Wei Chen[1]
Department of Mechanical Engineering,
University of Maryland,
College Park, MD 20742
e-mail: wchen459@umd.edu

Mark Fuge
Department of Mechanical Engineering,
University of Maryland,
College Park, MD 20742
e-mail: fuge@umd.edu

Synthesizing Designs With Interpart Dependencies Using Hierarchical Generative Adversarial Networks

Real-world designs usually consist of parts with interpart dependencies, i.e., the geometry of one part is dependent on one or multiple other parts. We can represent such dependency in a part dependency graph. This paper presents a method for synthesizing these types of hierarchical designs using generative models learned from examples. It decomposes the problem of synthesizing the whole design into synthesizing each part separately but keeping the interpart dependencies satisfied. Specifically, this method constructs multiple generative models, the interaction of which is based on the part dependency graph. We then use the trained generative models to synthesize or explore each part design separately via a low-dimensional latent representation, conditioned on the corresponding parent part(s). We verify our model on multiple design examples with different interpart dependencies. We evaluate our model by analyzing the constraint satisfaction performance, the synthesis quality, the latent space quality, and the effects of part dependency depth and branching factor. This paper's techniques for capturing dependencies among parts lay the foundation for learned generative models to extend to more realistic engineering systems where such relationships are widespread. [DOI: 10.1115/1.4044076]

1 Introduction

Representing a high-dimensional design space with a low-dimensional *latent space* makes it easier to explore, visualize, or optimize complex designs. This often means finding a *latent representation* or a *manifold* along which valid designs, such as geometries, lie [1,2].

While this works well for single parts, designs usually have multiple parts with interpart dependencies. For example, the size and position of a conduit, lightening, or alignment hole in an airframe structure depend on the shape of the airfoil. We can describe interpart dependencies in a design using a *directed acyclic graph* (DAG). This DAG captures whether the geometry of a part depends on the geometry of its parent part(s). In this case, one may want to identify first the *parent manifold* that captures major variation of parent shapes and then the *child manifold* that captures major variation of feasible child shapes conditioned on any parent shape (Fig. 1). Because, for example, we may first optimize the airfoil shape on the airfoil manifold (parent) to obtain the optimal lift and drag, and then given the optimal airfoil, we may optimize the hole's size and position on the hole manifold (child) for other considerations like light-weighting, etc.

However, finding individual part manifolds that both represent the design space well, while also satisfying part configuration, is nontrivial. Traditionally, to learn the interpart dependency, one has to either define explicit constraints [3,4] or learn implicit constraints via adaptive sampling [2,5]. The former uses hard-coded (often application-specific) constraints and hence lacks flexibility; whereas, the latter queries external sources by human annotation, experiment, or simulation, and thus is expensive. In this paper, we solve these problems by instead learning these constraints given examples. We assume that we only have the prior knowledge on interpart dependencies but not the specific types of constraints (e.g., concentric, alignment, tangent, etc.) that confine the geometry of each part. We do this by identifying different levels of manifolds, where the higher-level (parent) manifold imposes implicit constraints on the lower-level (child) manifolds. We propose a deep generative model that synthesizes designs in a hierarchical manner according to those interpart dependencies: it first synthesizes parent parts, then synthesizes parts conditioned on those parent parts, and so on. At each level, the model simultaneously captures a parent manifold and an infinite number of child manifolds which are conditioned on parent parts. This results in latent spaces that can synthesize and search each part individually as well as their assemblies. Importantly, our method is fully data-driven and requires no hard-coded rules or querying external sources, except for providing an example dataset of designs with part correspondence and known interpart dependencies. For designs without part correspondence, we can apply unsupervised cosegmentation [6] as a preprocessing step.

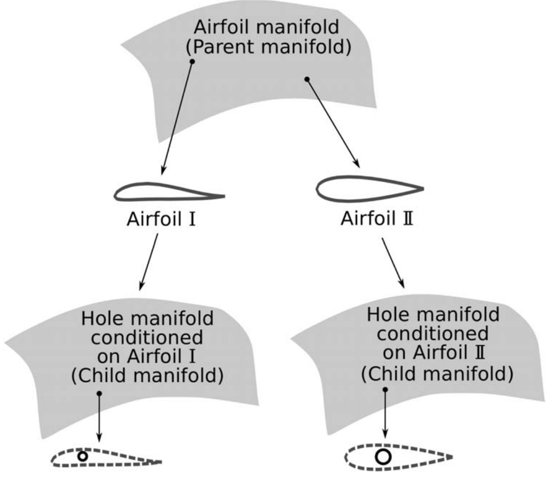
Airfoil manifold
(Parent manifold)
Airfoil I
Airfoil II
Hole manifold conditioned on Airfoil I (Child manifold)
Hole manifold conditioned on Airfoil II (Child manifold)

Fig. 1 Manifolds of parent and child shapes

[1]Corresponding author.

Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received February 18, 2019; final manuscript received June 16, 2019; published online June 24, 2019. Assoc. Editor: Jitesh H. Panchal.

The paper's key contributions are as follows:

(1) A novel deep generative model architecture that simultaneously learns a design's interpart dependencies and each part's geometry variation conditioned on the corresponding parent part(s). It decouples each part's latent space so that we can perform design space exploration in separate low-dimensional latent spaces.
(2) New benchmark datasets—both real-world and synthetic—that can be used for studying different kinds of interpart dependencies, including type of geometric constraints, depth of hierarchy, and branching factor of parent/child relationships. This dataset can aid in future evaluation of generative models of hierarchical parts.
(3) Characterizing the effects of sample size and part dependencies' complexity (depth and branching factor) on the synthesis performance of our generative model.
(4) A new evaluation metric for generative models that measures the consistency of shape variation in the latent space.

2 Related Work

Our work produces generative models that synthesize designs from latent representations. There are primarily two streams of related research from the fields of engineering design and computer graphics—specifically, design space dimensionality reduction and design synthesis. We also review generative adversarial networks (GANs) [7], which we use to build our model.

2.1 Design Space Dimensionality Reduction. While designs can be parametrized by various techniques [8], the number of design variables (i.e., the dimensionality of a design space) increases with the geometric variability of designs. In tasks like design optimization, to find better designs, we usually need a design space with higher variability, i.e., higher dimensionality. This demand creates the problem of exploring a high-dimensional design space. Based on the curse of dimensionality [9], the cost of exploring the design space grows exponentially with its dimensionality. Thus, researchers have studied approaches for reducing the design space dimensionality. Normally, dimensionality reduction methods identify a lower-dimensional latent space that captures most of the design space's variability. This can be grouped into linear and nonlinear methods.

Linear dimensionality reduction methods select a set of optimal directions or basis functions where the variance of shape geometry or certain simulation output is maximized. Such methods include the Karhunen–Loève expansion [10,11], principal component analysis (PCA) [12], and the active subspaces approach [13].

In practice, it is more reasonable to assume that design variables lie on a nonlinear manifold rather than a hyperplane. Thus, researchers also apply nonlinear methods to reduce the dimensionality of design spaces. This nonlinearity can be achieved by (1) applying linear reduction techniques locally to construct a nonlinear global manifold [12,14–17], (2) using kernel methods with linear reduction techniques (i.e., using linear methods in a reproducing kernel Hilbert space that then induces nonlinearity in the original design space) [1,12], (3) latent variable models like Gaussian process latent variable model (GPLVM) and generative topographic mapping [18], and (4) neural network-based approaches such as self-organizing maps [19] and autoencoders [1,12,20,21].

This work differs from these past approaches in that we aim at identifying two-level latent spaces with the lower-level encodes interpart dependencies, rather than learning only one latent space for the complete design.

2.2 Data-Driven Design Synthesis. Design synthesis methods can be divided into two categories: rule-based and data-driven design synthesis. The former (e.g., grammar-based design synthesis [3,4,22]) requires labeling of the reference points or surfaces and defining rule sets so that new designs are synthesized according to this hard-coded prior knowledge while the latter learns rules/constraints from a database and generates plausible new designs with similar structure/function to exemplar in the database.

Usually, dimensionality reduction techniques allow inverse transformations from the latent space back to the design space, thus can synthesize new designs from latent variables [1,11,12,21]. For example, under the PCA model, the latent variables define a linear combination of principal components to synthesize a new design [11]; for local manifold-based approaches, a new design can be synthesized via interpolation between neighboring points on the local manifold [17]; and under the autoencoder model, the trained decoder maps any given point in the latent space to a new design [20,21]. Researchers have also employed generative models such as kernel density estimation [23], Boltzmann machines [24], variational autoencoders (VAEs) [25], and GANs [26,27] to learn the distribution of samples in the design space and synthesize new designs by drawing samples from the learned distribution. Discriminative models like deep residual networks [28] are also used to generate 3D shapes.

These aforementioned models synthesize a design or a shape as a whole. There are methods that synthesize new shapes by assembling or reorganizing parts from an existing shape database while preserving the desired structures [29–33]. The shapes are usually parametrized by high-level abstract representations, such as hand-crafted feature vectors [30] or shape grammars [31]. While these methods edit shapes at a high-level, they do not control the local geometry of each synthesized component.

Previously, the interpart dependencies of shapes have been modeled by grammar induction [31], kernel density estimation [34], probabilistic graph models [24,29,30], and recursive autoencoders [27]. Those methods handle part relations and design synthesis separately. In contrast, our method encodes part relations through the model architecture so that it simultaneously learns the interpart dependencies and single part geometry variation. The model can also be used for inferring the generative distribution of each part conditioned on any parent part.

2.3 Generative Adversarial Networks. Generative adversarial nets [7] model a game between a generative model (*generator*) and a discriminative model (*discriminator*). The generative model maps an arbitrary noise distribution to the data distribution (i.e., the distribution of designs in our scenario), thus can generate new data while the discriminative model tries to perform classification, i.e., to distinguish between real and generated data (Fig. 2). The generator G and the discriminator D are usually built with deep neural networks. As D improves its classification ability, G also improves its ability to generate data that fools D. Thus, the objective of GAN is

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim P_{\text{data}}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim P_{\boldsymbol{z}}}[\log(1 - D(G(\boldsymbol{z})))] \tag{1}$$

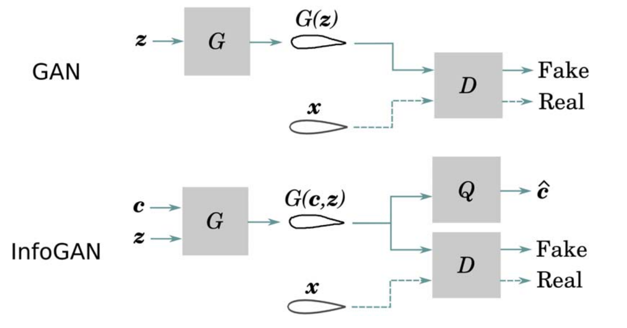

Fig. 2 Architectures of the standard GAN and the InfoGAN

where $\boldsymbol{x}$ is sampled from the data distribution P_{data}, z is sampled from the noise distribution P_z, and $G(z)$ is the generator distribution. A trained generator thus can map the predefined noise distribution to the distribution of designs. The noise input z is considered as the latent representation in the dimensionality reduction scenario, since z captures the variability of the data.

One advantage GANs hold over VAEs [35] is that GANs tend to generate more realistic data [36]. But a disadvantage of the original GAN formulation is that it cannot learn an interpretable latent representation. Built upon these "vanilla" GANs, the InfoGAN [37] aims at regularizing the latent representation of the data space by maximizing a lower bound of the mutual information between a set of *latent codes* $\boldsymbol{c}$ and the generated data. The generator is provided with both z and $\boldsymbol{c}$. Thus, the generator distribution $P_G = G(\boldsymbol{c}, z)$ is conditioned on $\boldsymbol{c}$. The mutual information lower bound L_I is

$$L_I(G, Q) = \mathbb{E}_{\boldsymbol{c}\sim P(\boldsymbol{c}),\boldsymbol{x}\sim G(\boldsymbol{c},z)}[\log Q(\boldsymbol{c}|\boldsymbol{x})] + H(\boldsymbol{c}) \tag{2}$$

where $H(\boldsymbol{c})$ is the entropy of the latent codes, and $Q(\boldsymbol{c}|\boldsymbol{x})$ is called the auxiliary distribution which approximates $P(\boldsymbol{c}|\boldsymbol{x})$. We direct interested readers to Ref. [37] for the derivation of L_I. The InfoGAN objective combines L_I with the standard GAN objective

$$\min_{G,Q} \max_{D} V_{\text{InfoGAN}}(D, G, Q) = V(D, G) - \lambda L_I(G, Q) \tag{3}$$

where λ is a weight parameter.

In practice, $H(\boldsymbol{c})$ is treated as a constant if the distribution of $\boldsymbol{c}$ is fixed. The auxiliary distribution Q is parametrized by a neural network—here, we call it the *auxiliary network*.

In our design synthesis scenario, the latent codes $\boldsymbol{c}$ can represent any continuous or discrete factor that controls the geometry of the design, e.g., the upper/lower surface protrusion of the airfoil.

Previously, there are GAN-based models that generate specific types of data (i.e., images and videos) by a two-level hierarchy [38,39]. For example, a generator first generates the structure of an image, and then conditioned on that structure, another generator generates the texture of that image. In the case of design synthesis, it is intuitive to generate each design component given all its dependencies, which results in a hierarchical model structure with multiple levels of generators.

3 Method

In this section, we introduce our proposed deep neural network architecture and its training details.

3.1 Problem Formulation. We can use a directed acyclic graph to define interpart dependencies of a design. We call this graph a *part dependency graph*. For example, suppose we want to design an airfoil with two holes inside (top left in Fig. 3). We might first design the airfoil (part A) and, then, set the position and diameter of one hole (part B) based on the shape of the airfoil, followed by the design of the second hole (part C) based on both the airfoil shape and the first hole. Thus, the dependencies can be expressed by the graph shown in the bottom left of Fig. 3.

This paper deals with a design synthesis and a design space exploration problem, where we have a database of geometric designs, each of which has multiple parts. We have no prior knowledge on the specific types of constraints that confine the geometry of each part but only interpart dependencies. We propose a model that learns to synthesize designs based on those interpart dependencies. We want to use this model to (1) correctly synthesize each part that follows both the shape feasibility and the dependency constraints and (2) use low-dimensional *latent spaces* $\mathcal{C}$ to represent the design spaces $\mathcal{X}$ of each part. The ability to learn separate representations of each part is useful for decomposing a design space exploration problem (e.g., in design optimization).

Desired latent spaces satisfy the requirements as follows:

(1) Any child latent space should be conditioned on a parent shape (e.g., for the design in Fig. 3, any latent space of the first hole should be conditioned on an airfoil).
(2) Major variability across designs in the database should be captured by those latent spaces.
(3) Designs should change consistently as we move along any basis of the latent space. This regularity/consistency/smoothness will improve latent space design exploration and optimization.

To meet the first requirement, we construct a composite generative model—a model with multiple generators, each of which learns a (conditional) generative distribution of a part from the design. We ensure the rest of the requirements by adapting InfoGAN's architecture and objective, i.e., conditioning the generator on latent codes and maximizing a lower bound of the mutual information between the synthesized designs and their latent representations. This is known to (1) make latent codes target salient features of the designs and (2) disentangle the latent representation of the data space [37]. We introduce the details of our model's architecture in Sec. 3.2.

3.2 Model Architecture. To learn separate distributions/representations of each part, we use a generative adversarial net with

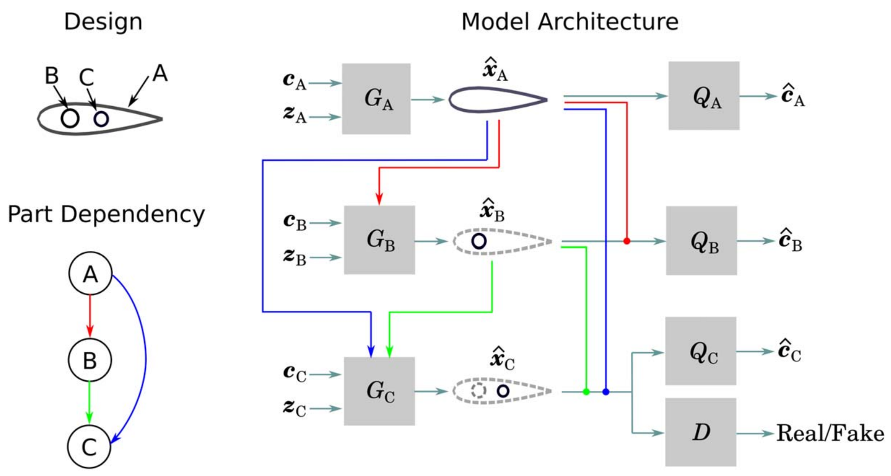

Fig. 3 An example of part dependency and the corresponding hierarchical GAN architecture. The interaction between generators (G_A, G_B, and G_C) and auxiliary networks (Q_A, Q_B, and Q_C) is based on the connection of the part dependency graph. This figure is best viewed in color.

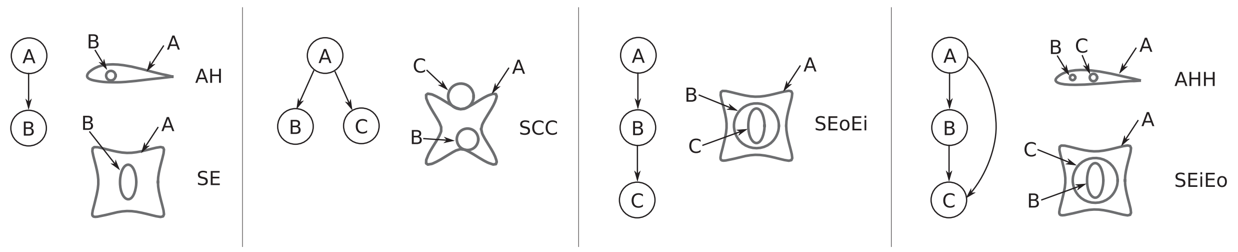

Fig. 4 Datasets with different interpart dependencies

multiple generators/auxiliary networks. We call this network the hierarchical generative adversarial networks (HGANs). An example of its architecture is shown in Fig. 3. We use multiple generators to synthesize different parts in a design. The interaction between generators/auxiliary networks changes with the part dependency graph. The latent code $\boldsymbol{c}_j$ defines the latent representation of the jth part, where $j = 1, \ldots, n$, and n is the number of parts. The generator G_j learns a (conditional) shape distribution $P(\boldsymbol{x}_j|\boldsymbol{c}_j, \boldsymbol{x}_{\mathrm{Par}(j)})$, where Par ($j$) denotes the set of parent(s) of the jth part. For the example in Fig. 3, $\mathrm{Par}(A) = \varnothing$, $\mathrm{Par}(B) = \{A\}$, and $\mathrm{Par}(C) = \{A, B\}$. The output distribution $P_{G_j} = G_j(\boldsymbol{c}_j, z_j, \hat{\boldsymbol{x}}_{\mathrm{Par}(j)})$ represents the distribution of the synthesized part $\hat{\boldsymbol{x}}_j$, where $\hat{\boldsymbol{x}}_{\mathrm{Par}(j)}$ denotes the synthesized parent part(s) of $\boldsymbol{x}_j$, and z_j is the noise input to G_j.

The auxiliary network Q_j predicts the latent code distribution of the corresponding part, i.e., to estimate the conditional distributions $P(\boldsymbol{c}_j|\boldsymbol{x}_j, \boldsymbol{x}_{\mathrm{Par}(j)})$. The discriminator D predicts whether a full design is from the database or generated by generators. A properly trained D should distinguish designs with unrealistic or mismatched parts.

The objective of HGAN is expressed as

$$\min_{\{G_j\},\{Q_j\}} \max_{D} V_{\mathrm{HGAN}}(D, \{G_j\}, \{Q_j\}) = V(D, \{G_j\}) - \lambda L_I(\{G_j\}, \{Q_j\}) \tag{4}$$

where $\{G_j\}$ and $\{Q_j\}$ denote the set of generators and auxiliary networks, respectively. The first term in Eq. (4) denotes the standard GAN objective

$$\begin{aligned} V(D, \{G_j\}) &= \mathbb{E}_{\{\boldsymbol{x}_j\}\sim P_{\mathrm{data}}}[\log D(\{\boldsymbol{x}_j\})] \\ &\quad + \mathbb{E}_{\{\boldsymbol{c}_j\sim P(\boldsymbol{c}_j)\},\{z_j\sim P(z_j)\}}[\log(1 - D(\{\hat{\boldsymbol{x}}_j\}))] \end{aligned} \tag{5}$$

where $\hat{\boldsymbol{x}}_j = G_j(\boldsymbol{c}_j, z_j, \hat{\boldsymbol{x}}_{\mathrm{Par}(j)})$. The second term in Eq. (4) is the lower bound of the mutual information between the latent codes and the synthesized designs

$$\begin{aligned} L_I(\{G_j\}, \{Q_j\}) = \sum_{j=1}^{n} \mathbb{E}_{\boldsymbol{c}_j\sim P(\boldsymbol{c}_j), \boldsymbol{x}_j\sim G_j(\boldsymbol{c}_j, z_j, \hat{\boldsymbol{x}}_{\mathrm{Par}(j)})}[\log Q_j(\boldsymbol{c}_j|\boldsymbol{x}_j, \hat{\boldsymbol{x}}_{\mathrm{Par}(j)})] \\ + H(\boldsymbol{c}_j) \end{aligned} \tag{6}$$

4 Experiments

To demonstrate the performance of our model, we built six datasets with different ground-truth interpart dependencies. We train the proposed network on these datasets and evaluate the generative performance, constraint satisfaction, and the latent space property through qualitative and quantitative measures.

4.1 Experimental Setup

4.1.1 Dataset. We created the following datasets, as shown in Fig. 4:

(1) AH: An 2D airfoil (part A) with a hole (circle, part B) *inside*.
(2) AHH: An 2D airfoil (part A) with two *nonintersecting* holes (parts B and C) *inside*. The centers of the two holes lie on a *horizontal* line.
(3) SE: A superformula [40] (part A) with a *concentric* ellipse (part B) *inside*.
(4) SEoEi: A superformula (part A) with two ellipses *inside*. The second ellipse (part C) is also *inside* the first one (part B). All three shapes are *concentric*.
(5) SEiEo: A superformula (part A) with two ellipses *inside*. The second ellipse (part C) is *outside* the first one (part B). All three shapes are *concentric*.[2]
(6) SCC: A superformula (part A) with two *tangent* circles—one (part B) inside and the other (part C) outside.

In addition, we control the part dependency graph's depth and branching factor by adding more circles/ellipses to SCC/SEoEi. This results in six extra datasets—**S**, **SC**, **SCCC**, **SCCCC**, **SEEE**, and **SEEEE**. Here, the dataset of superformulas (S) is used as a baseline where no interpart dependency is presented.

Specifically, the airfoil shapes are from the UIUC airfoil coordinates database,[3] which provides the Cartesian coordinates for nearly 1600 airfoils. Each airfoil is reparametrized and represented with 64 Cartesian coordinates, resulting in a 64×2 matrix.

Though targeted for real-world applications, the airfoils may not be a perfect experimental dataset to visualize the latent space, because the ground truth intrinsic dimension (ID)[4] of the airfoil dataset is unknown. Thus, we create another synthetic dataset using superformulas and ellipses, the IDs of which are controllable. The superformula is a generalization of the ellipse [40]. We generate superformulas using the equations as follows:

$$\begin{aligned} n_1 &= 10s_1 \\ n_2 = n_3 &= 10(s_1 + s_2) \\ r(\theta) &= (|\cos\theta|^{n_2} + |\sin\theta|^{n_3})^{-(1/n_1)} \\ (x, y) &= (r(\theta)\cos\theta, r(\theta)\sin\theta) \end{aligned} \tag{7}$$

where $s_1, s_2 \in [0, 1]$, and (x, y) is a Cartesian coordinate. For each superformula, we sample 64 evenly spaced θ from 0 to 2 and get 64 grid-point Cartesian coordinates. Equation (7) shows that we can control the deformation of the superformula shape with s_1 and s_2. Thus, the ground truth ID of our superformula dataset is two. For further details on how to modify the superformula parameters to adjust the ID, complexity, and number of manifolds, see Ref. [1].

The other shapes (e.g., circles and ellipses) are also represented using 64 Cartesian coordinates. The ground truth ID of ellipses in SE, SEoEi, and SEiEo is two, since we fix their centers and only change their semi-major axis and semi-minor axis lengths. The ground truth ID of circles in SCC is also two, since they can change their radii and move tangential to the superformulas. Part B (circle) in AH and AHH has a ground truth ID of three, as both their centers and radii can change while part C in AHH has a ground truth ID of two, since the y-coordinate of its center is fixed. Figure 5 shows samples drawn from each dataset.

[2]Note that the assembly of SEiEo is the same as SEoEi, but since we synthesize its parts in a different order, the interpart dependency changes (see Fig. 4). We create this dataset for comparing different interpart dependencies rather than simulating practical use case.

[3]https://m-selig.ae.illinois.edu/ads/coord_database.html

[4]The intrinsic dimension is the minimum number of variables required to represent the data. It indicates the degrees of freedom we have to control the shape and the position.

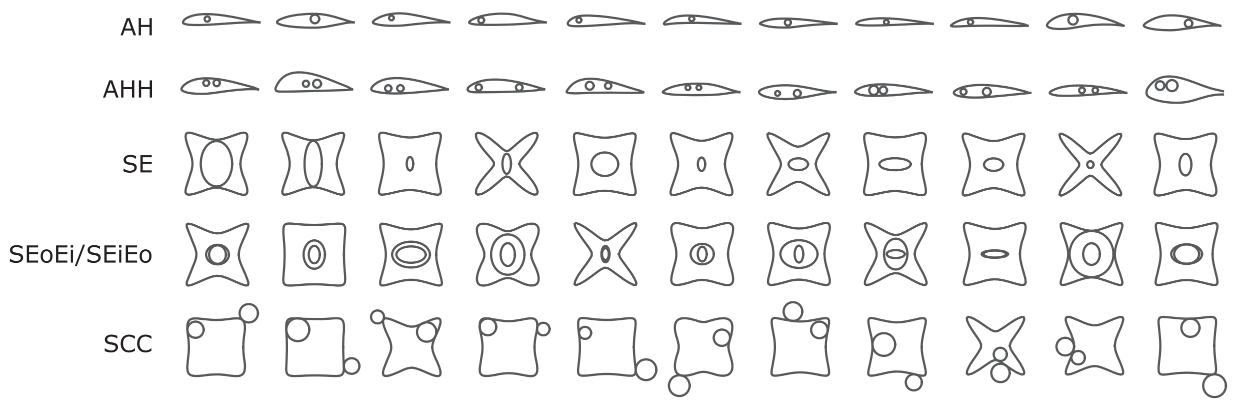

Fig. 5 Samples drawn from datasets

For each dataset, we run experiments with sample sizes ranging from 500 to 10,000.

4.1.2 Network and Training. We implement our airfoil/superformula generators by adopting the generator architecture of the BézierGAN [41,42]. The circle/ellipse generators first generate shape parameters (e.g., the center coordinates and the radius for a circle) and then convert them into grid-point coordinates using corresponding parametric functions. For a generator with parent parts as inputs, we use an encoder to convert each parent part into a feature vector before concatenating all inputs and feeding them into the generator. Similarly, we use an encoder to convert each generated part into a feature vector before feeding them into the auxiliary networks and the discriminator. The use of encoders reduces the model complexity by reducing the input dimension of the generators, the auxiliary networks, and the discriminator. But one has to carefully choose the feature vector dimensions to avoid loss of information from dimensionality reduction.

At training, we sample the latent codes from uniform distribution $\mathcal{U}(0, 1)$ and the noise inputs from normal distribution $\mathcal{N}(0, 0.25)$. The hyperparameter λ in Eq. (4) was set to 0.1 in all experiments. The network was optimized using Adam [43] with the momentum terms $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The learning rates were set to 0.0001. The total number of training steps was 100,000. The batch size was 32. Interested readers who wish to reproduce our exact architectures, hyperparameters, and training procedures are directed to our code located on GitHub.[5] The training procedure is summarized in Algorithm 1.

Algorithm 1 Train HGAN for designs with n parts

```
1: ▷ $T$: number of training steps
2: ▷ $m$: batch size
3: ▷ $X$: training set
4: procedure Train ($T$, $m$)
5:     for $t = 1{:}T$ do
6:         Sample $\{x^1, \ldots, x^m\}$ from $X$, where $x^i = [x^i_1, \ldots, x^i_n]$, $i = 1, \ldots, m$
7:         for $j = 1{:}n$ do
8:             Sample $\{c^1_j, \ldots, c^m_j\}$ from a uniform distribution
9:             Sample $\{z^1_j, \ldots, z^m_j\}$ from a normal distribution
10:        end for
11:        ▷ Based on Eqs. (4)–(6):
12:        Train $D$ using $\{x^1, \ldots, x^m\}$, fixing $G_1, \ldots, G_n$
13:        $\hat{x}^i := [G_1(c^i_1, z^i_1, \hat{x}^i_{Par(1)}), \ldots, G_n(c^i_n, z^i_n, \hat{x}^i_{Par(n)})]$
14:        Train $Q_1, \ldots, Q_n$ and $D$ using $\{\hat{x}^1, \ldots, \hat{x}^m\}$, fixing $G_1, \ldots, G_n$
15:        Train $G_1, \ldots, G_n$, fixing $Q_1, \ldots, Q_n$ and $D$
16:    end for
17: end procedure
```

[5]https://github.com/IDEALLab/hgan_jmd_2019

We used TENSORFLOW [44] to build the networks. We trained our networks on a Nvidia Titan X GPU. For each experiment, the training process took around 2.2 h for SE and AH and 3 h for other examples. The inference took less than 10 s.

5 Results and Discussion

We evaluated the performance of our trained generative models using both visual inspection (Figs. 6–10) and quantitative measures (Fig. 11). We analyze the effect of sample size and problem complexity on those quantitative performance metrics.

5.1 Visual Inspection. The captured latent spaces for different examples are visualized in Figs. 6–9. All these plots are generated using a sample size of 10,000. The results show that each child latent space adjusts itself according to its parent part so that the sizes/positions of child parts matches their parent parts. This indicates that the child generator figures out the implicit constraints encoded in data. The latent spaces capture major shape variations and show consistent shape change. For example, in Fig. 6, the outer ellipse (middle subplot) has a consistently decreasing width from left to right and increasing height from top to bottom. Interestingly, Fig. 8 shows that the circles (middle and right subplots) change in the latent space according to a polar coordinate system, instead of a Cartesian coordinate system like in other examples. For example, the inner circle's radius decreases with the radial coordinate, and its position moves with the angular coordinate. This behavior is interesting because we did not explicitly encode this polar-coordinate representation into the HGAN architecture—rather, the HGAN automatically learns that such a representation is appropriate for this constraint. Figure 10 shows that when we removed the mutual information lower bound L_I in Eq. (4), latent spaces either failed to capture major shape variation or became entangled and inconsistent.

In the AHH example (Fig. 9), unfeasible synthesized designs occur when the two holes intersect. The figure shows that while the second hole moves from one side of the airfoil to the other side, it has to pass through a narrow unfeasible region. This unfeasible region cuts off the latent space, but the generator ignores this fact and learns a continuous latent space by interpolating designs inside the unfeasible region. The small volume of this unfeasible region may cause the discriminator to ignore it. In other words, the generator is willing to take the minor loss incurred by this small region of the infeasible design space in order to avoid making the latent space representation more complicated. To solve this problem, we can perform adaptive sampling in the latent space to more accurately identify the feasible region(s) [2,5].

5.2 Constraint Satisfaction. We measure the precision of constraint satisfaction by computing the proportion of feasible designs among all the synthesized designs. We call this metric the

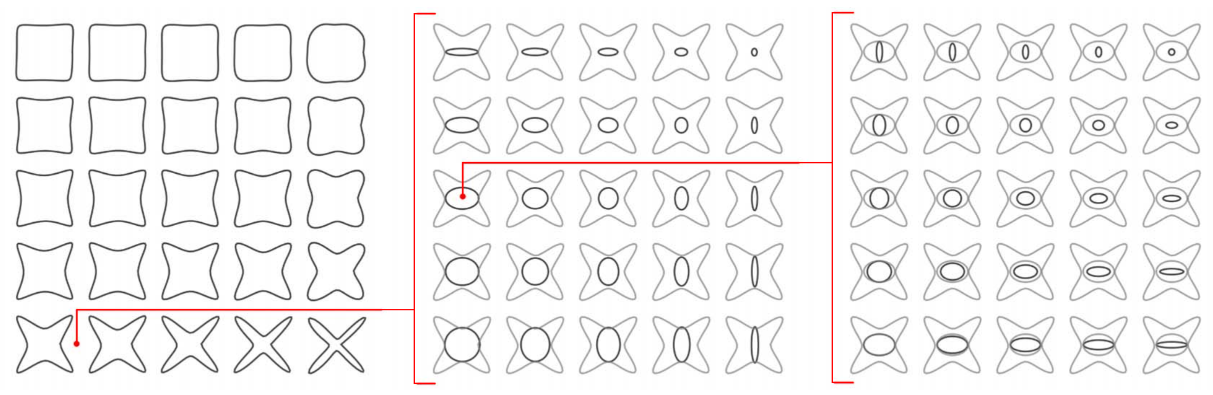

Fig. 6 Latent space visualization of the SEoEi example. Left: synthesized superformulas in a 2D latent space; middle: synthesized outer ellipses in a 2D latent space conditioned on a random superformula; right: synthesized inner ellipses in a 2D latent space conditioned on a random outer ellipse.

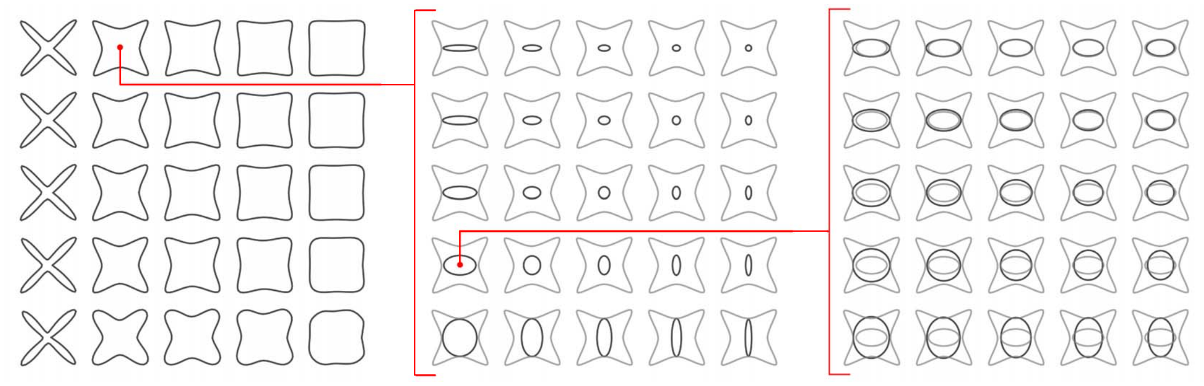

Fig. 7 Latent space visualization of the SEiEo example. Left: synthesized superformulas in a 2D latent space; middle: synthesized inner ellipses in a 2D latent space conditioned on a random superformula; right: synthesized outer ellipses in a 2D latent space conditioned on that same superformula and a random outer ellipse.

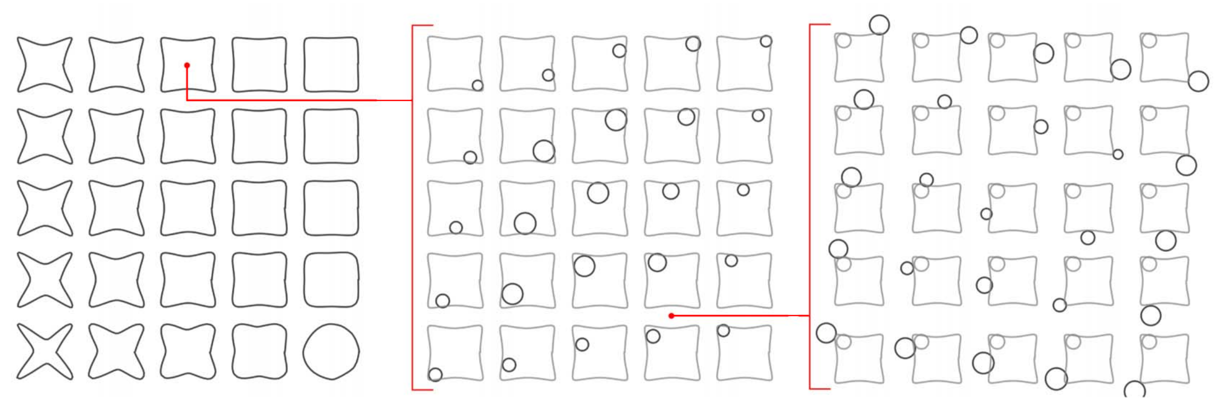

Fig. 8 Latent space visualization of the SCC example. Left: synthesized superformulas in a 2D latent space; middle: synthesized inner circles in a 2D latent space conditioned on a random superformula; right: synthesized outer circles in a 2D latent space conditioned on that same superformula. Interestingly, the HGAN automatically learns a polar-coordinate representation for the tangent constraint.

constraint satisfaction score (CSS). Specifically, different constraints define feasibility in different examples:

(1) AH: Each point on the hole should be inside the airfoil.
(2) AHH: (i) Each point on both holes should be inside the airfoil; (ii) the centers of the two holes should have a vertical distance of less than 0.01 (alignment constraint);[6] and (iii) the distance of the two centers should be larger than the sum of the two radii (nonintersection constraint).
(3) SE: (i) Each point on the ellipse should be inside the superformula and (ii) the distance between the origin and the center of the ellipse should not exceed 0.01 (concentric constraint).[7]

[6]All designs are rescaled such that the airfoils and the superformulas have unit widths.

[7]For all the examples, we assume that the superformula is centered at the origin.

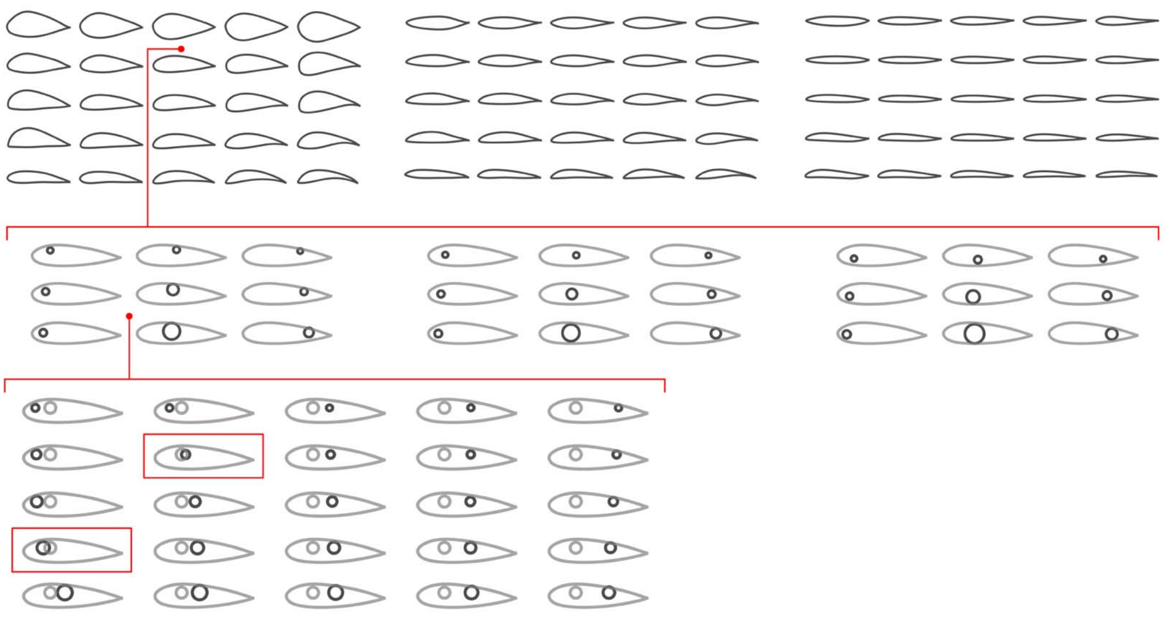

Fig. 9 Latent space visualization of the AHH example. Top: synthesized airfoils in a 3D latent space (visualized by multiple slices of 2D latent spaces); middle: synthesized holes in a 3D latent space conditioned on a random airfoil; bottom: synthesized holes in a 2D latent space conditioned on that same airfoil and another random hole. Unfeasible synthesized designs occur when the two holes intersect (indicated by the boxes).

(4) SEoEi: (i) Each point on both ellipses should be inside the superformula; (ii) both the semi-major and semi-minor axes lengths of the first ellipse should be larger than the second one; and (iii) for both ellipses, the distance between their centers and the origin should not exceed 0.01 (concentric constraint).
(5) SEiEo: Same as the SEoEi example, except that the first and the second ellipses are swapped.
(6) SCC: For each circle with a center C_o and a radius r, the difference between r and the distance from C_o to the superformula should be less than 0.03 (tangent constraint).

The results on CSS in Fig. 11 show that the tangent constraint in the example SCC is the hardest to learn. But the learning performance improves with a larger sample size. The SE example outperforms SEoEi and SEiEo on CSS, which is expected since the task of learning constraints in SE can be considered as a subtask in SEoEi and SEiEo. This also applies to AH and AHH. It is also expected that SEoEi outperforms SEiEo, since the two are dealing with the same design but the former has fewer dependencies.

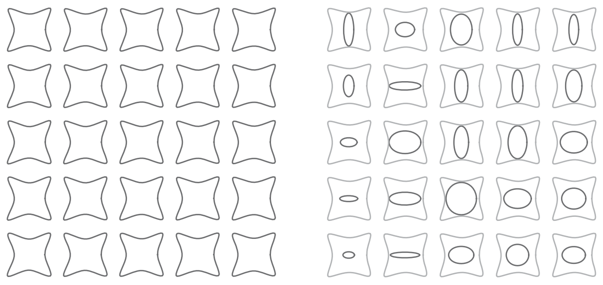

Fig. 10 Latent spaces learned without maximizing the mutual information lower bound. Left: the latent code c failed to capture major shape variation; right: inconsistent shape variation along LSC = 0.562 ± 0.008.

5.3 Distance Between Data and Generator Distributions.

We measure how well our generator approximates the real data distribution by computing the kernel maximum mean discrepancy (MMD) [45] between the data and the generator distribution

$$\text{MMD}^2(P_{\text{data}}, P_G) = \mathbb{E}_{x_d, x'_d \sim P_{\text{data}}; x_g, x'_g \sim P_G}\left[k(\boldsymbol{x}_d, \boldsymbol{x}'_d) - 2k(\boldsymbol{x}_d, \boldsymbol{x}_g) + k(\boldsymbol{x}_g, \boldsymbol{x}'_g)\right]$$

where $k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2/(2\sigma^2)\right)$. A lower kernel MMD indicates that the generator distribution is closer to the data distribution.

The top middle plot in Fig. 11 shows the results of MMD. Here, the SEoEi and SEiEo examples have similar MMD values, which suggests that synthesized designs in both examples have similar perceptual quality, despite the fact that SEiEo requires more model parameters to learn the additional dependency. The MMD plot provides insight into how realistic the generated designs can get as the training sample size changes. In general, MMD first decreases steeply with the sample size and then reaches a plateau at some point. This point indicates the smallest sample size required to reach a "perceptually good" synthesis performance.

5.4 Diversity of Generated Designs.

A common problem in GANs' training is mode collapse, during which the generator only generates a few types of designs to fool the discriminator instead of properly learning the complete data distribution. Therefore, it is important to measure the diversity of the synthesized designs. We use relative diversity (R-Div) to measure the relative level of variability captured by the generator.

We define the diversity of N samples $\boldsymbol{X} \in \mathbb{R}^{d\times N}$ as[8]

$$\text{Div}(\boldsymbol{X}) = \frac{1}{N}\text{trace}(\text{cov}[\boldsymbol{X}, \boldsymbol{X}])$$

[8]Here, each sample is represented as a column vector containing all the coordinates of the design.

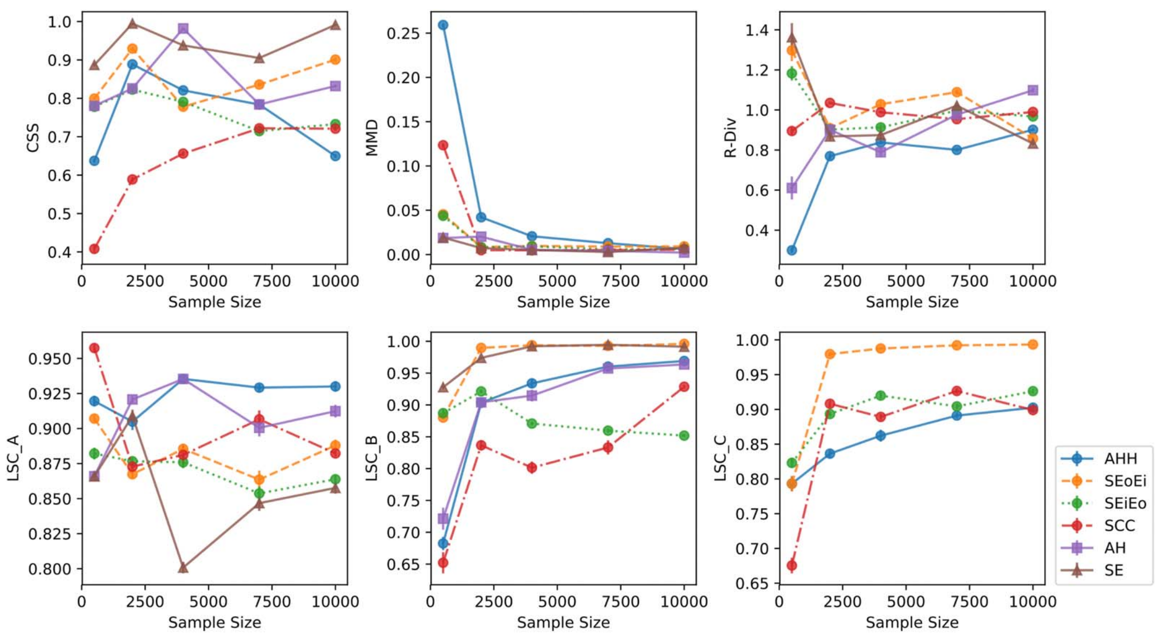

Fig. 11 Quantitative measure of synthesized design quality and latent space properties. LSC_A, LSC_B, and LSC_C denote the LSC for the latent spaces of parts A, B, and C, respectively.

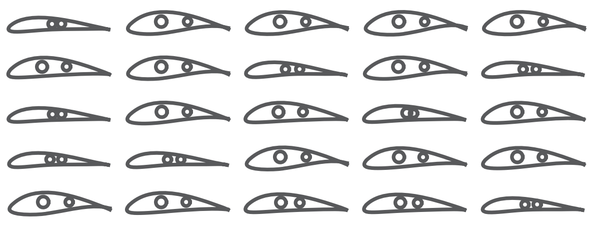

Fig. 12 Mode collapse occurs when the sample size is 500. The diversity is low in these randomly generated designs.

The R-Div metric can then be expressed as

$$\text{R-Div} = \frac{\text{Div}(\boldsymbol{X}_g)}{\text{Div}(\boldsymbol{X}_{\text{data}})}$$

where $\boldsymbol{X}_g$ and $\boldsymbol{X}_{\text{data}}$ denote the set of synthesized designs and designs from the dataset, respectively. A R-Div close to 0.0 means that there is little variation within the synthesized designs, which could be an indicator of mode collapse. A R-Div around 1.0 indicates that the synthesized designs have a similar level of variability with the dataset. Note that high diversity does not always indicate good performance, as there could be unrealistic designs being synthesized, which also contribute to diversity. Thus, we should view this metric in concert with the kernel MMD to determine how well the generator performs.

Figure 11 shows that synthesized designs tend to have high divergence of R-Div when the sample size is small. Particularly, in the AHH example with a sample size of 500, the low R-Div combined with the high kernel MMD indicates the occurrence of mode collapse (Fig. 12). Increasing the sample size stabilized the R-Div and eventually bounded it between 0.8 and 1.1.

5.5 Latent Space Consistency. A desirable latent space has two properties: (1) *disentanglement*: each latent variable is related to only one factor and (2) *consistency*: shapes change consistently along any basis of the latent space. Note that this consistency is evaluated along one direction at a time, since scales along different directions may vary. To the best of our knowledge, existing quantitative measurements for the first property—latent space disentanglement—are supervised, i.e., the ground-truth independent factors causing shape deformation have to be provided [46–48]. The second property is important for latent space design exploration. When searching for designs along a direction in the latent space, optimization algorithms and humans usually prefer if shapes change consistently, such that the objective function over the latent space is less complicated (i.e., has better Lipschitz continuity) and has fewer local optima.

We propose latent space consistency (LSC) as a quantitative measure of how consistently shapes change along any basis of the latent space. Since change from one shape to another can be measured by their dissimilarity, distances between samples along a certain direction in the latent space should be consistent with the dissimilarity between those samples. We use Pearson correlation coefficient to measure this consistency. Algorithm 2 describes how to compute the LSC. The choice of the dissimilarity function d is not central to the overall method. In our experiments, we simply use the Euclidean distance to measure the dissimilarity of designs.

Algorithm 2 Evaluate latent space consistency

```
1: procedure LatentConsistency ($G$, $m$, $n$, $d$)
2:     ▷ $G$: the mapping from a latent space to a design space
3:     ▷ $m$: the number of lines to be evaluated
4:     ▷ $n$: the number of points sampled on each line
5:     ▷ $d$: a dissimilarity function
6:     ▷ $\mathcal{C}$: the latent space
7:     ▷ $\mathcal{X}$: the design space
8:     sum = 0
9:     for $i = 1{:}m$ do
10:        Sample a line $\mathcal{L}$ parallel to any basis of $\mathcal{C}$
11:        Sample $n$ points $\{\boldsymbol{c}^1, \boldsymbol{c}^2, \ldots, \boldsymbol{c}^n\}$ along $\mathcal{L}$
12:        $\{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^n\} := \{G(\boldsymbol{c}^1), G(\boldsymbol{c}^2), \ldots, G(\boldsymbol{c}^n)\}$
13:        $D_{\mathcal{C}} := \{\|\boldsymbol{c}^i - \boldsymbol{c}^j\|\}$, $D_{\mathcal{X}} := \{d(\boldsymbol{x}^i, \boldsymbol{x}^j)\}$, where $i, j \in \{1, ..., n\}$
14:        Compute Pearson correlation coefficient:
               $\rho := \frac{\text{cov}(D_{\mathcal{C}}, D_{\mathcal{X}})}{\sigma(D_{\mathcal{C}})\sigma(D_{\mathcal{X}})}$
15:        sum := sum + $\rho$
16:    end for
17:    Return sum/$m$
18: end procedure
```

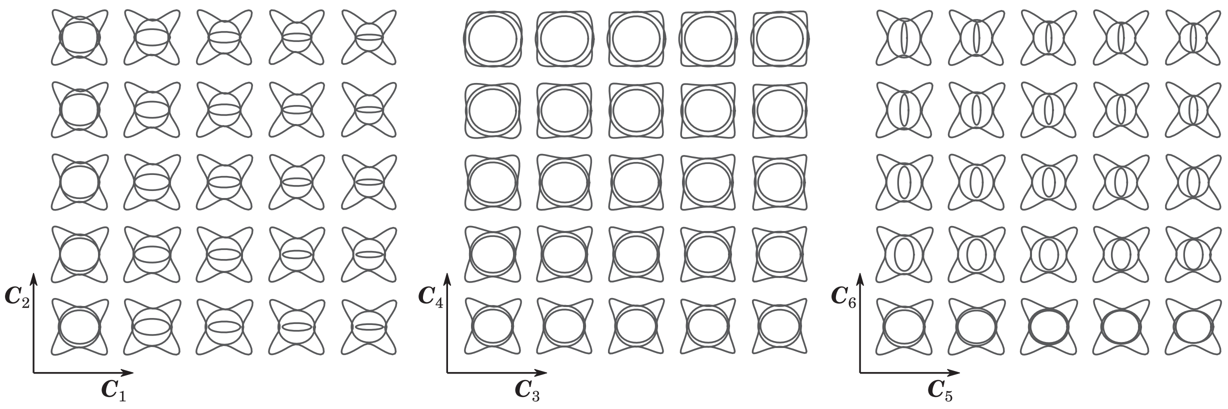

Fig. 13 The latent space learned by a standard InfoGAN. This plot visualizes a 6D latent space of the SEoEi example by showing each two dimensions while setting the latent coordinates of other dimensions to zero. In each dimension, three parts change simultaneously. This shows that the latent space learned by a standard InfoGAN does not disentangle each part's shape variation. In contrast, the results of the HGAN disentangle and separate each part's latent space.

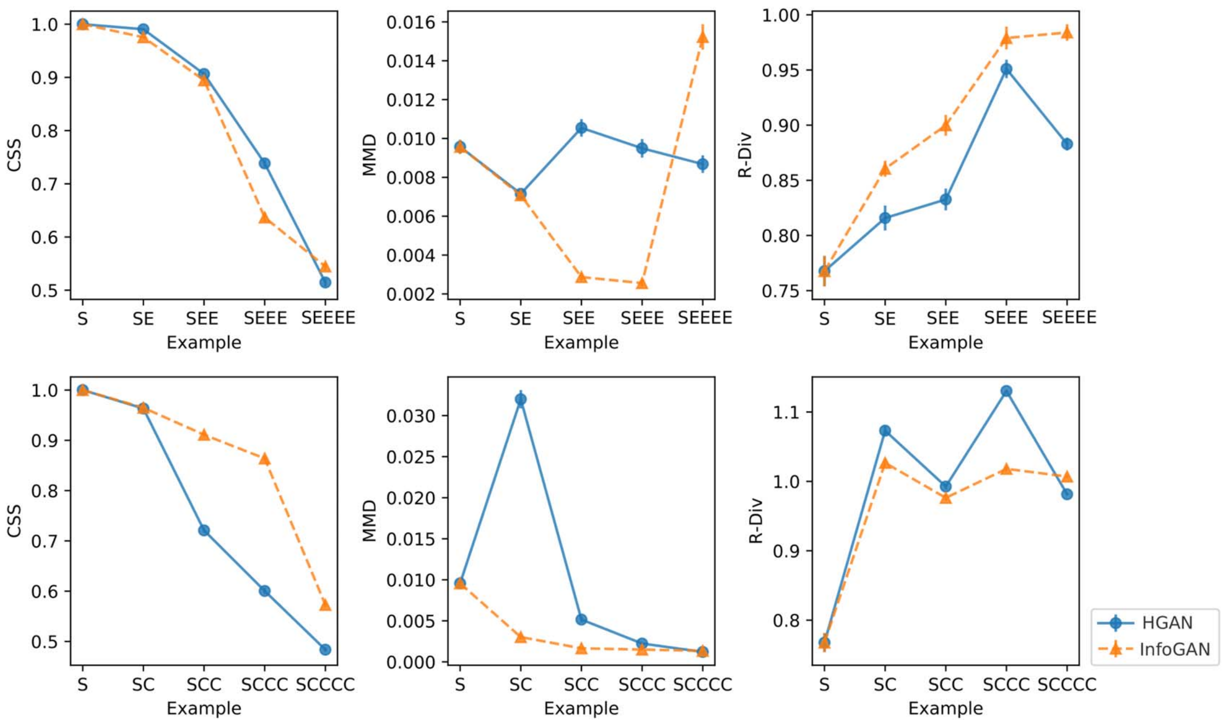

Fig. 14 Effects of depth (top) and branching factor (bottom) on synthesis performance. Here, we denote the example SEoEi as SEE for simplicity.

Both the middle plots in Figs. 6 and 7 show latent spaces with LSCs above 0.9. In contrast, the right plot in Fig. 10 provides a visual example of an LSC of around 0.56. The bottom plots in Fig. 11 show that a larger sample size does not improve LSCs of part A (at least with sample sizes in the range from 500 to 10,000) but improves the LSCs of parts B and C in most cases.

5.6 Effect of Encoding Interpart Dependencies. To further study the effect of encoding interpart dependencies in the GAN, we compared the results of our HGAN with a standard InfoGAN, where there is only one generator and all parts are synthesized from a single latent space. The latent space dimension was set to the sum of all latent space dimensions in the HGAN experiments. Figure 13 visualizes the latent space learned by a standard InfoGAN. It shows that in each latent dimension, three parts change simultaneously, which indicates that the latent space does not disentangle each part's shape variation.

We also study the effects of part dependency graph's depth and branching factor on the model's synthesis performance. The examples S, SE, SEE, SEEE, and SEEEE simulate increasing depths, respectively, while the examples S, SC, SCC, SCCC, and SCCCC simulate increasing branch factors, respectively. The results are shown in Fig. 14. The InfoGAN model is used as a base line where we do not encode interpart dependencies. As expected for both HGAN and InfoGAN, CSS decreases with increased number of parts, since additional parts bring extra constraints. Theoretically, encoding interpart dependencies introduces extra constraints to the model and hence complicates the overall task. However, based on the results, the HGAN shows no significant performance drop comparing to the InfoGAN, except for the CSS when increasing the branching factor and the R-Div when increasing the depth. Note that the MMD values are below 0.05 and do not change notably. The lower CSS or R-Div indicates that HGAN compromises its synthesis precision (i.e., the precision of satisfying constraints) or generator distribution's coverage for disentangling each part's latent space. We also included the training history of HGAN and InfoGAN in the supplementary material on the ASME Digital Collection.

6 Limitations and Future Work

One limitation of our approach is that it is difficult to achieve high precision in satisfying some constraints. This problem exists in all purely data-driven design methods, where there is no process of incorporating restricted constraints in the generative model or validating the constraint satisfaction of the outputs. While we can address this by encoding constraints explicitly in the generative model, this requires us to know these constraints in advance and create hand-coded rules for all types of constraints. For example, to generate concentric ellipses, one can simply fix the center of the second ellipse to have the same coordinate as the first one. However, it may be difficult to incorporate some constraints in the generative model (e.g., the constraint of one part being inside another). Also, it is sometimes hard to even know the type of the exact constraint between parts (e.g., aesthetic preferences when placing handles on a vase). Thus, here, we assume that we have no prior knowledge on the types of constraints, and interpart dependencies are the only knowledge we need for our model. Despite the limitation of purely data-driven design methods, they can be used in the conceptual design stage for exploring a wide range of design alternatives and inspiring novel designs. We can also use validation based on simulation, experiment, or human annotation to exclude infeasible synthesized designs when performing latent space exploration [2,49], which could be an interesting avenue for future work.

Another limitation is that all designs must have the same part dependency graph, which is impractical in some cases. For example, not all tables have four legs; thus, for some designs, their part dependency graphs might miss some nodes. Future study needs to address this situation.

Sometimes the design data are not partitioned into separate components. One possible solution to this problem is to apply unsupervised cosegmentation [6] to partition designs and establish correspondences between common components in different designs. Then, we can use HGAN to learn the latent spaces and generate new designs.

7 Conclusion

We introduced a GAN-based generative model for synthesizing designs with interpart dependencies. It decomposes the synthesis into synthesizing each part conditioned on the corresponding parent part. This also creates a conditioned low-dimensional latent representation that allows accelerated design exploration. This model is built for problems where design space exploration over the latent space has to be staged since the optimal solution of one part depends on the geometry of another. Such models can accelerate design optimization problems, which we are exploring in our future work.

An advantage of neural-network-based generative models (e.g., GANs and VAEs), compared to other dimensionality reduction models (e.g., PCA and GPLVM), is that one can define or regularize latent distributions. Our model adapts InfoGAN's mutual information objective to derive a consistent latent space, where the change of shapes is consistent along any basis of the latent space. This property is desirable in latent space design exploration, as the objective function over the latent space is less complicated and has less local optima.

We also created new benchmark datasets for studying different kinds of interpart dependencies, including type of geometric constraints, depth of hierarchy, and branching factor of parent/child relationships. By using these datasets, we characterized the effects of sample size and part dependencies' complexity (depth and branching factor) on the synthesis performance of our generative model. We also proposed a new evaluation metric for generative models that measure the consistency of shape variation in the latent space. Compared to a standard InfoGAN, the HGAN disentangles each part's latent space at the cost of weakened synthesis precision when the branching of the part dependency increases.

The concept of decoupling the latent space of a design is important for design space exploration in general. It allows separate exploration and synthesis of each part and helps us understand how different constraints control shape variation. Though we use GANs to achieve this goal, the idea of encoding interpart dependencies, learning conditioned generative distributions, and the latent space analysis is also applicable to other generative models. Thus, this paper's techniques lay the foundation for learned generative models to extend to more realistic engineering systems where interpart dependencies are widespread.

Acknowledgment

This work was supported by the Defense Advanced Research Projects Agency (DARPA-16-63-YFA-FP-059; Funder ID: 10.13039/100000185) via the Young Faculty Award (YFA) Program. The views, opinions, and/or findings contained in this article are those of the author and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense (Funder ID: 10.13039/100000005).

References

[1] Chen, W., Fuge, M., and Chazan, N., 2017, "Design Manifolds Capture the Intrinsic Complexity and Dimension of Design Spaces," ASME J. Mech. Des., **139**(5), p. 051102.
[2] Chen, W., and Fuge, M., 2017, "Beyond the Known: Detecting Novel Feasible Domains Over an Unbounded Design Space," ASME J. Mech. Des., **139**(11), p. 111405.
[3] Königseder, C., and Shea, K., 2016, "Visualizing Relations Between Grammar Rules, Objectives, and Search Space Exploration in Grammar-Based Computational Design Synthesis," ASME J. Mech. Des., **138**(10), p. 101101.
[4] Königseder, C., Stanković, T., and Shea, K., 2016, "Improving Design Grammar Development and Application Through Network-Based Analysis of Transition Graphs," Des. Sci., **2**, p. e5.
[5] Chen, W., and Fuge, M., 2018, "Active Expansion Sampling for Learning Feasible Domains in an Unbounded Input Space," Struct. Multidiscipl. Optim., **57**(3), pp. 925–945.
[6] Sidi, O., van Kaick, O., Kleiman, Y., Zhang, H., and Cohen-Or, D., 2011, "Unsupervised Co-segmentation of a Set of Shapes Via Descriptor-Space Spectral Clustering," ACM Trans. Graph., **30**(6), p. 126.
[7] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., 2014, "Generative Adversarial Nets," Advances in Neural Information Processing Systems 27, Montreal, Canada, Dec. 8–13, Curran Associates, Inc., pp. 2672– 2680.
[8] Samareh, J. A., 2001, "Survey of Shape Parameterization Techniques for High-Fidelity Multidisciplinary Shape Optimization," AIAA J., **39**(5), pp. 877–884.
[9] Bellman, R., 1957, *Dynamic Programming*, Princeton University Press, Princeton, NY.
[10] Diez, M., Campana, E. F., and Stern, F., 2015, "Design-Space Dimensionality Reduction in Shape Optimization by Karhunen–Loève Expansion," Comput. Methods Appl. Mech. Eng., **283**, pp. 1525–1544.
[11] Chen, X., Diez, M., Kandasamy, M., Zhang, Z., Campana, E. F., and Stern, F., 2015, "High-Fidelity Global Optimization of Shape Design by Dimensionality Reduction, Metamodels and Deterministic Particle Swarm," Eng. Optim., **47**(4), pp. 473–494.
[12] D'Agostino, D., Serani, A., Campana, E. F., and Diez, M., 2017, "Nonlinear Methods for Design-Space Dimensionality Reduction in Shape Optimization," Proceedings of the Third International Workshop on Machine Learning, Optimization, and Big Data, MOD 2017, Volterra, Italy, Sept. 14–17, Springer International Publishing, pp. 121–132.
[13] Tezzele, M., Salmoiraghi, F., Mola, A., and Rozza, G., 2018, "Dimension Reduction in Heterogeneous Parametric Spaces With Application to Naval Engineering Shape Design Problems," Adv. Model. Simul. Eng. Sci., **5**(1), p. 25.
[14] Raghavan, B., Breitkopf, P., Tourbier, Y., and Villon, P., 2013, "Towards a Space Reduction Approach for Efficient Structural Shape Optimization," Struct. Multidiscipl. Optim., **48**(5), pp. 987–1000.
[15] Raghavan, B., Xia, L., Breitkopf, P., Rassineux, A., and Villon, P., 2013, "Towards Simultaneous Reduction of Both Input and Output Spaces for

Interactive Simulation-Based Structural Design," Comput. Methods Appl. Mech. Eng., **265**, pp. 174–185.
[16] Raghavan, B., Le Quilliec, G., Breitkopf, P., Rassineux, A., Roelandt, J.-M., and Villon, P., 2014, "Numerical Assessment of Springback for the Deep Drawing Process by Level Set Interpolation Using Shape Manifolds," Int. J. Mater. Form., **7**(4), pp. 487–501.
[17] Le Quilliec, G., Raghavan, B., and Breitkopf, P., 2015, "A Manifold Learning-Based Reduced Order Model for Springback Shape Characterization and Optimization in Sheet Metal Forming," Comput. Methods Appl. Mech. Eng., **285**, pp. 621–638.
[18] Viswanath, A., and Keane, A., 2011, "Dimension Reduction for Aerodynamic Design Optimization," AIAA J., **49**(6), pp. 1256–1266.
[19] Qiu, H., Xu, Y., Gao, L., Li, X., and Chi, L., 2016, "Multi-Stage Design Space Reduction and Metamodeling Optimization Method Based on Self-Organizing Maps and Fuzzy Clustering," Exp. Syst. Appl., **46**, pp. 180–195.
[20] Burnap, A., Liu, Y., Pan, Y., Lee, H., Gonzalez, R., and Papalambros, P. Y., 2016, "Estimating and Exploring the Product Form Design Space Using Deep Generative Models," ASME. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Charlotte, NC, Aug. 21–24, American Society of Mechanical Engineers, p. V02AT03A013.
[21] D'Agostino, D., Serani, A., Campana, E. F., and Diez, M., 2018, "Deep Autoencoder for Off-Line Design-Space Dimensionality Reduction in Shape Optimization," 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Kissimmee, FL, Jan. 8–12, p. 1648.
[22] Gmeiner, T., and Shea, K., 2013, "A Spatial Grammar for the Computational Design Synthesis of Vise Jaws," ASME. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Portland, OR, Aug. 4–7, American Society of Mechanical Engineers, p. V03AT03A006.
[23] Talton, J. O., Gibson, D., Yang, L., Hanrahan, P., and Koltun, V., 2009, "Exploratory Modeling With Collaborative Design Spaces," ACM Trans. Graph., **28**(5), p. 167.
[24] Huang, H., Kalogerakis, E., and Marlin, B., 2015, "Analysis and Synthesis of 3D Shape Families Via Deep-Learned Generative Models of Surfaces," Comput. Graph. Forum, **34**(5), pp. 25–38.
[25] Nash, C., and Williams, C. K., 2017, "The Shape Variational Autoencoder: A Deep Generative Model of Part-Segmented 3D Objects," Comput. Graph. Forum, **36**(5), pp. 1–12.
[26] Wu, J., Zhang, C., Xue, T., Freeman, B., and Tenenbaum, J., 2016, "Learning a Probabilistic Latent Space of Object Shapes Via 3D Generative-Adversarial Modeling," Advances in Neural Information Processing Systems 29, Barcelona, Spain, Dec. 5–10, Curran Associates, Inc., pp. 82–90.
[27] Li, J., Xu, K., Chaudhuri, S., Yumer, E., Zhang, H., and Guibas, L., 2017, "Grass: Generative Recursive Autoencoders for Shape Structures," ACM Trans. Graph., **36**(4), p. 52.
[28] Sinha, A., Unmesh, A., Huang, Q., and Ramani, K., 2017, "Surfnet: Generating 3D Shape Surfaces Using Deep Residual Networks," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, July 21–26, pp. 6040–6049.
[29] Chaudhuri, S., Kalogerakis, E., Guibas, L., and Koltun, V., 2011, "Probabilistic Reasoning for Assembly-Based 3D Modeling," ACM Trans. Graph., **30**(4), p. 35.
[30] Kalogerakis, E., Chaudhuri, S., Koller, D., and Koltun, V., 2012, "A Probabilistic Model for Component-Based Shape Synthesis," ACM Trans. Graph., **31**(4), p. 55.
[31] Talton, J., Yang, L., Kumar, R., Lim, M., Goodman, N., and Měch, R., 2012, "Learning Design Patterns With Bayesian Grammar Induction," Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST'12), Cambridge, MA, Oct. 7–10, ACM, pp. 63–74.
[32] Xu, K., Zhang, H., Cohen-Or, D., and Chen, B., 2012, "Fit and Diverse: Set Evolution for Inspiring 3D Shape Galleries," ACM Trans. Graph., **31**(4), p. 57.
[33] Zheng, Y., Cohen-Or, D., and Mitra, N. J., 2013, "Smart Variations: Functional Substructures for Part Compatibility," Comput. Graph. Forum, **32**(2pt2), pp. 195–204.
[34] Fish, N., Averkiou, M., Van Kaick, O., Sorkine-Hornung, O., Cohen-Or, D., and Mitra, N. J., 2014, "Meta-Representation of Shape Families," ACM Trans. Graph., **33**(4), p. 34.
[35] Kingma, D. P., and Welling, M., 2014, "Auto-Encoding Variational Bayes," Proceedings of the 2nd International Conference on Learning Representations, Banff, Canada, Apr. 14–16.
[36] Radford, A., Metz, L., and Chintala, S., 2015, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," Proceedings of the 4th International Conference on Learning Representations, San Juan, PR, May 2–4.
[37] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P., 2016, "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets," Advances in Neural Information Processing Systems, Barcelona, Spain, Dec. 5–10, pp. 2172–2180.
[38] Wang, X., and Gupta, A., 2016, "Generative Image Modeling Using Style and Structure Adversarial Networks," European Conference on Computer Vision, Amsterdam, The Netherlands, Oct. 8–16, Springer, pp. 318–335.
[39] Ohnishi, K., Yamamoto, S., Ushiku, Y., and Harada, T., 2018, "Hierarchical Video Generation From Orthogonal Information: Optical Flow and Texture," Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, Feb. 2–7.
[40] Gielis, J., 2003, "A Generic Geometric Transformation That Unifies a Wide Range of Natural and Abstract Shapes," Am. J. Bot., **90**(3), pp. 333–338.
[41] Chen, W., and Fuge, M., 2018, "Béziergan: Automatic Generation of Smooth Curves From Interpretable Low-Dimensional Parameters," e-print arXiv:1808.08871.
[42] Chen, W., Chiu, K., and Fuge, M., 2019, "Aerodynamic Design Optimization and Shape Exploration Using Generative Adversarial Networks," AIAA 2019 Science and Technology Forum and Exposition, AIAA No. 2019-2351.
[43] Kingma, D. P., and Ba, J., 2015, "Adam: A Method for Stochastic Optimization," Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, May 7–9.
[44] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., 2015, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," Software, tensorflow.org.
[45] Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A., 2012, "A Kernel Two-Sample Test," J. Mach. Learn. Res., **13**(Mar.), pp. 723–773.
[46] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A., 2017, "Beta-VAE: Learning Basic Visual Concepts With a Constrained Variational Framework," Proceedings of the 5th International Conference on Learning Representations, Toulon, France, Apr. 24–26.
[47] Kim, H., and Mnih, A., 2018, "Disentangling by Factorising," Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholm, Sweden, July 10–15, pp. 2649–2658.
[48] Chen, T. Q., Li, X., Grosse, R. B., and Duvenaud, D. K., 2018, "Isolating Sources of Disentanglement in Variational Autoencoders," Advances in Neural Information Processing Systems 31, Montreal, Canada, Dec. 2–8, Curran Associates, Inc., pp. 2610–2620.
[49] Guo, T., Herber, D. R., and Allison, J. T., 2019, "Circuit Synthesis Using Generative Adversarial Networks (GANs)," AIAA 2019 Science and Technology Forum and Exposition, AIAA No. 2019-2350.