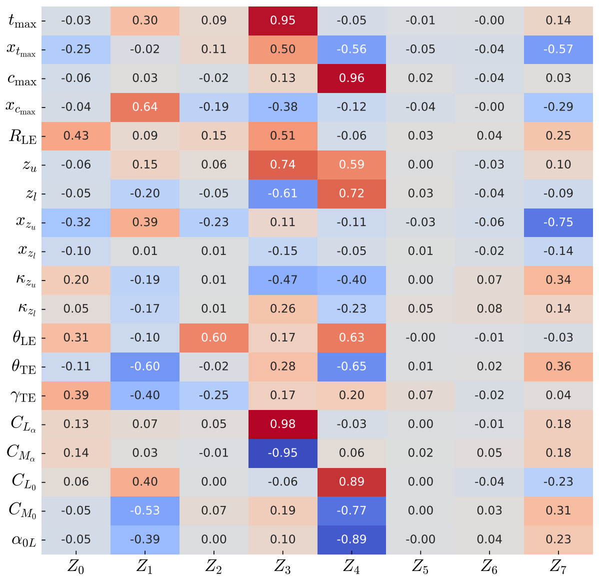
The main results of the article are Pearson's correlation matrices between the airfoil features and the latent variables, as shown in Figure A.1.



pearson Correlation matrix

	Z0	Z1	Z2	Z3	Z4	Z5	Z6	Z7
$t_{\max}$	-0.03	0.28	0.04	0.94	0.02	0.00	-0.00	0.07
$x_{t_{\max}}$	-0.20	-0.19	0.17	0.55	-0.39	-0.01	-0.00	-0.62
$c_{\max}$	-0.06	-0.01	-0.02	0.25	0.94	0.00	-0.01	0.04
$x_{c_{\max}}$	-0.08	0.69	-0.24	-0.25	-0.35	0.02	0.01	-0.33
R_{LE}	0.26	-0.25	0.22	0.57	0.05	0.01	0.03	0.11
z_u	-0.06	0.09	0.02	0.76	0.60	0.00	-0.01	0.06
z_l	-0.02	-0.29	0.03	-0.54	0.73	-0.00	-0.00	-0.06
x_{z_u}	-0.30	0.13	-0.13	0.21	-0.10	0.01	0.00	-0.86
x_{z_l}	-0.06	-0.30	0.19	-0.15	0.15	-0.00	0.00	-0.22
κ_{z_u}	0.26	-0.07	-0.06	-0.72	-0.38	0.01	0.01	-0.30
κ_{z_l}	-0.04	0.47	-0.15	0.09	-0.17	0.01	0.01	0.23
θ_{LE}	0.21	-0.04	0.49	0.01	0.77	-0.01	0.02	-0.09
θ_{TE}	-0.22	-0.64	-0.03	0.12	-0.59	-0.01	0.02	0.40
γ_{TE}	0.36	-0.46	-0.36	0.63	0.09	0.01	-0.01	-0.06
C_{L_α}	0.12	-0.01	-0.02	0.96	0.06	0.00	-0.00	0.11
C_{M_α}	0.11	0.05	0.07	-0.93	-0.04	-0.00	0.00	0.26
C_{L_0}	0.14	0.40	0.00	0.07	0.85	0.01	-0.01	-0.26
C_{M_0}	-0.15	-0.55	0.06	0.05	-0.73	-0.01	0.01	0.35
α_{0L}	-0.13	-0.40	-0.00	-0.03	-0.86	-0.01	0.01	0.27

(a) Latent parameter traversal data



	Z_0	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7
$t_{\max}$	-0.03	0.30	0.09	0.95	-0.05	-0.01	-0.00	0.14
$x_{t_{\max}}$	-0.25	-0.02	0.11	0.50	-0.56	-0.05	-0.04	-0.57
$c_{\max}$	-0.06	0.03	-0.02	0.13	0.96	0.02	-0.04	0.03
$x_{c_{\max}}$	-0.04	0.64	-0.19	-0.38	-0.12	-0.04	-0.00	-0.29
R_{LE}	0.43	0.09	0.15	0.51	-0.06	0.03	0.04	0.25
z_u	-0.06	0.15	0.06	0.74	0.59	0.00	-0.03	0.10
z_l	-0.05	-0.20	-0.05	-0.61	0.72	0.03	-0.04	-0.09
x_{z_u}	-0.32	0.39	-0.23	0.11	-0.11	-0.03	-0.06	-0.75
x_{z_l}	-0.10	0.01	0.01	-0.15	-0.05	0.01	-0.02	-0.14
κ_{z_u}	0.20	-0.19	0.01	-0.47	-0.40	0.00	0.07	0.34
κ_{z_l}	0.05	-0.17	0.01	0.26	-0.23	0.05	0.08	0.14
θ_{LE}	0.31	-0.10	0.60	0.17	0.63	-0.00	-0.01	-0.03
θ_{TE}	-0.11	-0.60	-0.02	0.28	-0.65	0.01	0.02	0.36
γ_{TE}	0.39	-0.40	-0.25	0.17	0.20	0.07	-0.02	0.04
C_{L_α}	0.13	0.07	0.05	0.98	-0.03	0.00	-0.01	0.18
C_{M_α}	0.14	0.03	-0.01	-0.95	0.06	0.02	0.05	0.18
C_{L_0}	0.06	0.40	0.00	-0.06	0.89	0.00	-0.04	-0.23
C_{M_0}	-0.05	-0.53	0.07	0.19	-0.77	0.00	0.03	0.31
α_{0L}	-0.05	-0.39	0.00	0.10	-0.89	-0.00	0.04	0.23

(b) 500 random samples $\boldsymbol{z} \sim \mathcal{U}(\mu - 2\sigma, \mu + 2\sigma)$

Figure A.1: Pearson correlation matrices of the base β-VAE constructed by varying latent dimension in a range of two standard deviations from its mean [4]

Both matrices are based on data where the latent dimensions are varied within a range of two standard deviations σ from its mean μ. In Figure A.1a, each latent dimension is varied one at the time with a fixed step size. For Figure A.1b, all latent dimensions are sampled from a uniform random distribution. This matrix shows the degree of disentanglement of the latent variables with respect to geometric and aerodynamic airfoil features.

As can be seen from the figure, thickness and camber are captured most effectively by individual latent parameters. As expected, the corresponding aerodynamic characteristics that are most influenced by these geometric features show a strong correlation as well.

A.2. Validation plots

The validation plots are regenerated using MSE rather than the relative MSE used by Swannet et al. Furthermore, the original paper did not take the mean, which is why the decoded losses indicated the sum of squares loss rather than the intended mean squared loss. This is corrected in figure A.2.

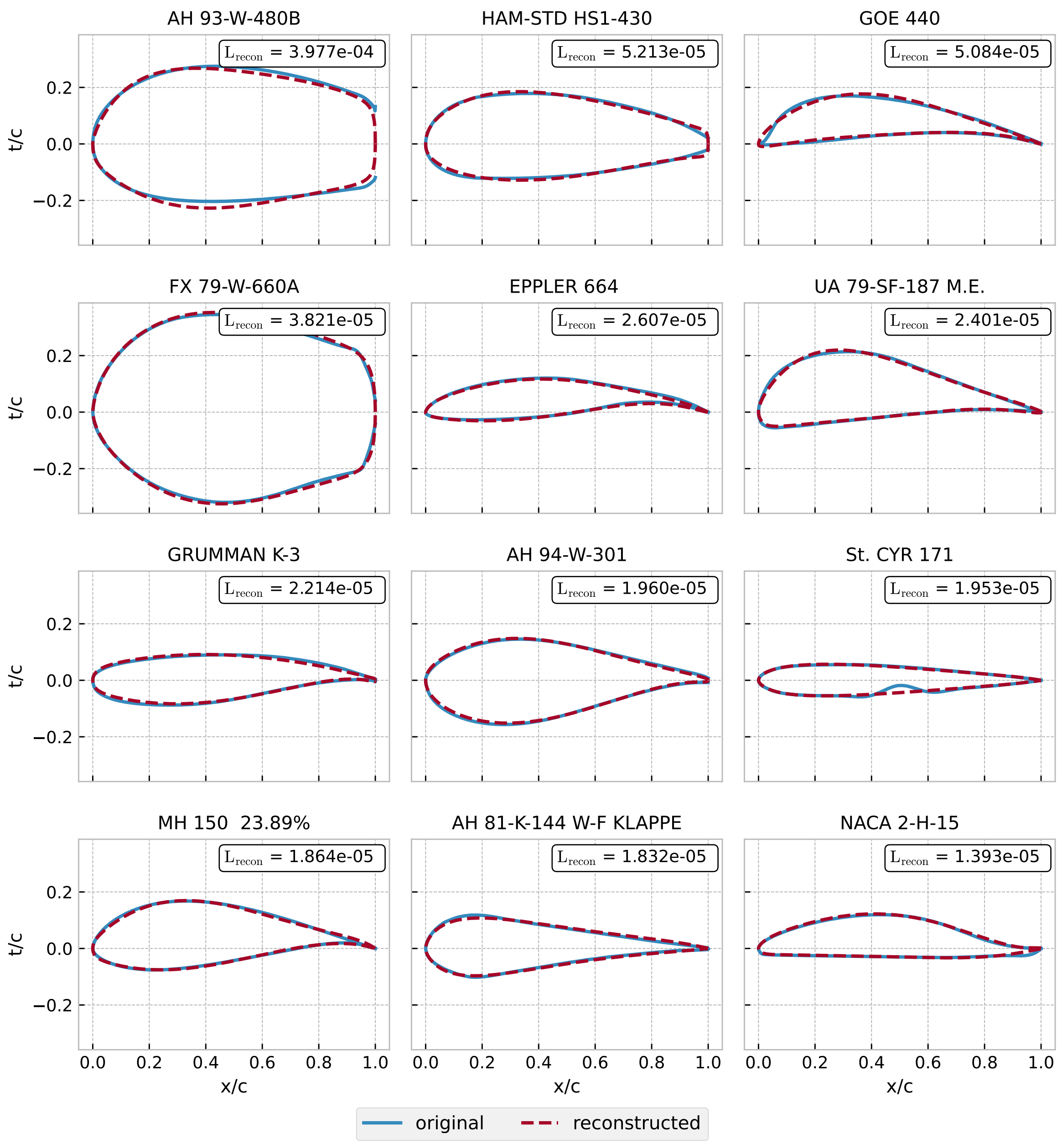


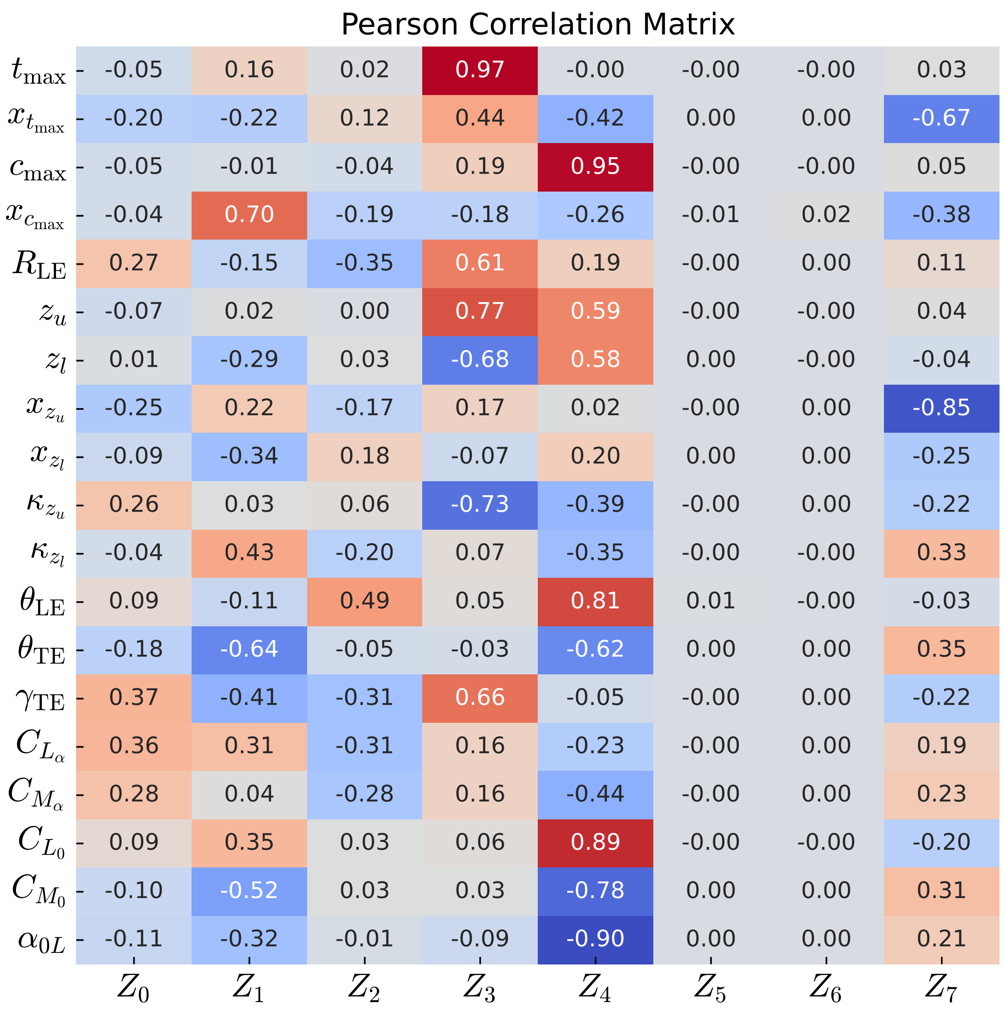
Figure A.2: Validation plots showing the reconstruction of the 12 airfoils with the highest reconstruction loss. (Corrected) MSE reconstruction loss values are marked in the corner of each plot. [4]

A.3. Custom tuned base-VAE

As mentioned in the thesis report, a baseline VAE had to be established for SR training integration consistent with the new loss function definitions. The results of custom tuned VAE based on the original VAE, at $\beta = 2.154 \cdot 10^{-5}$ is shown below.

A.3.1. Pearson Correlation matrix

The Pearson correlation matrix is quite similar to the original VAE, showing the same overall trends per latent parameter. However, it does show slightly higher correlations for thickness and camber. Overall, the strongest correlations seem to a bit more pronounced compared to the original Pearson correlation matrix. There are also some key differences for the aerodynamic parameters due to the use of NeuralFoil instead of XFOIL. Since this thesis only considers geometric parameters, this is neglected.



Pearson Correlation Matrix

	Z_0	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7
$t_{\max}$	-0.05	0.16	0.02	0.97	-0.00	-0.00	-0.00	0.03
$x_{t_{\max}}$	-0.20	-0.22	0.12	0.44	-0.42	0.00	0.00	-0.67
$c_{\max}$	-0.05	-0.01	-0.04	0.19	0.95	-0.00	-0.00	0.05
$x_{c_{\max}}$	-0.04	0.70	-0.19	-0.18	-0.26	-0.01	0.02	-0.38
R_{LE}	0.27	-0.15	-0.35	0.61	0.19	-0.00	0.00	0.11
z_u	-0.07	0.02	0.00	0.77	0.59	-0.00	-0.00	0.04
z_l	0.01	-0.29	0.03	-0.68	0.58	0.00	-0.00	-0.04
x_{z_u}	-0.25	0.22	-0.17	0.17	0.02	-0.00	0.00	-0.85
x_{z_l}	-0.09	-0.34	0.18	-0.07	0.20	0.00	0.00	-0.25
κ_{z_u}	0.26	0.03	0.06	-0.73	-0.39	-0.00	0.00	-0.22
κ_{z_l}	-0.04	0.43	-0.20	0.07	-0.35	-0.00	-0.00	0.33
θ_{LE}	0.09	-0.11	0.49	0.05	0.81	0.01	-0.00	-0.03
θ_{TE}	-0.18	-0.64	-0.05	-0.03	-0.62	0.00	0.00	0.35
γ_{TE}	0.37	-0.41	-0.31	0.66	-0.05	-0.00	0.00	-0.22
C_{L_α}	0.36	0.31	-0.31	0.16	-0.23	-0.00	0.00	0.19
C_{M_α}	0.28	0.04	-0.28	0.16	-0.44	-0.00	0.00	0.23
C_{L_0}	0.09	0.35	0.03	0.06	0.89	-0.00	-0.00	-0.20
C_{M_0}	-0.10	-0.52	0.03	0.03	-0.78	0.00	0.00	0.31
α_{0L}	-0.11	-0.32	-0.01	-0.09	-0.90	0.00	0.00	0.21

Figure A.3: Pearson correlation matrix of base-VAE re-tuned in the new code architecture, with $\beta = 2.154 \cdot 10^{-5}$

B. Summary of PySR runs for latent analysis

This appendix contain a summary of all the runs conducted with PySR during the latent analysis, for the feature equations as well as the parameterization equation. An overview of model parameter combinations and the amount of runs can be found below, for the feature equations as well as the parameterization equations.

B.1. Feature SR runs

The different model configurations to estimate t_{max} and c_{max} is shown below in Table B.1 and Table B.2 respectively.

The equations with the highest R^2 for every complexity for t_{max} and c_{max} is shown in Table B.3 and Table B.4, respectively. Note that while there have been many more model configurations using the unary operator sin, these almost never make it to the best equations. Overall, the tuned PySR settings are the most prevalent in these equations.

Table B.1: All model parameters combinations and the amount of runs for t_{max}, sorted by most runs

Binary Operators	Unary Operators	Max Size	Iterations	Runs
[’*’, ’+’]	[]	15	300	22
$['*','+','\wedge']$	[]	20	500	21
[’*’, ’+’]	[’sin’]	15	300	12
[’*’, ’+’]	[’sin’]	15	400	12
[’*’, ’+’]	[’sin’]	15	200	7
[’*’, ’+’]	[’sin’]	15	100	7
[’*’, ’+’]	[’sin’]	30	200	6
$['*','+','\wedge']$	[]	25	500	6
[’*’, ’+’]	[’sin’]	12	300	5
[’*’, ’+’]	[’sin’]	16	500	5
[’*’, ’+’]	[’sin’]	14	300	4
[’*’, ’+’]	[]	12	300	4
[’*’, ’+’, ’pow_int’]	[]	20	500	3
[’*’, ’+’, ’pow_int’]	[]	25	500	3
[’*’, ’+’]	[’sin’]	12	200	3
[’*’, ’+’]	[’sin’]	12	400	3
[’*’, ’+’]	[’sin’]	16	300	2
[’*’, ’+’]	[]	15	500	2
[’*’, ’+’]	[’sin’]	30	300	2
[’*’, ’+’]	[’sin’]	15	1000	2
[’*’, ’+’, ’pow_int’]	[]	30	500	2
$['*','+','\wedge']$	[’sin’]	12	300	2
[’*’, ’+’]	[]	25	500	2
[’*’, ’+’]	[’sin’]	16	200	2
$['*','+','\wedge']$	[]	25	10000	1
[’*’, ’+’, ’pow_int’]	[]	20	3000	1
$['*','+','\wedge']$	[’exp’, ’sin’, ’log’]	15	1000	1
[’*’, ’+’]	[’sin’]	15	20000	1
[’*’, ’+’]	[’sin’]	15	3000	1
[’*’, ’+’]	[’sin’]	15	2000	1
[’*’, ’+’]	[’sin’]	16	2000	1
[’*’, ’+’]	[’sin’]	16	4000	1
[’*’, ’+’]	[’sin’]	16	5000	1
[’*’, ’+’]	[]	25	10000	1

Table B.2: All model parameter combinations and the amount of runs for c_{max}

Binary Operators	Unary Operators	Max Size	Iterations	Runs
$['*','+','\wedge']$	[]	20	500	18
['*', '+']	[]	20	300	13
['*', '+']	[]	15	300	12
['*', '+']	[]	15	500	6
['*', '+']	[]	20	500	6
['*', '+']	['sin']	15	200	5
['*', '+']	['sin']	16	500	5
['*', '+', 'pow_int']	[]	20	500	4
['*', '+']	['sin']	15	300	4
['*', '+']	[]	12	300	4
['*', '+']	['sin']	15	100	3
$['*','+','\wedge']$	[]	20	300	2
['*', '+']	['sin']	16	300	2
['*', '+']	['sin']	16	200	2
$['*','+','\wedge']$	[]	25	500	2
['*', '+']	[]	16	300	2
['*', '+', 'pow_int']	[]	30	500	2
['*', '+']	['sin']	15	3000	1
['*', '+']	['sin']	15	2000	1
$['*','+','\wedge']$	[]	20	2000	1
['*', '+', 'pow_int']	[]	20	3000	1
['*', '+']	['sin']	16	1000	1
['*', '+']	['sin']	16	2000	1
['*', '+']	['sin']	16	5000	1
['*', '+']	['sin']	16	4000	1
['*', '+']	[]	20	7500	1
['*', '+']	[]	25	10000	1

Table B.3: t_{max} equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
3	$0.0361Z_3 + 0.1012$	$['*','+','\wedge','/']$	[]	25	10000	$2.14 \cdot 10^{-4}$	0.8697	0.8667
5	$0.0382Z_3 + 0.1001$	$['*','+','\wedge','/']$	['sin']	20	500	$2.02 \cdot 10^{-4}$	0.8763	0.8713
6	$(0.05356Z_3 + 0.31718)^2$	$['*','+']$	[]	20	500	$1.84 \cdot 10^{-4}$	0.88854	0.88254
7	$0.0361 \cdot 1.4084^{Z_1} + 0.0361Z_3 + 0.0641$	$['*','+','\wedge']$	['square']	20	500	$7.30 \cdot 10^{-5}$	0.9556	0.9486
9	$0.00995Z_1 + 0.1016(0.2240Z_3 + 1)^{1.55}$	$['*','+','\wedge']$	[]	20	500	$6.06 \cdot 10^{-5}$	0.9574	0.9484
11	$0.0671(0.3474 \cdot 1.3675^{Z_1} + 0.3474Z_3 + 1)^{1.36}$	$['*','+','\wedge']$	['sin']	20	500	$4.69 \cdot 10^{-5}$	0.9678	0.9568
13	$0.00407Z_7 + 0.0684(0.2230 \cdot 1.3379^{Z_1} + 0.2230Z_3 + 1)^{1.88}$	$['*','+','\wedge']$	['sin']	20	500	$4.05 \cdot 10^{-5}$	0.9741	0.9611
15	$0.0030Z_1(Z_1 + 3.522) + 0.0030Z_7 + 0.0030(Z_3 + 5.734)^{2.02}$	$['*','+',$'pow_int'$]$	['square']	25	500	$4.27 \cdot 10^{-5}$	0.9761	0.9611
17	$0.0979 \cdot 0.7441^{Z_3} + Z_1(0.0036Z_1 + 0.0110) + 0.0628Z_3$	$['*','+','\wedge']$	['sin']	20	500	$4.10 \cdot 10^{-5}$	0.9764	0.9594
19	$0.0037Z_7(0.5909^{Z_3} + Z_7) + 0.0661(0.2244 \cdot 1.3192^{Z_1} + 0.2244Z_3 + 1)^{1.93}$	$['*','+','\wedge']$	['sin']	20	500	$3.87 \cdot 10^{-5}$	0.9776	0.9586
21	$0.0031Z_1(Z_1 + 3.3527) + 0.0031Z_3(Z_3 + 10.9936) + 0.0031Z_7(Z_7 + 1.2513) + 0.0969$	$['*','+','\wedge']$	[]	25	500	$4.21 \cdot 10^{-5}$	0.9778	0.9568
23	$0.0029Z_7 + 0.0029(0.6229Z_4)^2 + 0.0029Z_7^2 + 0.0029(Z_1 + 1.7989)^2 + (0.0571Z_3 + 0.2942)^2$	$['*','+']$	['sin']	16	500	$4.00 \cdot 10^{-5}$	0.9799	0.9569
25	$0.0031Z_7(Z_7 + 3.0813) + 0.0032(Z_2 + 0.8425)^2 + (0.0573Z_3 + 0.2951)^2$	$['*','+']$	[]	20	500	$4.03 \cdot 10^{-5}$	0.9782	0.9566
27	$0.00307Z_1(Z_1 + 3.45392) + 0.00307Z_3(-0.68247Z_0 + Z_3 - 0.68247Z_7 + 11.3559) + 0.00307Z_7(Z_7 + 1.50977) + 0.096859$	$['*','+']$	[]	30	500	$2.26 \cdot 10^{-5}$	0.9866	0.9596

Table B.4: c_{max} equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
3	$0.01910Z_4 + 0.02235$	['*','+']	[]	25	10000	$4.52 \cdot 10^{-5}$	0.87927	0.87627
5	$0.02060^{(0.82713^{Z_4})}$	['*','+','∧']	[]	20	500	$3.85 \cdot 10^{-5}$	0.90011	0.89511
6	$(0.06140Z_4 + 0.14141)^2$	['*','+','∧']	['square']	20	500	$3.38 \cdot 10^{-5}$	0.91249	0.90649
7	$0.02024 \cdot 1.20528^{Z_3} + 0.02024Z_4$	['*','+']	[]	20	500	$2.78 \cdot 10^{-5}$	0.91825	0.91125
9	$0.00391Z_3 + 0.01844(0.45826Z_4 + 1)^{1.98594}$	['*','+','∧','/']	['sin']	20	500	$1.95 \cdot 10^{-5}$	0.94907	0.94007
10	$(0.01105Z_3 + 0.06030Z_4 + 0.13624)^2$	['*','+']	['square']	20	500	$1.80 \cdot 10^{-5}$	0.95385	0.94385
11	$0.01815(0.11168Z_3 + 0.60803Z_4 + 1)^{1.61910}$	['*','+','∧','/']	['sin']	20	500	$1.62 \cdot 10^{-5}$	0.95816	0.94716
15	$(1.22078^{Z_3} + Z_4)(0.00308Z_4 + 0.01467) + 0.00360$	['*','+','∧']	[]	25	500	$1.64 \cdot 10^{-5}$	0.95855	0.94355
17	$(Z_4 + 5.74740)(0.00026Z_3(Z_3 + 1.35592) + 0.00259Z_4 + 0.00310)$	['*','+','pow_int']	[]	20	500	$1.74 \cdot 10^{-5}$	0.95982	0.94282
19	$(Z_4 + 3.17680)(Z_3(0.00044Z_3 + 0.00057) + 0.00252Z_4 + 0.00961) - 0.01249$	['*','+']	[]	20	500	$1.55 \cdot 10^{-5}$	0.96365	0.94465
21	$(Z_4 + 5.11469)(0.00278Z_4 + (-0.00014Z_0 + 0.00029Z_3)(Z_3 + 1.50388) + 0.00352)$	['*','+']	[]	25	500	$1.43 \cdot 10^{-5}$	0.96684	0.94584
23	$(Z_4 + 5.76562)(0.00261Z_4 + 0.00026(-0.46745Z_0 + Z_3)(Z_3 + (0.62241)^{Z_1}) + 0.00315)$	['*','+','pow_int']	[]	20	500	$1.33 \cdot 10^{-5}$	0.96807	0.94507
25	$(0.00147 - 6.72 \cdot 10^{-5}Z_0)(Z_1^2 + Z_3^2 + (Z_4 + 1.07845)(Z_3 + 1.53120Z_4 + 10.83939))$	['*','+']	[]	20	7500	$1.39 \cdot 10^{-5}$	0.96915	0.94415
29	$-0.00179Z_0 + (0.00237Z_4 + 0.01808) \cdot (0.06863Z_1(Z_1 - 1.06185Z_7) + 0.06863Z_3(Z_3 + 2.05184) + Z_4) + 0.01754$	['*','+']	[]	30	500	$1.29 \cdot 10^{-5}$	0.97102	0.94202

B.2. Parameterization SR runs

Details on the PySR model congiurations for parameterization equations for thickness and camber are included in Table B.5 and Table B.6. Note that due to much bigger datasets and higher required complexity, far fewer runs have been conducted here than for the feature equations.

The equations with the highest R^2 per complexity are shown in Table B.7 and Table B.8. Once again, despite only a few runs using sin, this seems to be included in almost every best equation per complexity. This once again puts more trust in the tuned PySR settings. The final equations from all these runs, resembling the final results of latent analysis, are tested in the validation plot shown in figure B.1.

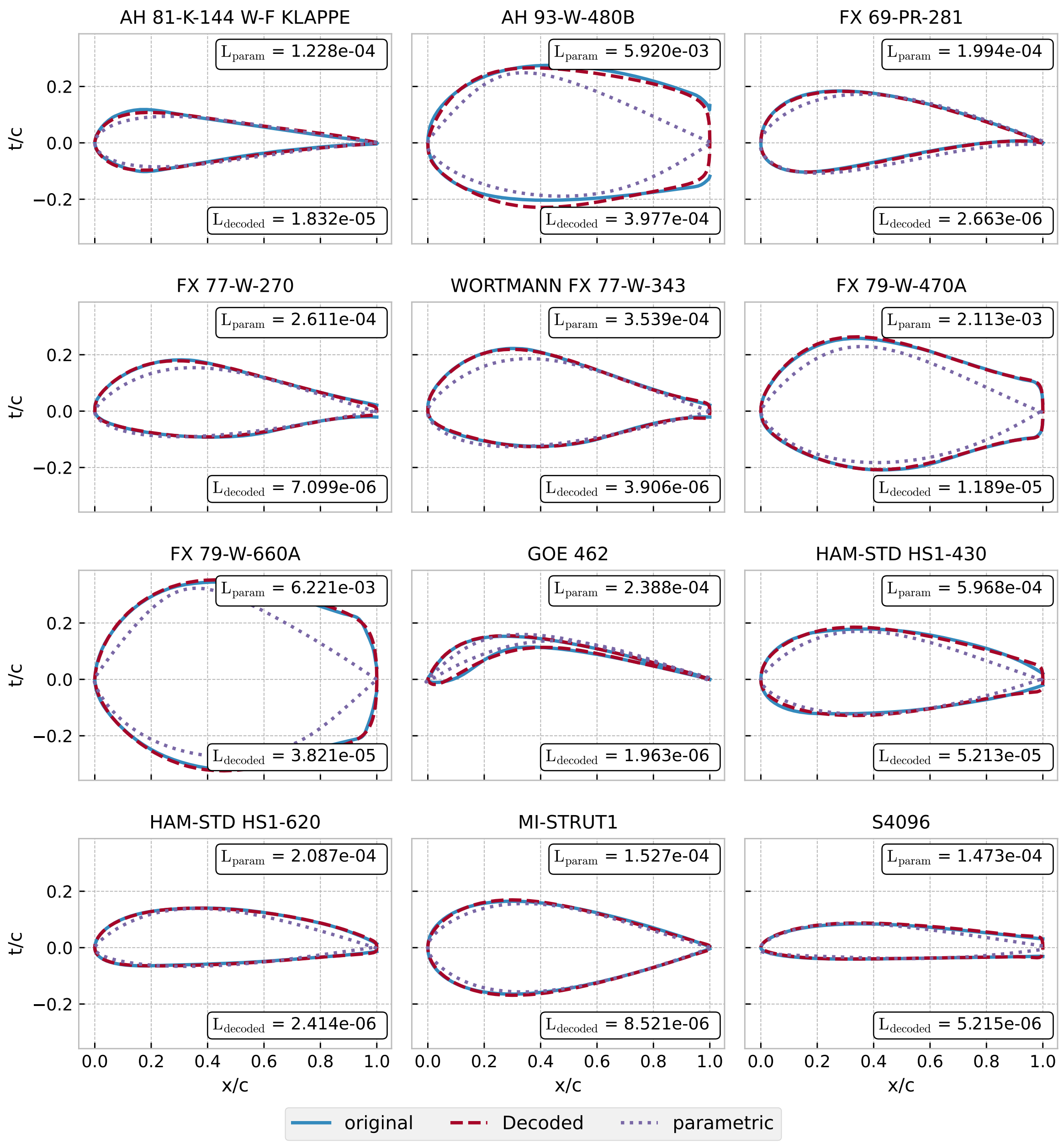


Figure B.1: Validation plots showing the reconstruction of the 12 pre-selected airfoils. Decoder reconstruction loss $L_{decoded}$ and parametric reconstruction loss L_{param} values are both marked in the corners of each plot. (adapted from [4])

Table B.5: All model parameters combinations and the amount of runs for thickness distribution, sorted by most runs

Binary Operators	Unary Operators	Max Size	Iterations	Runs
$['*','+']$	[]	30	500	6
$['*','+','\wedge']$	[]	50	10	4
$['*','+']$	[]	35	500	4
$['*','+']$	[]	30	1000	3
$['*','+']$	[]	40	50	3
$['*','+','/','\wedge']$	[]	40	500	2
$['*','+']$	[]	15	500	2
$['*','+']$	[]	30	200	2
$['*','+']$	[]	30	2000	2
$['*','+']$	[]	14	1000	2
$['*','+','\wedge']$	[]	50	500	2
$['*','+','\wedge','/']$	[]	40	10000	2
$['*','+','/','\wedge']$	[]	35	500	1
$['*','+','/']$	[]	30	2000	1
$['*','+','\wedge','/']$	[]	30	500	1
$['*','+','/']$	[]	45	500	1
$['*','+','/']$	[]	40	500	1
$['*','+','\wedge']$	[]	40	500	1
$['*','+','\wedge']$	['sin']	50	500	1
$['*','+','\wedge','/']$	[]	35	500	1
$['*','+','\wedge','/']$	[]	40	5000	1
$['*','+','\wedge']$	['sin']	45	5000	1
$['*','+']$	[]	14	500	1
$['*','+']$	['sin']	30	500	1
$['*','+','\wedge']$	[]	60	750	1
$['*','+','\wedge']$	[]	50	200	1
$['*','+','\wedge']$	[]	45	500	1
$['*','+','\wedge']$	[]	30	500	1
$['*','+','\wedge']$	[]	30	2000	1
$['*','+']$	[]	25	500	1
$['*','+']$	[]	30	700	1
$['*','+']$	[]	30	400	1
$['*','+']$	[]	25	1000	1
$['*','+']$	[]	25	5000	1
$['*','+']$	[]	35	2000	1
$['*','+']$	[]	40	1000	1
$['*','+']$	[]	45	500	1
$['*','+']$	[]	45	1000	1

Table B.6: All model parameters combinations and the amount of runs for camber distribution, sorted by most runs

Binary Operators	Unary Operators	Max Size	Iterations	Runs
$['*','+','\wedge']$	[]	50	10	8
$['*','+']$	[]	30	500	5
$['*','+']$	[]	35	500	4
$['*','+']$	[]	30	1000	3
$['*','+']$	[]	30	2000	2
$['*','+','\wedge']$	[]	50	200	2
$['*','+']$	[]	30	200	2
$['*','+']$	[]	25	500	2
$['*','+','\wedge']$	['sin']	50	500	2
$['*','+','\wedge']$	[]	50	500	2
$['*','+','/','\wedge']$	[]	35	500	1
$['*','+','/']$	[]	30	2000	1
$['*','+','/','\wedge']$	[]	40	500	1
$['*','+','\wedge']$	[]	20	200	1
$['*','+','\wedge']$	['sin']	45	5000	1
$['*','+','\wedge','/']$	[]	35	500	1
$['*','+','/']$	[]	40	500	1
$['*','+','\wedge','/']$	[]	30	500	1
$['*','+','/']$	[]	45	500	1
$['*','+','\wedge']$	[]	60	750	1
$['*','+','\wedge']$	[]	30	2000	1
$['*','+','\wedge']$	[]	40	500	1
$['*','+','\wedge']$	[]	45	500	1
$['*','+','\wedge']$	[]	30	500	1
$['*','+','\wedge','/']$	[]	40	10000	1
$['*','+']$	[]	25	1000	1
$['*','+']$	['sin']	30	500	1
$['*','+']$	[]	30	400	1
$['*','+']$	[]	25	5000	1
$['*','+']$	[]	14	1000	1
$['*','+']$	[]	30	700	1
$['*','+']$	[]	35	2000	1
$['*','+']$	[]	40	1000	1
$['*','+']$	[]	45	500	1
$['*','+']$	[]	45	1000	1

Table B.7: Thickness distribution equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
4	$0.04513\sin(4.44519X) + 0.06028$	$['*','+','\wedge']$	['sin']	50	500	$1.11 \cdot 10^{-3}$	0.43785	0.43385
6	$X(X(X - 1.90158) + 0.92129)$	['*', '+']	[]	35	500	$9.52 \cdot 10^{-4}$	0.50974	0.50374
7	$X(X^{-0.27105} - 0.99939)$	$['*','+','\wedge']$	[]	30	2000	$8.25 \cdot 10^{-4}$	0.58875	0.58175
8	$(0.46532 - 0.48608X) \cdot (X + 0.09856Z_3)$	['*', '+']	[]	35	500	$6.00 \cdot 10^{-4}$	0.73661	0.72861
9	$(X - 0.97452) \cdot (-0.20560XZ_3 - 0.20560X - 0.06865)$	['*', '+']	[]	35	500	$3.76 \cdot 10^{-4}$	0.82485	0.81585
10	$-0.17831(X - 0.96899) \cdot (X(Z_3 + 1.48871) + 0.34277)$	['*', '+']	[]	35	500	$3.22 \cdot 10^{-4}$	0.85295	0.84295
11	$(X - 1.00219X^{1.10087}) \cdot (Z_3 + 2.97219)$	$['*','+','/','\wedge']$	[]	40	500	$2.22 \cdot 10^{-4}$	0.89787	0.88687
12	$(0.04513Z_3 + 0.09118) \cdot \sin(3.19122X^{0.62286}) + 0.00816$	$['*','+','\wedge']$	['sin']	50	500	$1.16 \cdot 10^{-4}$	0.94073	0.92873
15	$0.08328(-X^{0.46824}(X - 1.00226) \cdot (Z_3 + 2.92917))^{1.08854}$	$['*','+','/','\wedge']$	[]	35	500	$1.17 \cdot 10^{-4}$	0.94426	0.92926
16	$(0.04513Z_3(X + 0.47150) + 0.09750) \cdot \sin(3.15972X^{0.56732}) + 0.00443$	$['*','+','\wedge']$	['sin']	50	500	$1.16 \cdot 10^{-4}$	0.94313	0.92713
17	$X^{0.50525}(0.02816 - 0.02810X) \cdot (Z_1 + 9.13612(0.22872Z_3 + 1)^{1.49958})$	$['*','+','\wedge']$	[]	30	2000	$8.68 \cdot 10^{-5}$	0.96231	0.94531
19	$X^{0.48630}((-0.25614 \cdot 1.07499^{Z_1} - 0.09239Z_3) \cdot (X - 1.00745) - 0.00431)$	$['*','+','\wedge']$	[]	45	500	$9.20 \cdot 10^{-5}$	0.95909	0.94009
20	$(-0.33708 \cdot 1.08587^{Z_1}X^{0.49642} - 0.21621XZ_3) \cdot (\sin(X) - 0.84290)$	$['*','+','\wedge']$	['sin']	45	5000	$8.00 \cdot 10^{-5}$	0.96362	0.94362
21	$(0.23635 - 0.23751X) \cdot (0.42815^{X}XZ_3 + 1.08394^{Z_1}X^{0.43348})$	$['*','+','\wedge','/']$	[]	30	500	$7.88 \cdot 10^{-5}$	0.96534	0.94434
23	$(0.04513Z_3(X + 0.49273) + 0.01396\sin(Z_1 - 0.53024) + 0.10220) \cdot \sin(3.16893X^{0.57226}) + 0.00528$	$['*','+','\wedge']$	['sin']	50	500	$7.52 \cdot 10^{-5}$	0.96650	0.94350
24	$-0.00641Z_1 + (-0.33475 \cdot 1.18331^{Z_1}X^{0.49542} - 0.21821XZ_3) \cdot (\sin(X) - 0.84226)$	$['*','+','\wedge']$	['sin']	45	5000	$7.25 \cdot 10^{-5}$	0.96772	0.94372

Table B.7: (Continued) Thickness distribution equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
25	$-0.11295X^{0.61969}(X - 0.98719) \cdot (1.68729^{Z_1}(0.94594 - 1.43798X) + Z_3 + 2.06696) + 0.00554$	$['*',' +',' \wedge']$	[]	50	500	$6.73 \cdot 10^{-5}$	0.96943	0.94443
27	$(0.01447Z_1 + 0.04513Z_3(X + 0.50131) + 0.09783) \cdot \sin(3.16404X^{0.56907}) - 0.04513\sin(3.16115X^{0.57049}) + 0.00573$	$['*',' +',' \wedge']$	['sin']	50	500	$6.68 \cdot 10^{-5}$	0.97012	0.94412
28	$(X^{0.48630}(0.25614 \cdot 1.07499^{Z_1} - 0.09239Z_3) \cdot (X - 1.00745) + 0.00431)$	$['*',' +',' \wedge']$	[]	40	3000	$5.60 \cdot 10^{-5}$	0.97060	0.94460
31	$(0.04513Z_3(X + 0.49183) - 0.03932Z_7(X - 0.38397) + 0.01716\sin(Z_1 - 0.92789) + 0.10885) \cdot \sin(3.15651X^{0.56618}) + 0.00369$	$['*',' +',' \wedge']$	['sin']	50	500	$5.24 \cdot 10^{-5}$	0.97605	0.94505
33	$0.02953X^{0.62443}(3.82574 - 3.85811X) \cdot (Z_3 + (X - 0.10603Z_1 - 0.43068) \cdot (-0.76006Z_1 - 0.76006Z_7 - 1.70447) + 2.34839) + 0.00503$	$['*',' +',' \wedge']$	[]	50	500	$4.68 \cdot 10^{-5}$	0.97761	0.94461
35	$(0.01427Z_1 + 0.04513Z_3(X + 0.49839) - 0.03853Z_7(X - 0.37675) + 0.09788) \cdot \sin(3.1632X^{0.56842}) - 0.04513\sin(0.35014Z_1 + 1.29446) + 0.04616$	$['*',' +',' \wedge']$	['sin']	50	500	$4.48 \cdot 10^{-5}$	0.97915	0.94415
37	$(0.01434Z_1 + 0.05075Z_3(X + 0.40663) - 0.03823Z_7(X - 0.37722) + 0.09778) \cdot \sin(3.1648X^{0.56468}) - 0.04513\sin(0.35211Z_1 + 1.29375) + 0.04620$	$['*',' +',' \wedge']$	['sin']	50	500	$4.48 \cdot 10^{-5}$	0.97930	0.94230
38	$(-0.00797Z_1 + (X - 0.99582) \cdot (-0.27091 \cdot 1.15125^{Z_1} - 0.14412X^{0.30390}Z_3 - 0.14412(0.13793 - 0.35178X) \cdot (Z_4 + 2Z_7))) \cdot \sin(X^{0.49471})$	$['*',' +',' \wedge']$	['sin']	45	5000	$4.21 \cdot 10^{-5}$	0.98049	0.94249

Table B.7: (Continued) Thickness distribution equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
39	$(-0.00799Z_1 + (X - 0.99625) \cdot (-0.27069 \cdot 1.15150^{Z_1} - 0.14709Z_3 \sin^{0.31346}(X) - 0.14709(0.13534 - 0.34500X) \cdot (Z_4 + 2Z_7))) \cdot \sin(X^{0.49470})$	$['*','+','\wedge']$	['sin']	45	5000	$4.19 \cdot 10^{-5}$	0.98059	0.94159
40	$-0.00499Z_1 + (X - 0.99586) \cdot (-0.27117 \cdot 1.15479^{Z_1} - 0.14277X^{0.29714}Z_3 - 0.14277(0.13345 - 0.34376X) \cdot (Z_1 + Z_4 + 2Z_7)) \cdot \sin(X^{0.49508})$	$['*','+','\wedge']$	['sin']	45	5000	$4.02 \cdot 10^{-5}$	0.98146	0.94146
42	$(0.00283Z_1(Z_1 - 2.41730) + (X - 0.99082) \cdot (-0.26930 \cdot 1.14281^{Z_1} - 0.14921X^{0.31959}Z_3 - 0.14921(0.13570 - 0.34653X) \cdot (Z_4 + 2Z_7))) \cdot \sin(X^{0.49269})$	$['*','+','\wedge']$	['sin']	45	5000	$3.98 \cdot 10^{-5}$	0.98214	0.94014
43	$(X - 1.00585) \cdot (-0.27131 \cdot 1.13726^{Z_1} - 0.14012X^{0.29918}Z_3 - 0.14012(0.13058 - 0.32988X) \cdot (Z_1 + Z_4 + 2Z_7)) \cdot \sin(X^{0.48217}) - 0.00734\sin(Z_1 + 0.53794)$	$['*','+','\wedge']$	['sin']	45	5000	$3.73 \cdot 10^{-5}$	0.98298	0.93998
44	$Z_1(0.00216Z_1 - 0.00367) + (X - 0.99134) \cdot (-0.27016 \cdot 1.13355^{Z_1} - 0.14665X^{0.29689}Z_3 - 0.14665(0.13718 - 0.34668X) \cdot (Z_1 + Z_4 + 2Z_7)) \cdot \sin(X^{0.50051})$	$['*','+','\wedge']$	['sin']	45	5000	$3.66 \cdot 10^{-5}$	0.98368	0.93968
47	$(0.06546X^{0.44514}Z_3 + 0.04513X + 0.01282Z_1 - 0.04513(0.38830X - 0.14944) \cdot (Z_0 + Z_1 + Z_4 + 2Z_7) + 0.08444) \cdot \sin(3.1533X^{0.51252}) - 0.00836\sin(Z_1 + 0.83747) + 0.00692$	$['*','+','\wedge']$	['sin']	50	5000	$3.702 \cdot 10^{-5}$	0.98321	0.93821

Table B.7: (Continued) Thickness distribution equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
48	$(0.06586X^{0.45006}Z_3 + 0.04513X + 0.01348Z_1 - 0.04513(0.40915X - 0.15574) \cdot (Z_0 + Z_4 + 2Z_7 + \sin(Z_1)) + 0.08438) \cdot \sin(3.1538X^{0.51302}) - 0.00861\sin(Z_1 + 0.81455) + 0.00702$	$['*','+','\wedge']$	['sin']	50	500	$3.695 \cdot 10^{-5}$	0.98293	0.93493
49	$(0.0659X^{0.44895}Z_3 + 0.04513X + 0.01437Z_1 - 0.04513(0.42128X - 0.15937) \cdot (XZ_1 + Z_0 + Z_4 + 2Z_7) + 0.08430) \cdot \sin(3.1544X^{0.51370}) - 0.00845\sin(Z_1 + 0.85049) + 0.00717$	$['*','+','\wedge']$	['sin']	50	500	$3.61 \cdot 10^{-5}$	0.98331	0.93431
50	$(0.06583X^{0.44769}Z_3 + 0.04513X + 0.01430Z_1 - 0.04513(0.43911X - 0.16550) \cdot (XZ_1 + Z_4 + 2Z_7 + \sin(Z_0)) + 0.08439) \cdot \sin(3.1540X^{0.51341}) - 0.00841\sin(Z_1 + 0.85852) + 0.00713$	$['*','+','\wedge']$	['sin']	50	500	$3.56 \cdot 10^{-5}$	0.98344	0.93344
53	$-0.10897X + 0.00247Z_1(Z_1 - 1.42704) + (0.04513X^{0.29232} - 0.04503) \cdot (-9.3398X(Z_3 + 1.54301) + 2.48941) + (0.04513X^{0.44318} - 0.04658X) \cdot (Z_1 + (0.88426 - 2.43865X) \cdot (X + Z_4 + Z_7)) + 0.10539$	$['*','+','\wedge']$	[]	60	750	$4.38 \cdot 10^{-5}$	0.98018	0.92718
55	$-0.11002X + 0.00248Z_1(Z_1 - 1.49992) + (0.04513X^{0.28825} - 0.04504) \cdot (-9.4238X(Z_3 + 1.52040) + 2.53662) + (0.04513X^{0.43961} - 0.04625X) \cdot (Z_1 + (1.06894 - 2.95973X) \cdot (X + 0.61004Z_4 + Z_7)) + 0.10722$	$['*','+','\wedge']$	[]	60	750	$4.02 \cdot 10^{-5}$	0.98164	0.92664

Table B.7: (Continued) Thickness distribution equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
57	$-0.10946X + 0.00240Z_1(Z_1 - 1.51297) + (0.04513X^{0.29241} - 0.04503)\cdot(-9.2711X(Z_3 + 1.55513) + 2.49715) + (0.04513X^{0.43290} - 0.04666X)\cdot(Z_1 + (1.00312 - 2.76535X)\cdot(X + 0.55452Z_0 + 0.55452Z_4 + Z_7)) + 0.10592$	$['*','+','\wedge']$	[]	60	750	$3.69 \cdot 10^{-5}$	0.98300	0.92600
59	$-0.10723X + 0.00242Z_1(Z_1 - 1.56209) + (0.04513X^{0.29512} - 0.04503)\cdot(-9.2087X(Z_3 + 1.57982) + 2.43598) + (0.04513X^{0.42534} - 0.04658X)\cdot(Z_1 + (0.94917 - 2.59032X)\cdot(1.35159X + 0.55708Z_0 + 0.55708Z_4 + Z_7)) + 0.10390$	$['*','+','\wedge']$	[]	60	750	$3.67 \cdot 10^{-5}$	0.98280	0.92380

Table B.8: Camber distribution equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
2	$0.01553 \sin(Z_4) + 0.01549$	$[*, +, \wedge]$	[sin]	50	500	$2.20 \cdot 10^{-4}$	0.08221	0.08021
3	$0.01553Z_4 + 0.00687$	$[*, +, \wedge]$	[]	50	10	$2.15 \cdot 10^{-4}$	0.07196	0.06896
4	$0.01292Z_4 + 0.01163$	$[*, +]$	[]	30	200	$1.88 \cdot 10^{-4}$	0.20622	0.20222
6	$0.01553Z_4 \sin(3.53074X) + 0.01549$	$[*, +, \wedge]$	[sin]	50	500	$1.12 \cdot 10^{-4}$	0.49753	0.49153
7	$(0.08957 - 0.09228X)(XZ_4 + X)$	$[*, +]$	[]	35	500	$3.31 \cdot 10^{-5}$	0.85059	0.84359
9	$(0.08252 - 0.08459X)(X + Z_4(X + 0.05050))$	$[*, +]$	[]	35	500	$3.20 \cdot 10^{-5}$	0.85989	0.85089
10	$(XZ_4 + X)(X(0.06457X - 0.17840) + 0.11424)$	$[*, +]$	[]	35	500	$2.82 \cdot 10^{-5}$	0.86691	0.85691
13	$(-0.04813 + 0.04806X^{-X})(X + Z_4 + 0.50799)$	$[*, +, \wedge, /]$	[]	30	500	$2.87 \cdot 10^{-5}$	0.87833	0.86533
14	$X(0.06952X - 0.06993)(X - 1.76747)(0.18537Z_3 + Z_4 + 0.77256)$	$[*, +]$	[]	35	500	$2.32 \cdot 10^{-5}$	0.88513	0.87113
16	$0.02055(Z_4 + (X + 0.57832)^{Z_1}) \sin(3.11701X^{0.82955})$	$[*, +, \wedge]$	[sin]	45	5000	$2.03 \cdot 10^{-5}$	0.91131	0.89531
17	$(0.14804X - 0.14816X^{1.45117})(Z_4 + (X + 0.57300)^{Z_1})$	$[*, +, \wedge]$	[]	30	500	$2.02 \cdot 10^{-5}$	0.91416	0.89716
19	$(0.15886X - 0.15888X^{1.41255})(Z_4 + ((2X)^X)^{Z_1})$	$[*, +, \wedge]$	[]	30	500	$2.01 \cdot 10^{-5}$	0.91684	0.89784
21	$X(-0.12403 + \frac{0.26207}{X+1.10722})(1.13206^{Z_3} + Z_1(X - 0.43487) + Z_4)$	$[*, +, \wedge, /]$	[]	40	10000	$1.59 \cdot 10^{-5}$	0.92527	0.90427
22	$(0.02229Z_1(X - 0.43063) + 0.00309Z_3 + 0.02229Z_4 + 0.01811) \sin(3.11282X^{0.81369})$	$[*, +, \wedge]$	[sin]	45	5000	$1.47 \cdot 10^{-5}$	0.93179	0.90979
23	$X(-0.13360 + \frac{0.27675}{X+1.06545})(0.79712 \cdot 1.19150^{Z_3} + Z_1(X - 0.43223) + Z_4)$	$[*, +, \wedge, /]$	[]	40	10000	$1.41 \cdot 10^{-5}$	0.93507	0.91207
24	$-0.03703X(X - 0.99827)(XZ_1 + 2Z_4 + (X - 0.70408)(Z_1 - 1.88818Z_3 - 1.88818Z_4) + 2.00040)$	$[*, +]$	[]	30	200	$1.33 \cdot 10^{-5}$	0.93442	0.91042

Table B.8: (Continued) Camber distribution equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
26	$-0.03693X(X-0.99805)(XZ_1+2Z_4+(X-0.69651)(2Z_1-1.88891Z_3-1.88891Z_4)+1.99915))$	$['*','+']$	[]	30	200	$1.10\cdot10^{-5}$	0.946600	0.920600
28	$X(X-0.99877)(-0.03680XZ_1-0.07360Z_4-0.03680(X-0.68599)(XZ_0+2Z_1-1.89102Z_3-1.89102Z_4)-0.07587))$	$['*','+']$	[]	30	200	$1.03\cdot10^{-5}$	0.950090	0.922090
29	$-0.09087X(X-1.00280)(0.16138Z_3+Z_4+(X-0.43615)(Z_1-0.71789Z_3-0.71789Z_4-0.71789Z_7)+0.79520)$	$['*','+','/','\wedge']$	[]	40	500	$8.89\cdot10^{-6}$	0.958770	0.929770
31	$X(0.09246-0.09186X)(0.15483Z_3+Z_4+(X-0.43497)(Z_1-0.55308Z_3-0.89831Z_4-0.55308Z_7)+0.77278)$	$['*','+','/']$	[]	45	500	$8.49\cdot10^{-6}$	0.960990	0.930990
33	$X(0.09186-0.09123X)(0.15880Z_3+Z_4+(X-0.44243)(-0.65648XZ_7+Z_1-0.65648Z_3-0.88627Z_4)+0.77497)$	$['*','+','/']$	[]	45	500	$8.55\cdot10^{-6}$	0.962000	0.929000
35	$X(0.09169-0.09131X)(0.15333Z_3+Z_4+(X-0.43376)(X+Z_1-0.50300Z_3-0.72358Z_4-0.50300Z_7-0.82929)+0.76258)$	$['*','+','/']$	[]	45	500	$8.11\cdot10^{-6}$	0.964600	0.929600
37	$X(0.07655-0.07622X)(0.10824Z_1+0.10824Z_3+0.10824Z_4^2+Z_4+(X-0.53640)(-0.92395XZ_7+Z_1-0.92395Z_3-0.92395Z_4)+0.90039)$	$['*','+','/']$	[]	40	500	$7.65\cdot10^{-6}$	0.965610	0.928610
39	$0.13071\cdot1.12288^{Z_4}X(0.24634^X-0.24599)(1.23583^{Z_3}+2.13784^X(X-0.45135)(-0.69675X(Z_3+Z_7-0.91956)+Z_1)+Z_4)$	$['*','+','\wedge']$	['sin']	45	5000	$6.95\cdot10^{-6}$	0.971200	0.932200

Table B.8: (Continued) Camber distribution equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
40	$X \sin((1.12508^{Z_4}(0.13090 \cdot 0.24558^{X} - 0.03211)(1.23545^{Z_3} + 2.13785^{X}(X - 0.45190)(-0.69668X(Z_3 + Z_7 - 0.90922) + Z_1)))$	$['*', '+', '\wedge']$	$['\sin']$	45	5000	$6.81 \cdot 10^{-6}$	0.971490	0.931490
43	$1.12505^{Z_4}X(0.13046 \cdot 0.24047^{X} - 0.03136)(1.23934^{Z_3} + 2.13684^{X}(X - 0.45812)(X(0.33166Z_0 - 0.72332Z_3 - 0.72332Z_7 + 0.64854) + Z_1) + Z_4)$	$['*', '+', '\wedge']$	$['\sin']$	45	5000	$6.64 \cdot 10^{-6}$	0.973110	0.930110
44	$X \sin((0.13046 \cdot 1.12505^{Z_4} \cdot (0.24047^{X} - 0.24039)(1.23934^{Z_3} + 2.13684^{X}(X - 0.45812)(X(0.35032Z_0 - 0.76401Z_3 - 0.76401Z_7 + 0.64854) + Z_1) + Z_4)))$	$['*', '+', '\wedge']$	$['\sin']$	45	5000	$6.42 \cdot 10^{-6}$	0.973910	0.929910
47	$0.01482 \cdot 0.43343^{X} \cdot 1.11498^{Z_4}(6.83047 - 6.79799X)(X(0.21442Z_3 + Z_4 + (X - 0.45750)(X(Z_0 + Z_1 - 1.19099Z_3 - 1.19099Z_7 + 1.04871) + Z_1)) + X) - 0.00015$	$['*', '+', '\wedge']$	$[]$	50	200	$5.93 \cdot 10^{-6}$	0.974340	0.927340
49	$0.01486 \cdot 0.43646^{X} \cdot 1.11303^{Z_4}(6.83256 - 6.79622X)(X(0.21485Z_3 + Z_4 + (X - 0.45017)(X(X(Z_0 + Z_1) - 1.16790Z_3 - 1.16790Z_7 + 1.03194) + Z_1)) + X) - 0.00025$	$['*', '+', '\wedge']$	$[]$	50	200	$5.92 \cdot 10^{-6}$	0.973960	0.924960
53	$0.01553X(1.15676 - 1.15312X)(-1.00351XZ_7 + (2.67575 - 3.69588X)(X(Z_7 - 1.79210) + 0.78695Z_3 + Z_4 + 0.79011) + (Z_4 + 4.79172)(Z_1(X - 0.44445) + 0.49640Z_4 + 1.09238) - 1.49873) - 0.00024$	$['*', '+', '\wedge']$	$[]$	60	750	$5.86 \cdot 10^{-6}$	0.975140	0.922140

Table B.8: (Continued) Camber distribution equations with the highest R^2 per complexity where equations that performed worse than the previous row are filtered out, including model parameters

Size	Equation	Binary	Unary	Max Size	Iterations	MSE Avg.	R^2 Avg.	C. Score
55	$0.01553X(1.11517 - 1.11249X)(-0.85206XZ_7 + (2.44339 - 3.58971X)(X(-1.00840Z_0 + Z_7 - 1.60606) + Z_3 + Z_4 + 0.84007) + (Z_4 + 4.98321)(Z_1(X - 0.44816) + 0.54392Z_4 + 1.09189) - 1.65148) - 0.00013$	$['*', '+', '\wedge']$	[]	60	750	$5.22 \cdot 10^{-6}$	0.978210	0.923210
57	$0.01553X(1.16468 - 1.16044X)(-0.90129XZ_7 + (2.62793 - 3.76898X)(X(-0.90212Z_0 + Z_7 - 1.67332) + 0.81735Z_3 + Z_4 + 0.80417) + (Z_4 + 4.82455)(Z_1(X - 0.45010) + 0.50499Z_4 + 1.08920) - 1.55580) - 0.00031$	$['*', '+', '\wedge']$	[]	60	750	$5.07 \cdot 10^{-6}$	0.979310	0.922310
59	$0.01553X(1.16254 - 1.15956X)(-0.90316XZ_7 + (2.66484 - 3.80325X)(X(-0.88327Z_0 + Z_7 - 1.85512) + Z_3(X + 0.51648) + Z_4 + 0.82437) + (Z_4 + 4.83158)(Z_1(X - 0.44830) + 0.50278Z_4 + 1.09133) - 1.57776) - 0.00017$	$['*', '+', '\wedge']$	[]	60	750	$4.99 \cdot 10^{-6}$	0.979390	0.920390

B.3. Extra plot for chosen thickness distribution equation

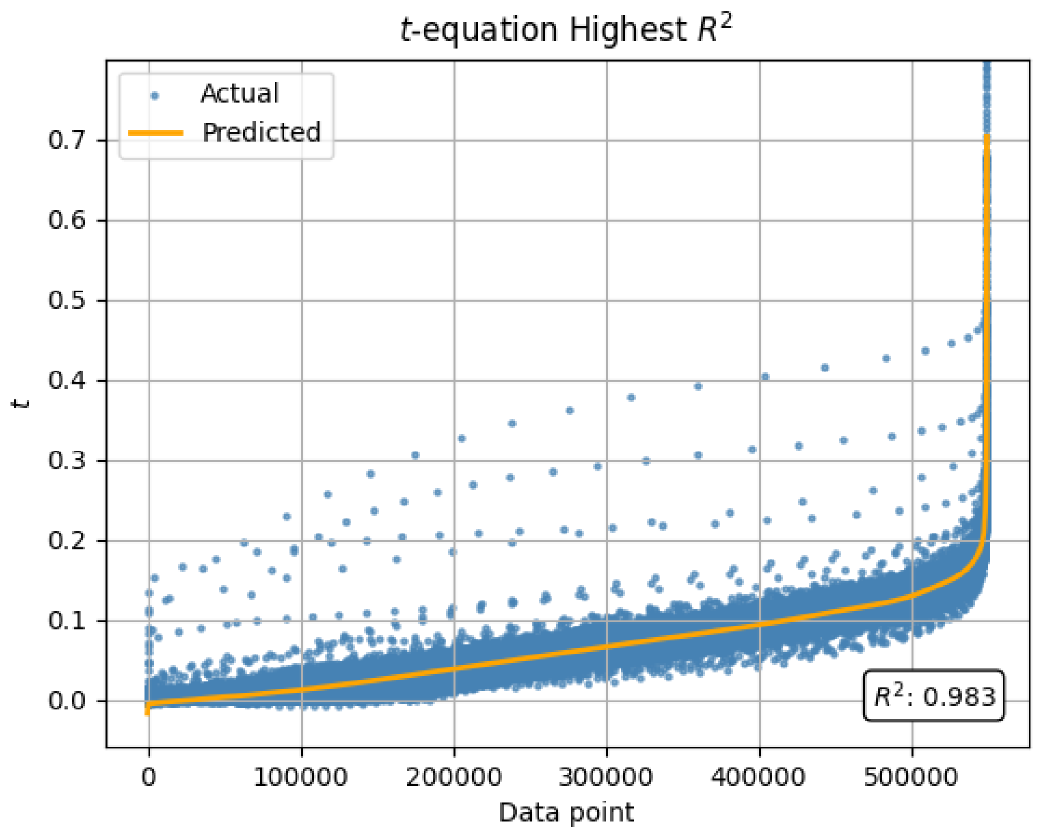


Figure B.2: $Z_1(0.00216Z_1 - 0.00367) + (X - 0.99134) \cdot (-0.270161.13355^{Z_1} - 0.14665X^{0.29689}Z_3 - 0.14665(0.13718 - 0.34668X) \cdot (Z_1 + Z_4 + 2Z_7)) \cdot \sin(X^{0.50051})$

C. Summary of trained SR-VAE models

This chapter contains the top trained SR-VAE models used for ranking. For reference, the VAE from Swannet et al. [4] is included as well.

C.1. Ranking based on MSE

Table C.1 contains part of the performance metrics used to assess the best SR-VAE. Note that while not apparent from the table, the MSE losses shown below are averages of both decoded losses and parametric losses, across both the UIUC and AFBench dataset.

Table C.1: Average MSE of best trained SR-VAE models tested with UIUC and AFBench datasets for both decoder and SR equations ranked from lowest to highest overall average

γ	β	**SR training**	**Outlier Average**	**Datasets Average**	**Overall Average**
1	$3.072 \cdot 10^{-5}$	per batch	$1.943 \cdot 10^{-4}$	$9.084 \cdot 10^{-6}$	$1.017 \cdot 10^{-4}$
1	$2.154 \cdot 10^{-5}$	fixed	$2.186 \cdot 10^{-4}$	$8.801 \cdot 10^{-6}$	$1.137 \cdot 10^{-4}$
1	$2.154 \cdot 10^{-5}$	fixed	$2.200 \cdot 10^{-4}$	$8.784 \cdot 10^{-6}$	$1.144 \cdot 10^{-4}$
1	$3.072 \cdot 10^{-5}$	per batch	$2.284 \cdot 10^{-4}$	$8.996 \cdot 10^{-6}$	$1.187 \cdot 10^{-4}$
1	$3.072 \cdot 10^{-5}$	per batch	$2.302 \cdot 10^{-4}$	$9.567 \cdot 10^{-6}$	$1.199 \cdot 10^{-4}$
1	$2.154 \cdot 10^{-5}$	per epoch	$2.423 \cdot 10^{-4}$	$9.370 \cdot 10^{-6}$	$1.259 \cdot 10^{-4}$
1	$3.072 \cdot 10^{-5}$	per batch	$2.435 \cdot 10^{-4}$	$9.282 \cdot 10^{-6}$	$1.264 \cdot 10^{-4}$
1	$2.154 \cdot 10^{-5}$	per batch	$2.505 \cdot 10^{-4}$	$8.893 \cdot 10^{-6}$	$1.297 \cdot 10^{-4}$
0	$2.154 \cdot 10^{-5}$	per batch	$2.502 \cdot 10^{-4}$	$9.689 \cdot 10^{-6}$	$1.300 \cdot 10^{-4}$
1	$2.154 \cdot 10^{-5}$	per batch	$2.592 \cdot 10^{-4}$	$1.095 \cdot 10^{-5}$	$1.351 \cdot 10^{-4}$
1	$3.072 \cdot 10^{-5}$	per epoch	$2.600 \cdot 10^{-4}$	$1.038 \cdot 10^{-5}$	$1.352 \cdot 10^{-4}$
1	$3.072 \cdot 10^{-5}$	per batch	$2.625 \cdot 10^{-4}$	$9.990 \cdot 10^{-6}$	$1.362 \cdot 10^{-4}$
1	$2.154 \cdot 10^{-5}$	per batch	$2.720 \cdot 10^{-4}$	$9.448 \cdot 10^{-6}$	$1.407 \cdot 10^{-4}$
1	0	per epoch	$2.968 \cdot 10^{-4}$	$8.702 \cdot 10^{-6}$	$1.527 \cdot 10^{-4}$
1	$2.154 \cdot 10^{-5}$	per batch	$3.014 \cdot 10^{-4}$	$1.126 \cdot 10^{-5}$	$1.563 \cdot 10^{-4}$
1	$2.154 \cdot 10^{-5}$	per epoch	$3.199 \cdot 10^{-4}$	$1.035 \cdot 10^{-5}$	$1.651 \cdot 10^{-4}$
1	$2.154 \cdot 10^{-5}$	per epoch	$3.233 \cdot 10^{-4}$	$1.036 \cdot 10^{-5}$	$1.668 \cdot 10^{-4}$
0	$2.154 \cdot 10^{-5}$	per batch	$3.429 \cdot 10^{-4}$	$1.006 \cdot 10^{-5}$	$1.765 \cdot 10^{-4}$
1	$3.072 \cdot 10^{-5}$	per batch	$3.604 \cdot 10^{-4}$	$1.155 \cdot 10^{-5}$	$1.860 \cdot 10^{-4}$
n/a	$4.840 \cdot 10^{-5}$	n/a	$4.285 \cdot 10^{-4}$	$1.367 \cdot 10^{-5}$	$2.211 \cdot 10^{-4}$

Purely taking an average of MSE values, already shows best performing SR-VAE models are the ones that have been retrained with fixed equations, directly after are SR-VAE models with a different β than was chosen as an optimum. However, as explained in the main body and further highlighted in the upcoming section, this ranking is flawed.

C.2. Ranking based on MSE z-score

As can be seen from the MSE averages in Table C.1, the outlier averages are almost two order of magnitudes bigger. Taking averages of the MSE values here will lead to the Outlier averages to dominate, even though the performance over the overall dataset is arguably more important. Therefore, before the averages are calculated, all the MSE values are normalized as a z-score, which results in the adjusted ranking shown in Table C.2.

Table C.2: Average MSE normalized as z-score of best trained SR-VAE models tested with UIUC and AFBench datasets for both decoder and SR equations ranked from lowest to highest overall average

γ	β	**SR training**	**Outlier Average**	**Datasets Average**	**Overall Average**
1	$2.154 \cdot 10^{-5}$	fixed	-0.8284	-0.6998	-0.7641
1	$2.154 \cdot 10^{-5}$	fixed	-0.7497	-0.7112	-0.7304
1	$2.154 \cdot 10^{-5}$	per batch	-0.4529	-0.6255	-0.5392
1	$3.072 \cdot 10^{-5}$	per batch	-0.5334	-0.5302	-0.5318
1	$3.072 \cdot 10^{-5}$	per batch	-0.4099	-0.6167	-0.5133
0	$2.154 \cdot 10^{-5}$	per batch	-0.5967	-0.3595	-0.4781
1	$3.072 \cdot 10^{-5}$	per batch	-0.2448	-0.5965	-0.4206
1	$3.072 \cdot 10^{-5}$	per batch	-0.3045	-0.4492	-0.3769
1	$2.154 \cdot 10^{-5}$	per epoch	-0.3861	-0.3485	-0.3673
1	$2.154 \cdot 10^{-5}$	per batch	-0.2015	-0.3942	-0.2979
1	$3.072 \cdot 10^{-5}$	per batch	-0.4212	0.0623	-0.1794
1	$2.154 \cdot 10^{-5}$	per epoch	-0.0001	-0.1017	-0.0509
1	$2.154 \cdot 10^{-5}$	per epoch	0.1829	0.0032	0.0931
1	0	per epoch	0.5173	-0.2836	0.1169
0	$2.154 \cdot 10^{-5}$	per batch	0.4173	-0.1093	0.1540
1	$2.154 \cdot 10^{-5}$	per batch	-0.1175	0.4440	0.1633
1	$3.072 \cdot 10^{-5}$	per epoch	0.3464	0.3625	0.3545
1	$2.154 \cdot 10^{-5}$	per batch	0.4290	1.0910	0.7600
1	$3.072 \cdot 10^{-5}$	per batch	1.1938	0.8066	1.0002
n/a	$4.84 \cdot 10^{-5}$	n/a	2.1599	3.0562	2.6081

Note that, the top two are both VAE models that have been retrained once again with fixed SR equations. Coincidentally, the third best VAE model in the ranking was used as a warm start for the top two. This further confirms the positive effect of retraining with fixed SR equations, after a run has already been conducted where every batch contains a SR search. Nevertheless, alternative training procedure where SR is only run per epoch also shows promise. This option is more attractive if speed is of the essence.

C.3. Additional results of Fixed SR-VAE

This section contains some additional results regarding the fixed SR-VAE.

C.3.1. Latent effects plot

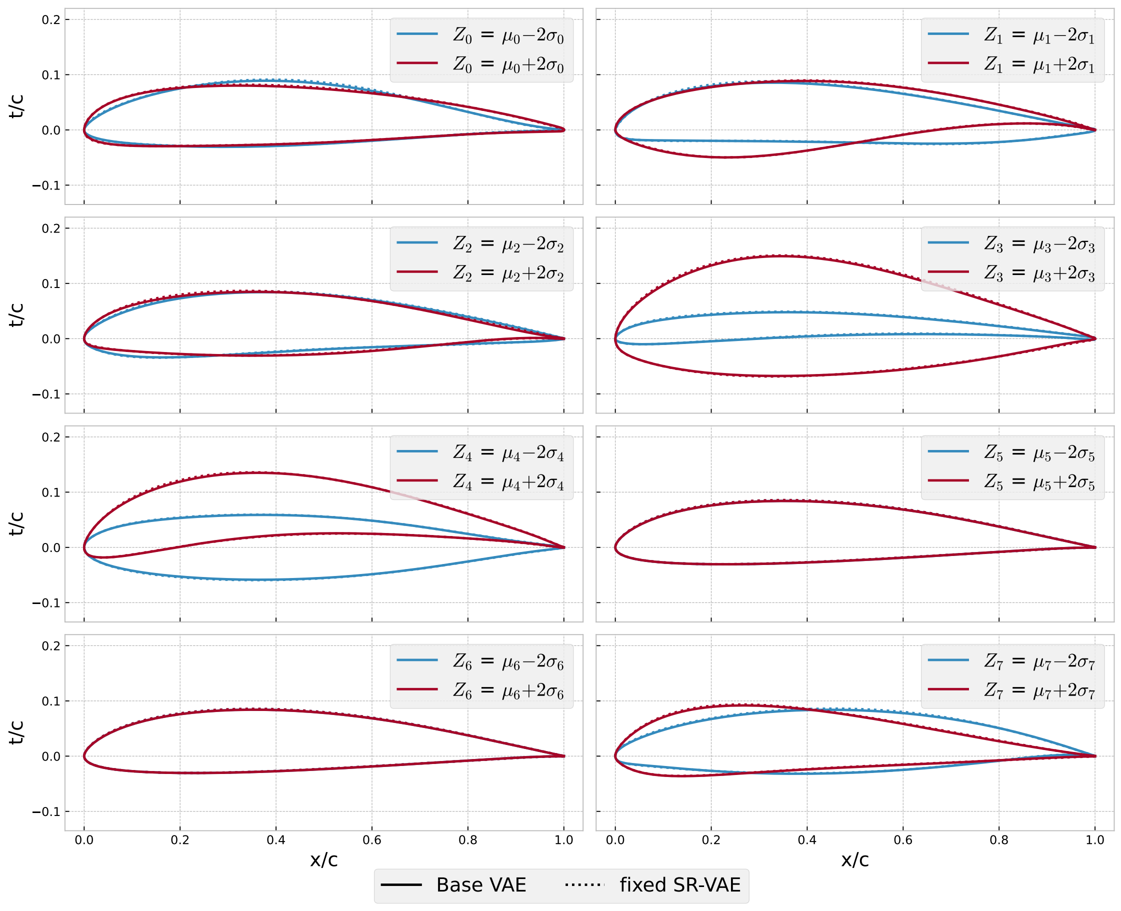
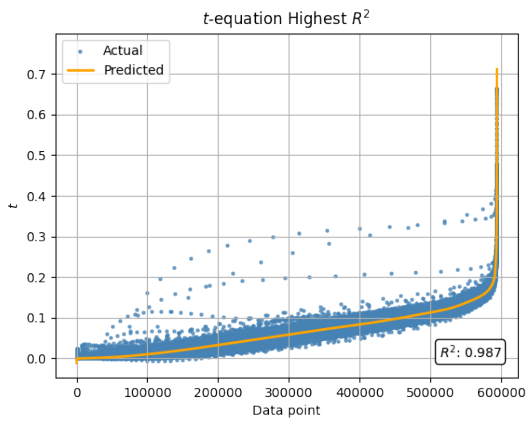


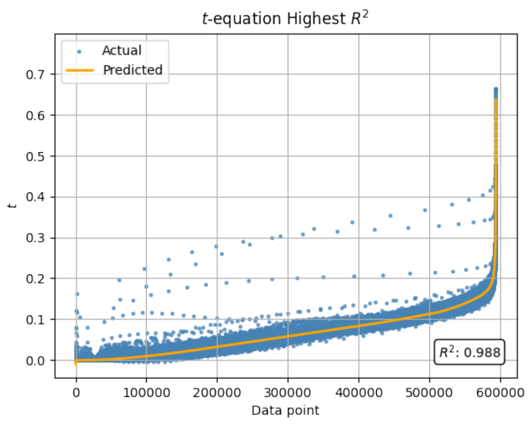
Figure C.1: Latent effects plot of fixed SR-VAE with tuned base-VAE for reference

C.4. Choice of t-equation SR-VAE fixed

Since a different thickness equation from the equation fixed during retraining of the SR-VAE per batch was chosen, this section contains a bit more information regarding the equation selection. In Figure C.2, the left figure shows the equation that was used to train the fixed SR-VAE, while the plot on the right shows the equation and plot of the chosen equation with lower average MSE. Note that a different equation has been used during training than was identified in Figure 5.23a. The reason for this is due to the similar performance at reduced complexity.



(a) $\sin((X - 1.0030) \cdot -0.0898 \cdot (X^{0.5332} + X \cdot (Z_1 + Z_3 + Z_3 - 0.9897) \cdot 0.0655) \cdot (Z_3 + (X - 0.3912) \cdot (Z_1 + Z_7 + (Z_4 + 1.4510) \cdot 0.3992) \cdot -1.2589 + 3.2107))$



(b) $(X^{0.5297} + X \cdot \sin(Z_3 \cdot 0.2051)) \cdot (Z_3 + (X - 0.4344) \cdot (Z_0 \cdot (X + (X \cdot 0.9099)^{0.3567} - 1.0805) + (1.3503^{Z_4} + Z_1 + Z_3 \cdot 0.0971 + Z_7) \cdot -1.0801) + 3.2726) \cdot (X - 0.9973) \cdot -0.0821$

Figure C.2: Predicted values, of thickness equations with the lowest overall MSE and fixed equation from retrained fixed SR-VAE, plotted against the actual values of adjusted test datasets, sorted by ascending predicted values

As can be seen here, there is a visibly larger spread in Figure C.2a compared to Figure C.2b, which is reflected in the difference in R^2-score. The equation choice is further quantified by investigating the metrics for SR-VAE with the fixed thickness equation, as shown in Table C.3.

Table C.3: SR performance of parametric equations from SR-VAE where SR-VAE per batch is retrained with fixed equations, using the thickness equation fixed during in training

Outliers UIUC	Outliers AFBench	Overall UIUC	Overall AFBench
$5.2942 \cdot 10^{-4}$	$1.8460 \cdot 10^{-4}$	$1.6902 \cdot 10^{-5}$	$1.4793 \cdot 10^{-5}$

Directly comparing this with the SR performance shown in Table 5.8, shows that the thickness equation fixed during training leads to better outlier performance but worse overall. Since the overall performance slightly outweighs outlier performance and because it exhibits the highest R^2, the equation from Figure C.2b is selected for the final SR-VAE. Both the performance metrics and the equation fit plot underline the equation selection.

D. SR-VAE performance on AFBench

This chapter contains the results of the final SR-VAE based on the AFBench dataset. The main reason it is not included in the main body is that the dataset has a lower overall variability than the UIUC dataset and because it contains airfoil shapes that are unrealistic.

D.1. Inferred latent distributions

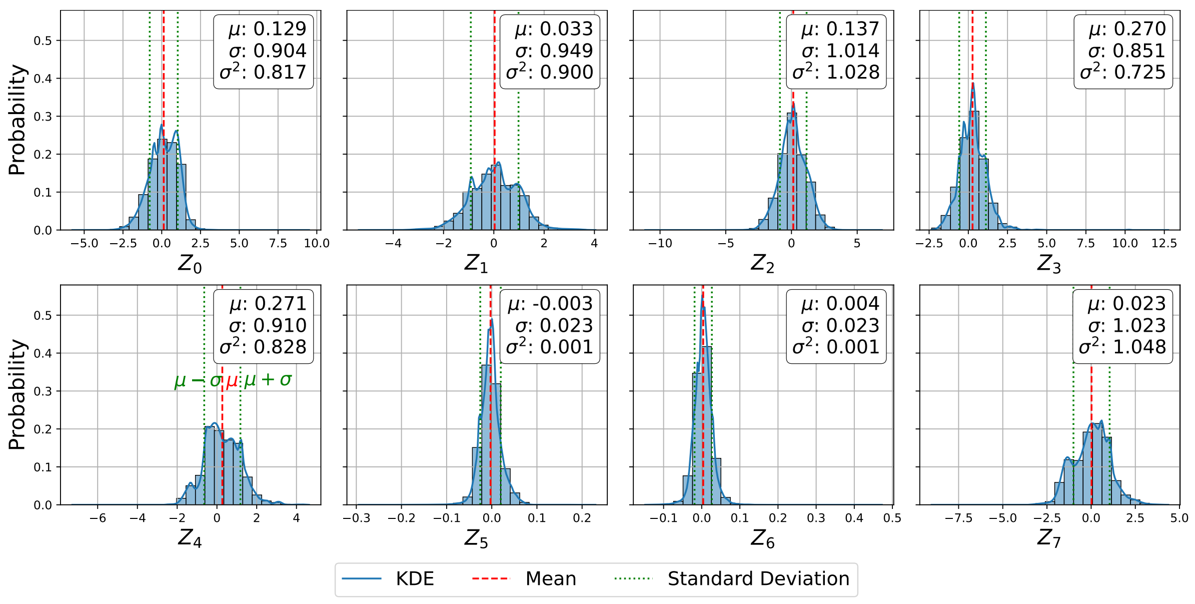


Figure D.1: Inferred latent distributions from AFBench dataset for the final SR-VAE

D.2. Cumulative distribution plot

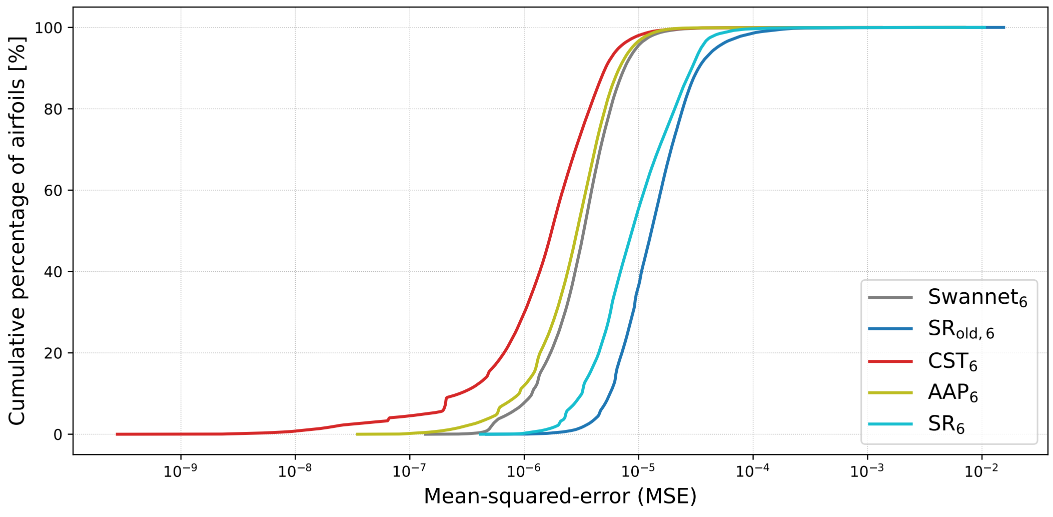


Figure D.2: Cumulative distribution plot of all airfoil MSE losses from the AFBench dataset, for SR-VAE model and its equations, the original model from Swannet et al., the SR equations from the latent analysis and CST parameterization

D.3. Validation sets worst decoded

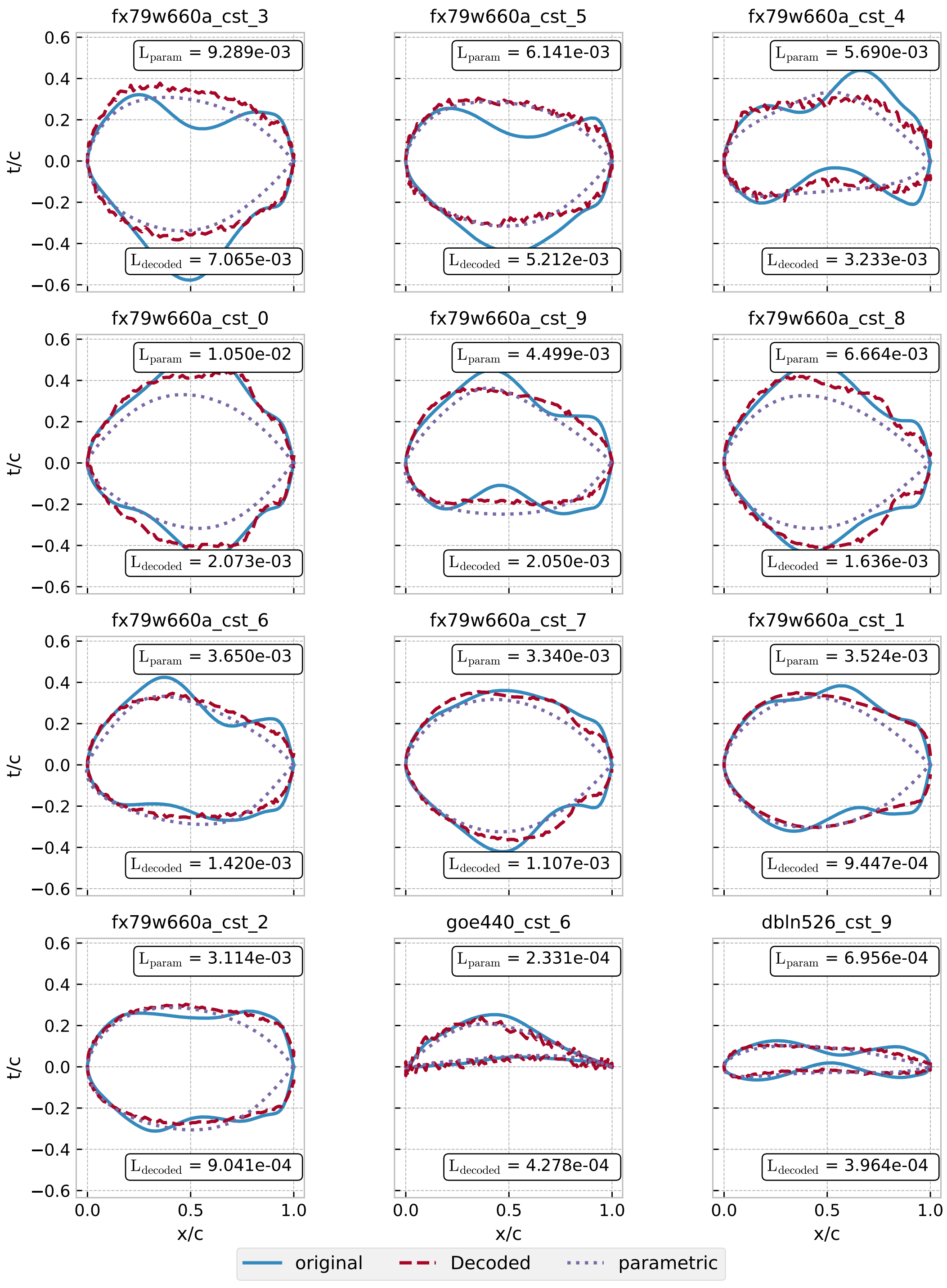


Figure D.3: Validation plots of final SR-VAE showing parametric and decoded reconstruction of 12 airfoils with highest decoder loss from the AFBench dataset, the losses marked at the corners of each plot is the MSE

D.4. Validation sets worst parametric

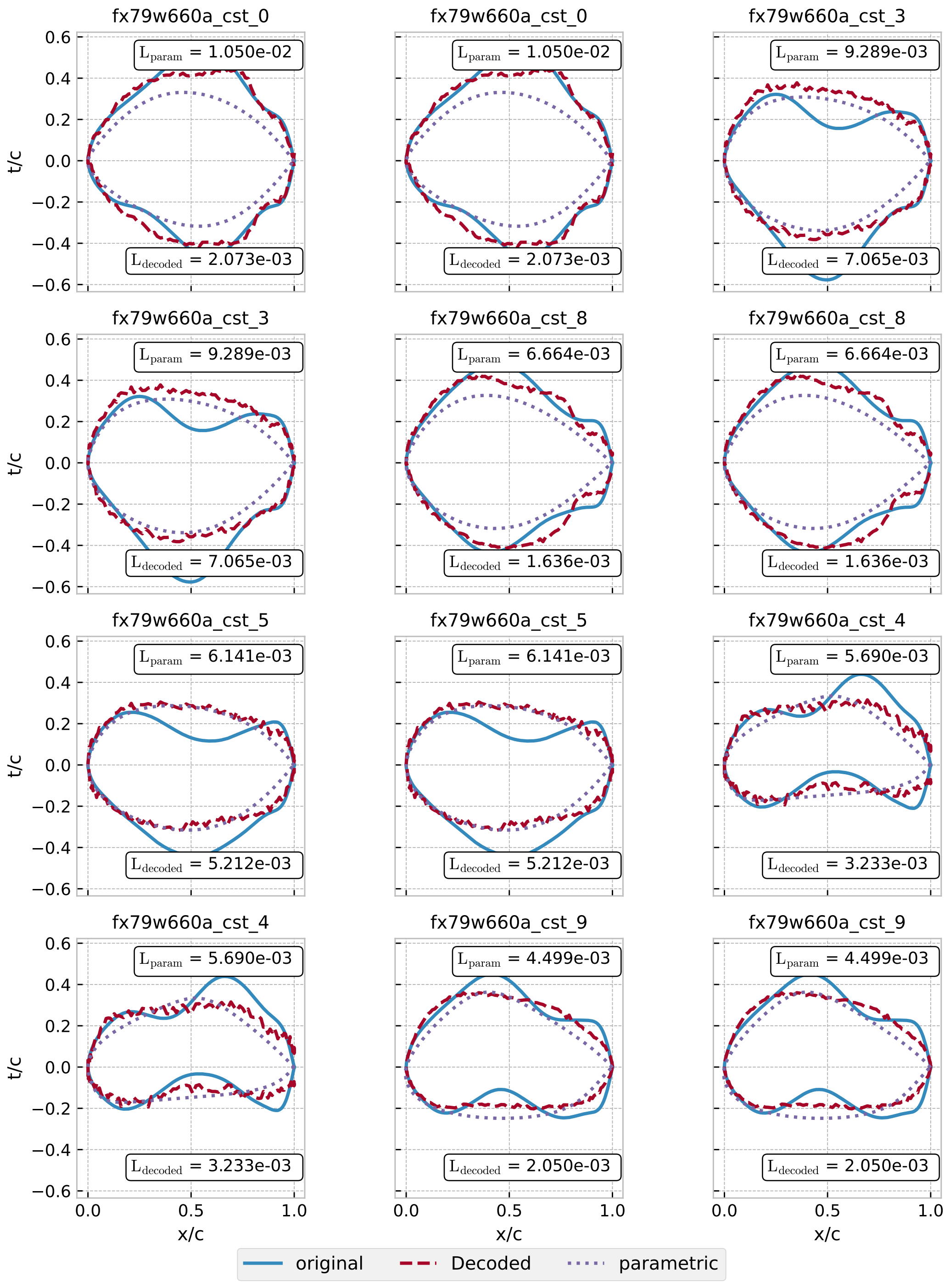


Figure D.4: Validation plots of final SR-VAE showing parametric and decoded reconstruction of 12 airfoils with highest parametric loss from the AFBench dataset, the loss marked at corner of each plot is the MSE

E. Additional Results DAE-11 Optimization

This appendix contains additional results of the DAE-optimization case by Drela [85].

E.1. Daedalus optimization without α

The DAE-11 optimization in the main body follows the problem defined by Sharpe to validate Neuralfoil [87]. The original problem is taken from Drela, who specified that using α as a free variable with the lift constraints essentially eliminates it as a design variable [85]. Nevertheless, as an extra check this appendix contains an adjusted version of the problem where the shape parameters are the only design variable. The reader is advised to first go through the optimization problem defined in the main body, before addressing this optimization problem. This chapter is purely meant to eliminate the possibility that the presence of α as a design variable had any detrimental effect to the main optimization results.

E.1.1. Problem definition and implementation

The new optimization problem is now defined as follows, with no C_L constraint or α as design variable:

$$\text{Minimize:} \quad F(\{S_i\}) \equiv \frac{5}{45} C_D|_{C_L=0.8} + \frac{6}{45} C_D|_{C_L=1.0} + \frac{7}{45} C_D|_{C_L=1.2} \tag{E.1}$$

$$+ \frac{8}{45} C_D|_{C_L=1.4} + \frac{9}{45} C_D|_{C_L=1.5} + \frac{10}{45} C_D|_{C_L=1.6} \tag{E.2}$$

$$\text{Subject to:} \quad \begin{array}{ll} C_M \geq -0.133 & (t/c)_{0.33c} \geq 0.128 \\ \theta_{\text{TE}} \geq 6.25^\circ & (t/c)_{0.90c} \geq 0.014 \end{array} \tag{E.3}$$

$$\text{Design variables:} \quad S_i \tag{E.4}$$

The rest of the optimization problem and implementation remains the exact same, but with no initial guess or bounds for the angle of attack. The C_L equality for the different optimization points in the objective function is now enforced by directly defining them within the objective function. Since NeuralFoil only takes α, Re and M as arguments, this is done by passing a range of α values to the solver where the desired C_D values are obtained at the specified C_L values through interpolation. This should lead to quicker optimizations, since the optimizer has fewer design variables to optimize.

E.1.2. Results

Once again, first the CST optimization is applied to the optimization case, as shown in Figure E.1. As can be seen here, the optimized shape is nearly identical to what was shown in Figure 5.34, with a higher C_D improvement over the DAE-11 higher than before. However, a closer look revealed that the objective value in this optimization is slightly higher. Therefore, the improvement in higher drag does not necessarily indicate that a lower optimum is found, but rather that it is a side effect of the way C_L is enforced within the objective function. The target objective value, calculated by passing the DAE-11 to the objective function, is higher than that of the original objective function as defined by Sharpe [87].

The overall runtime of the optimization was 22.46 seconds. As can be seen in the convergence history, it converged after 84 iterations, which means that the speed is now 3.74 iterations per seconds. As expected, this is quicker than the convergence speed of 2.87 found in the original optimization, and apparently fewer iterations are required for convergence. However, looking at the convergence history, there are some more overshoots than was shown in Figure 5.34. This indicates that this optimization might be slightly less stable.

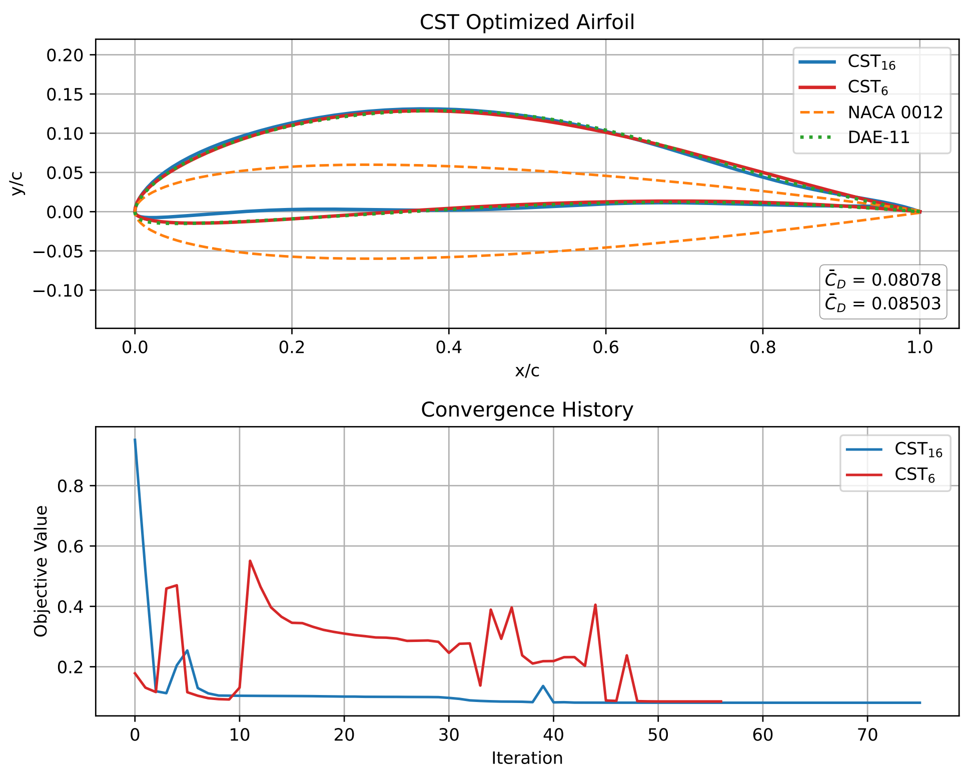


Figure E.1: Airfoil shape and convergence history of CST airfoil shape optimization of the subsonic Daedalus optimization case without α as design variable, including the mean drag objective value $\bar{C}_D$

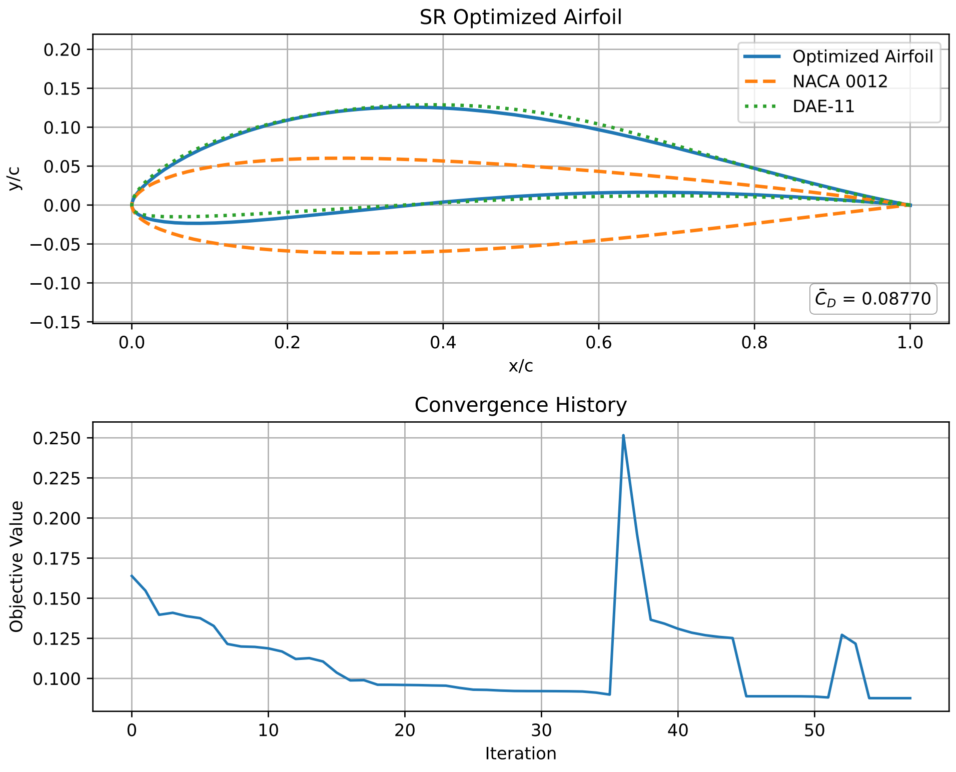


Figure E.2: Airfoil shape and convergence history of SR airfoil shape optimization of the subsonic Daedalus optimization case without α as design variable, including the mean drag objective value $\bar{C}_D$

Now, doing the same for the SR equations leads to the result shown in Figure E.2. Similarly to CST optimization, the optimized shape is nearly identical to that of Figure 5.35, with a similar C_D improvement over the target airfoil. However, once again it was observed that the overall objective values are slightly higher. This once again indicates that incorporating α as a design variable does lead to slightly more optimal minima.

The optimization converged in 4.24 seconds after 38 iterations. The amount of iterations required for convergence is identical to before, but the convergence speed increased to 8.96 iterations per second. The convergence behaviour is also identical this time. This indicated that the SR equations might be more resistant to changes to the optimization problem than the CST parameterization. Nevertheless, besides the higher convergence speed, removing α as a design variable does not seem to effect the optimization too much.

Finally, the capability of the SR-VAE can be assessed with this optimization. Especially since in the main optimization, it was identified that the C_L constraint caused the optimization to not converge. The result of the optimization is shown in Figure E.3

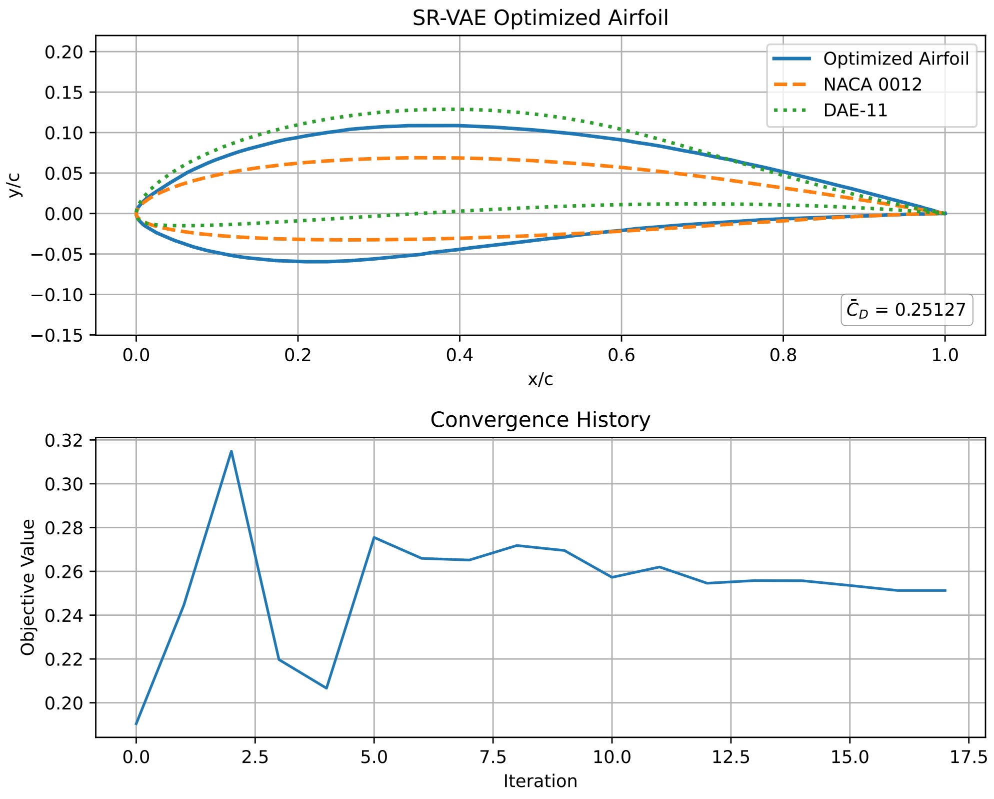


Figure E.3: Airfoil shape and convergence history of VAE airfoil shape optimization of the subsonic Daedalus optimization case without α as design variable, including the mean drag objective value $\bar{C}_D$

Unfortunately, the optimized result is significantly worse than that of Figure E.3, showing an increase of the mean drag by 209%. The overall runtime is 19.71 seconds for 61 iterations, which means that the speed is 3.09 iterations per second. Remarkably, the convergence speed has not changed due to absence of α as a design variable. While from the convergence history seems to indicate that the optimization converged, it in fact terminated early due to a ”positive directional derivative for linesearch” with success set to `False`. This means that the SLSQP optimization failed to converge because it could not find a direction to decrease the objective while also satisfying the constraints. Once again, the trailing edge angle constraint failed, as shown in Figure E.4.

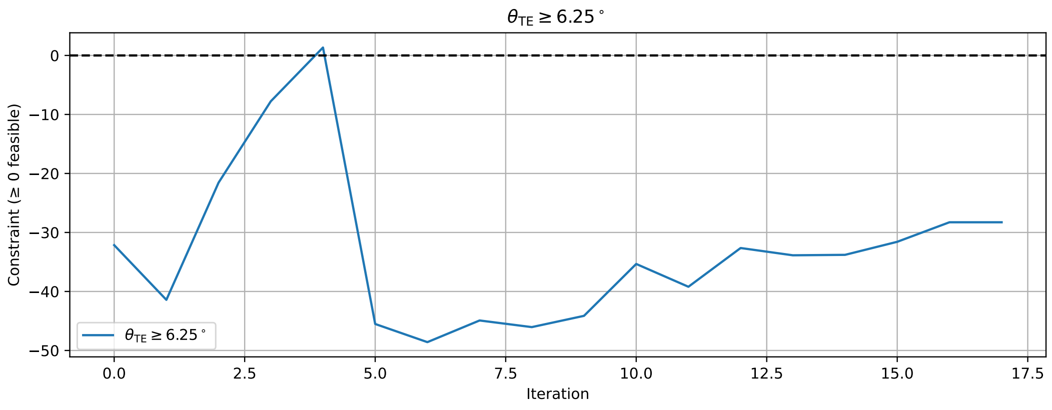


Figure E.4: Violated constraints history of VAE airfoil shape optimization without α as design variable, showing the worst-case violation for every iteration

Before in Figure 5.38, the optimization initially seemed to converge while satisfying this constraint before violating it in the converged result. This time however, besides the overshoots near the start, the trailing edge constraint is never satisfied. Once again it seems that the nonlinearity and the stability of the SR-VAE decoder limits its application in aerodynamic shape optimization. The exclusion of α as a design variable did not help, it in fact only underlined the observation in the main body.

E.2. SR-VAE Daedalus optimization with no changes

This section follows the DAE-11 optimization problem defined in the main body, including angle of attack as design variable. Figure E.5 shows the result of the SR-VAE optimization using NACA-0012 as the starting point without the NeuralFoil `analysis_confidence` constraint.

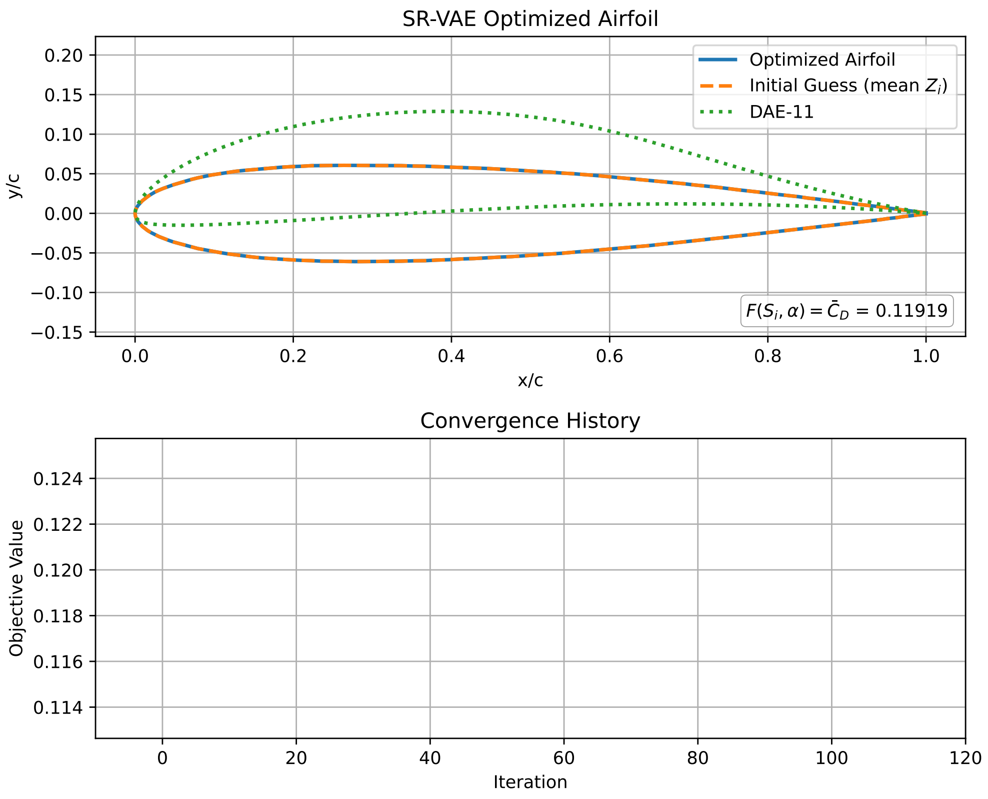


Figure E.5: Airfoil shape and convergence history of VAE airfoil shape optimization of the subsonic Daedalus optimization case with NACA-0012 as initial guess, including the mean drag objective value $\bar{C}_D$

E.3. Constrained NACA 0012 optimization additional results

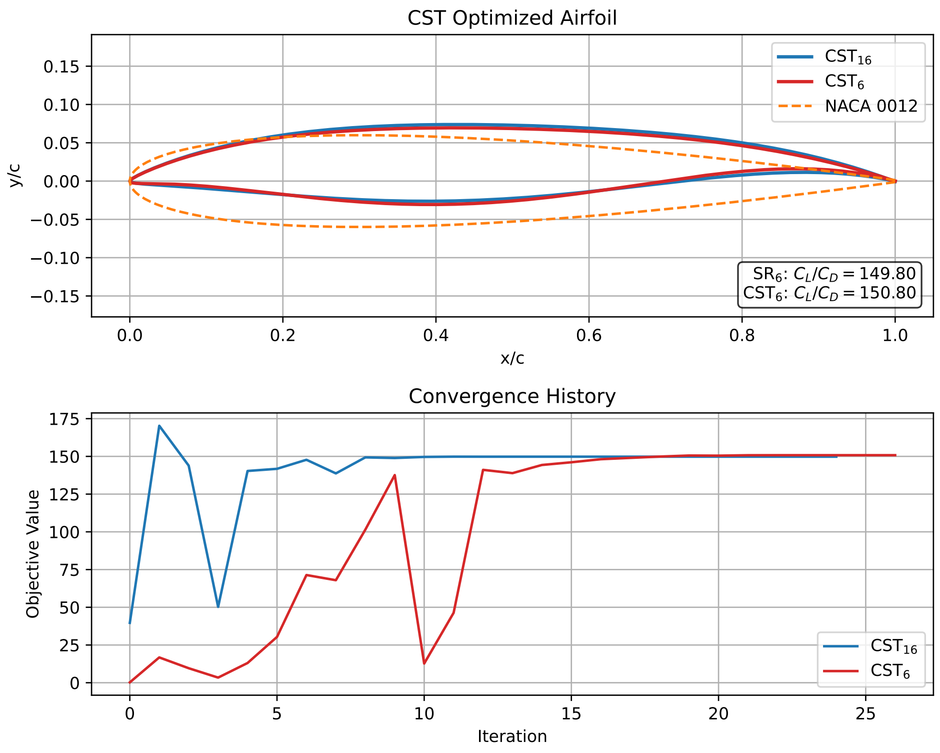


Figure E.6: Airfoil shape and convergence history of SR and CST$_6$ airfoil shape optimization for the constrained NACA 0012 optimization case, including the lift over drag objective value C_L/C_D

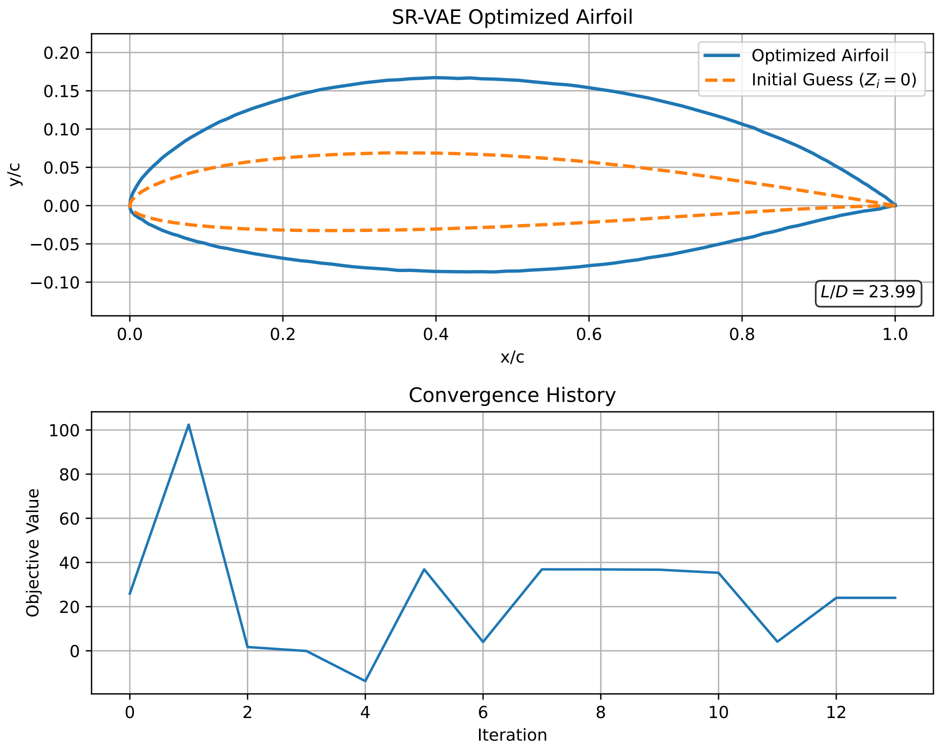


Figure E.7: Airfoil shape and convergence history of VAE airfoil shape optimization for the constrained NACA 0012 optimization case without NeuralFoil constraint, including the lift over drag objective value C_L/C_D

E.4. Unconstrained NACA 0012 optimization additional results

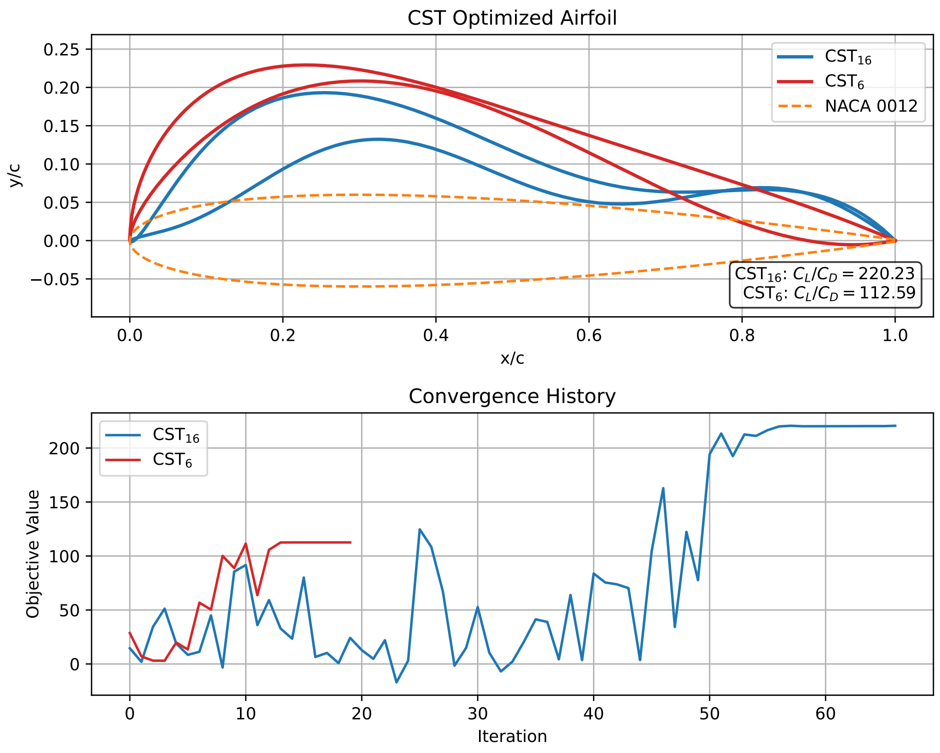


Figure E.8: Airfoil shape and convergence history of CST_{16} and CST_6 airfoil shape optimization without NeuralFoil constraint, for the unconstrained NACA 0012 optimization case, including lift over drag objective value C_L/C_D