Design subspace learning: structural design space exploration using performance-conditioned generative modeling

Renaud Danhaive[a,*], Caitlin T. Mueller[a]

[a] Massachusetts Institute of Technology, Building Technology Program, Department of Architecture, Cambridge, MA 02139, USA
[*] Corresponding author. *Address:* 77 Massachusetts Avenue, Room 5-418, Cambridge, MA 02139, USA. *Tel.* +1 732 693 7447. *Email address:* danhaive@mit.edu (R. Danhaive)

Abstract

Designers increasingly rely on parametric design studies to explore and improve structural concepts based on quantifiable metrics, generally either by generating design variations manually or using optimization methods. Unfortunately, both of these approaches have important shortcomings: effectively searching a large design space manually is infeasible, and design optimization overlooks qualitative aspects important in architectural and structural design. There is a need for methods that take advantage of computing intelligence to augment a designer's creativity while guiding—not forcing—their search for better-performing solutions. This research addresses this need by integrating conditional variational autoencoders in a performance-driven design exploration framework. First, a sampling algorithm generates a dataset of meaningful design options from an unwieldy design space. Second, a performance-conditioned variational autoencoder with a low-dimensional latent space is trained using the collected data. This latent space is intuitive to explore by designers even as it offers a diversity of high-performing design options.

Keywords: deep generative modeling, latent space, latent variable, variational autoencoder, design space, computational design

Highlights

- There is a need to balance performance and diversity in design space exploration.
- High-dimensional spaces are hard to explore without resorting to optimization.
- Deep latent learning (VAE) can usefully compress high-dimensional design spaces.
- Performance-driven sampling yields better latent spaces at less computational cost.
- A two-dimensional latent space is a natural interface for design exploration.

1 Introduction

Computational design offers structural designers ways to explore large arrays of options parametrically generated from formalized design spaces. Design spaces with a large number of parameters are appealing because they have the potential to yield solutions that are unexpected yet high-performing. Unfortunately, they are also tedious to effectively explore through manual means, and human cognition is not effective at processing high-dimensional information. A natural solution is thus to use automated optimization procedures. While optimization certainly has a role to play as part of the arsenal of methods available for design exploration, its applicability is limited because it does not effectively account for human input and leaves no room for intuition. In addition, design spaces may be ill-defined and objective functions are sometimes one but many aspects for a human designer to factor in, rendering optimization results close to useless.

1.1 Research objectives and scope

There is a need for methods that allow designers to intuitively explore chaotic design landscapes without resorting to automated procedures, given the importance of human factors in design. This research seeks to address this need by capitalizing on advances in generative modeling and artificial intelligence to help human designers explore large design spaces in more intuitive, performance-driven ways. To do so, it introduces a method that generates and uses design performance data to build high-performance low-dimensional design subspaces which may be explored directly and easily visualized in their entirety.

First, a newly proposed performance-driven sampling algorithm is used to generate a dataset of meaningful—i.e. biased toward high-performance design regions—options from a large, unwieldy design space. Second, these datasets are used to train low-dimensional deep generative models that are intuitive to explore by human designers and offer a diversity of high-performing design options. The models allow designers to freely and flexibly explore design options that attain performance levels prescribed by the designer, without the need to rely on optimization. Instead of replacing human intuition with deterministic, quantitative rules, the computer here acts as a design collaborator that augments the human intellect.

Because of the large amount of data required to train deep generative models, the proposed method is best suited for applications where simulating thousands of solutions is feasible, either thanks to reasonable simulation times or through massive parallel computing. In terms of simulation time, oft-used structural and architectural analyses—such as linear finite element analysis or energy analysis—range between seconds and minutes, meaning that thousands of solutions may be computed in hours or days at the most. For more computationally expensive analyses, the tenets of the proposed method still hold but it is best to substitute exact analyses for simpler ones—for example, ones that use coarser analysis resolution—or use fast data-driven models. Other than constraints on simulation time, this research applies to any design problem where performance must be considered alongside non-quantifiable characteristics.

1.2 Design space transformation and reduction

Some of the most recent advances in design space exploration powered by a surge of interest in data-driven algorithms focus on the transformation of high-dimensional design spaces into lower-dimensional representations. The established goal of such techniques is to reduce the dimensionality of the original space through visualization or variable transformation or both. When used for visualization, dimensionality reduction helps designers make sense of the design space in formats, 2D maps for example, that clearly emphasize patterns in the design space. In that regard, self-organizing maps [1], [2] have become popular as a means to organize design samples on a two-dimensional plane [3]–[5].

Variable transformation explicitly looks to build meaningful mappings from a new reduced set of new variables to the original variables, such that designers can control a small number of super-variables. The common denominator to these methods is that they are unsupervised, i.e. they do not directly rely on the objective function values. Rather, the objective function is used indirectly to gather the data, as a means to bias collected samples towards well-performing regions of the design space. The hope is that the structure of these regions can be uncovered later on through unsupervised learning. Brown and Mueller [6], for example, use optimization run histories as datasets for which they compute the principal components. The extracted principal directions may then be used as new, composite variables—the first of which is likely to represent the direction of steepest change of the objective function. A different, yet related area of research seeks to objective function as a design variable of sorts through the use of inference. Conti, for example, employs Bayesian networks to predict the probability that a given variable value will yield a desired objective function value [7].

This research similarly seeks to build embeddings in which design exploration is facilitated, but, compared to previous work, seeks to capture the nonlinear manifold structure of the collected dataset and learn a reduced and continuous latent representation of the high-performance regions that can be conditioned by a designer to meet prescribed performance levels.

1.3 Deep generative modeling

Generative modeling—not to be confused with generative design—is a branch of unsupervised machine learning that seeks to understand data by learning to recreate it. While they apply to diverse fields of study, generative models, such as generative adversarial networks (GAN), have made headlines and captured the popular imagination in particular for their ability to generate images that are nearly indistinguishable from real-world pictures [8].

Given a dataset of observations, generative models are trained to retrieve the probability distribution from which the dataset was drawn. Real-world data often lies on complex manifolds in high-dimensional space. The interest of generative modeling in design is that it may be used to generate previously unseen yet probable designs by learning the structure of that manifold. Unfortunately, sampling new designs from generative models is not necessarily intuitive or controllable, and recent years have seen a sharper focus on latent variable models, which overcome this issue by generating data distributions based on a fixed number of variables whose mapping to the original data space is learned through data.

This research focuses on one class of deep latent generative models: variational autoencoders [9], [10]. Variational autoencoders (VAE) assume that high-dimensional observations in a dataset are drawn from probability distributions defined over latent variables, which may be used to draw new samples by navigating the learned latent subspace. This subspace thus offers a controlled way to generate new data and, in the context of this research, explore new design options. This work uses the conditional variant of VAE [11] to include performance as an explicit input to the design generation.

VAEs consist of two differentiable (neural) networks, one encoding high-dimensional input data into a reduced latent representation and the other decoding back the low-dimensional code into its high-dimensional representation (Figure 1). The two networks are chained and trained to minimize a loss function with a term representing the reconstruction error (MSE) between input data fed to the encoder and the output data decoded by the decoder and a regularization term (Kullback-Leibler divergence). VAEs lead to continuous and smooth latent spaces, which are especially advantageous for design space exploration. By continuous and smooth latent space, we mean that the latent space is decoded onto original space by continuous mappings (no local jumps) that are not noisy and mostly exhibit low-frequency (smooth) variations across the latent space. From the standpoint of mathematical terminology, the mappings are indeed continuous, but they are not necessarily smooth—i.e. they are not necessarily differentiable everywhere—since they may use non-smooth activation functions like ReLU. Here, the concept of smoothness should be understood qualitatively and does not refer to the differentiability concept of smoothness.

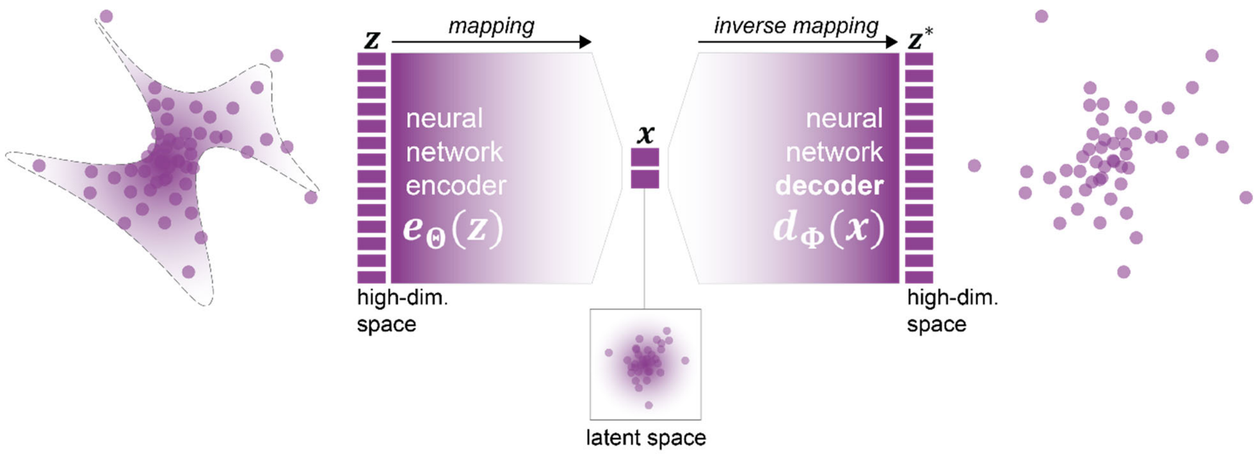

Figure 1: General architecture of variational autoencoders. e_Θ projects a high-dimensional input $\mathbf{z}$ to a compressed representation $\boldsymbol{x}$, which the decoder d_Φ projects back to its original representation as best as possible. Θ and Φ denote their respective network's weights which are optimized during training to minimize a composite loss function combining MSE (reconstruction term) and Kullback-Leibler divergence (which can be seen as a regularization term).

1.4 Deep generative modeling in design

The ability of VAEs to pack complex data distributions into continuous and low-dimensional latent spaces makes them particularly applicable to design applications, mostly thanks to the properties of their latent spaces. For example, Umetani [12] demonstrated that a 10-dimension latent representation may be found for cars based on a dataset of three-dimensional car models. The resulting latent space encompasses large variations and allows novice users to interactively design a car body. In a similar vein, but using image data, Burnap, Liu, Pan, Lee, Gonzalez, *et al.* [13] build a latent design space of two-dimensional automobile bodies. Further away from product design, Carter and Nielsen [14] use a variational autoencoder to build an intuitive design interface for fonts. This previous work demonstrates the usefulness of VAEs to build a low-dimensional design space based on observed real-world data.

Generative adversarial networks (GANs) have similarly been shown to be good candidates for generative design applications, though their latent spaces are often harder to explore. Some of the most impressive results produced by GANs have been images, of human faces in particular [8], and there have been similarly striking advances for design applications. For example, generative adversarial networks have been used to generate voxel-based or point-based three-dimensional models of furniture, cars, and other objects [15]–[17]. GANs and their conditional variations like pix2pix [18] have also been used to generate building floor plans [19] and indoor furniture layouts [20], [21].

This research differs from this previous work in that it establishes workflows, which may be applied systematically for design space exploration for which previously explored data is unavailable. While thousands of car designs may have been observed in the past, a dataset of structures designed for a particular site with specific loads is unlikely to exist. Perhaps closer to this research, Yumer, Asente, Mech, and Kara [22] showed how an autoencoder network may be used to ease the burden of exploring procedural models for non-expert users. However, their method does not incorporate performance in any way but focuses on shape features as a means to differentiate data. Conversely, the proposed method is geometry-agnostic and solely focuses on design parameters and objective function values, which ensures that it can be applied systematically for performance-driven design exploration.

2 Methodology

This section describes the workflow to build subspace representations of the high-performing regions.

In the following, a design option is specified by a n-dimensional vector of design variables $\boldsymbol{x} \in D \subset \mathbb{R}^n$. The domain D formally defines the design space. In this research, D is a bounded domain $\times_i \left[x_{i,min}, x_{i,max}\right](i = 1, \dots, n)$. Each design may be evaluated by a performance metric $f(x)$, which is computationally expensive to query and unknown except when computed for discrete samples. This formalism corresponds to the definitions broadly adopted in most design space exploration research and is useful for describing the algorithms below.

First, we introduce a performance-driven sampling algorithm used to efficiently collect a dataset $\left\{\left(\boldsymbol{x}^{(i)}, f\left(\boldsymbol{x}^{(i)}\right)\right) | i = 1, \dots, N\right\}$ of design samples. Then, we show how this dataset is used to train conditional generative models and how these can be used for design exploration.

2.1 Performance-driven sampling

Training deep latent generative models requires significant amounts of data, which may be collected by sampling through the design space. Because the explicit goal of such models is to uncover a reduced number of latent variables that explain the distribution of the data, they are only effective when the data is non-uniformly distributed. Sampled datasets produced by uniform sampling schemes are thus not adequate to train a latent variable model. Instead, the datasets used for training need to be biased and mostly include those design samples that present desirable attributes, which are assumed to be grouped in specific regions of the design space. In practice, these attributes are measured by some objective function. For the spatial truss example used to illustrate the proposed method (see Figure 5), the structural mass required to resist imposed loads is the objective of interest. If multiple objectives are simultaneously considered, they may be grouped in a composite objective function, an approach used successfully in previous related work [6], or multi-objective sampling schemes may be considered altogether.

The performance bias may be introduced downstream: samples are then obtained through uniform sampling schemes and filtered based on their objective values. If a fine enough sampling resolution may be achieved, this scheme ensures uniform coverage of all high-performance regions. In practice, however, it is typically computationally prohibitive to sample high-dimensional design spaces at a high resolution both because of the curse of dimensionality and the slow simulations typically required in structural and architectural design. An alternative is to sample using schemes that directly incorporate bias. Previous work [6], [23] proposes using optimization histories to collect design samples with a performance bias. Optimization, however, imposes a strong prejudice against regions in the design space that are suboptimal but still may be of interest to designers. In addition, optimization algorithms oversample best-performance regions before reaching convergence. This is true for stochastic optimization algorithms as well, albeit to a lesser extent. Though useful, optimization methods are not designed for sampling.

Sampling is often used to build fast surrogate models substituting for complex and slow engineering analyses, whether this sampling is accomplished through physical experiments as in early surrogate modeling research [24] or using computational fluid dynamics [25] and thermal [26] simulations, and much research has been devoted to devising sampling plans that yield surrogate models that are as accurate as possible for a minimum number of samples [27]–[30]. In contrast to previous work, the proposed sampling algorithm is designed to generate samples from pockets of high-performance more than it is geared toward optimal model accuracy. In performance-driven design, low-performance regions are of little interest, and model accuracy there is less important as a result. In other words, the performance distribution of the samples matters more than the quality of the surrogate model they could be used to build because these samples are generated to train unsupervised learning algorithms or building visualizations used to understand, explore, or generate high-performance design options.

The proposed algorithm is a sequential, model-based sampling scheme, similar conceptually to Bayesian optimization, that uses filtering to introduce performance bias. It starts by building an initial surrogate model based on a limited number of samples evaluated using the true objective function. These samples are used to build an initial surrogate model. This surrogate model is then used to evaluate another set of samples. These samples are filtered using an acceptance criterion which is conditioned on their performance as evaluated by the surrogate model. The filtered-in samples are in turn evaluated using the true objective function and added to the set of collected samples. Based on the samples collected so far, a new surrogate model is built, and the steps above are repeated a prescribed number of times.

The introduced sampling scheme is non-deterministic and uses a tunable sigmoid-like gating function defined in Algorithm 1 to determine which samples to evaluate with the true objective function. The algorithm (see Algorithm 1 for the detailed pseudocode) starts with a low-resolution, unbiased sampling—such as Latin hypercube sampling—of the design space to build a dataset of n_{init} samples $D = \{(\boldsymbol{x}_i, y_i = f(\boldsymbol{x}_i)), i = 1, \dots, n_{init}\}$ where $\boldsymbol{x}_i$ is the design vector of sample i and y_i is its corresponding score computed with f, the design performance function (calls to f are assumed to be slow). This initial dataset is used to build a surrogate model f^* that is hoped to approximate the function f as well as possible, but, given the limited sample size, it is expected to be flawed. Once the surrogate model is built, the design space is sampled again, but, this time around, the samples are evaluated using f^* instead. These evaluations are cheap and fast. Based on their predicted performance, designs are filtered using a sigmoid-like probabilistic gate with a user-specified growth rate g and a performance threshold p. Accepted designs are then evaluated using the true objective function f, and they are added to the dataset D. The augmented dataset is then used to build an updated surrogate model f^*, and this process is repeated $(n_{steps} - 1)$ times. It is worth noting that the growth rate is multiplied by a factor of $1 + \beta$ in each subsequent iteration, where $\beta \geq 0$ is usually small and allows to progressively increase the growth rate as the uncertainty of the surrogate model decreases.

Algorithm 1: Sequential performance-gated sampling

Input:

- $\Omega = \times_i \left[x_{i,min}, x_{i,max}\right](i = 1, \dots, d) \subset \mathbb{R}^d$, the design space,
- $f: \Omega \to \mathbb{R}$, the true objective function,
- n_{init}, the initial number of samples,
- N, desired number of samples in addition to initial samples
- n_{steps}, the number of sampling steps,
- $p \in]0,1[$, the performance threshold,
- $g > 0$, the growth rate,
- $\beta \geq 0$, the increase rate for g

Output:

- Dataset $D = \{(\boldsymbol{x}_i, y_i = f(\boldsymbol{x}_i))\}$ of generated design samples

Initialization:

- Sample n_{init} initial designs in Ω using LH sampling.
- Evaluate designs using f to build dataset $D = \{(\boldsymbol{x}_i, y_i = f(\boldsymbol{x}_i)), i = 1, \dots, n_{init}\}$.
- Train surrogate model f^* based on D.

for i in $[0, \dots, n_{steps} - 1]$ **do**

- Initialize empty set $D_{step} \leftarrow \{\}$.
- Set $g_{step} \leftarrow g * (1 + \beta)^i$.
- **while** $|D_{step}| < \frac{N}{n_{steps}}$ **do**
  - sample $\frac{N}{n_{steps}}$ designs $\boldsymbol{x_j}$ $\left(j = 1, \dots, \frac{N}{n_{steps}}\right)$ in Ω using LH sampling,
  - evaluate design scores $y_j^* = f^*(\boldsymbol{x_j})$ using surrogate model f^*,

- compute p-values $p_j = \frac{\left|\left\{y_j^* \le y_k^*, k=1,\ldots,\frac{N}{n_{steps}}\right\}\right| - 1}{N-1}$ for each design,
- compute acceptance probabilities $\alpha_j = \text{gate}_{g_{step},p}\left(1 - p_j\right)$ for each design,
- accept design j with probability α_j, evaluate using true objective function f, and add to D_{step}: $D_{step} \leftarrow D_{step} \cup \left\{(\boldsymbol{x}_j, y_j = f(\boldsymbol{x}_j)) \middle| Ber(\alpha_j) = 1,\ j = 1,\ldots,\frac{N}{n_{steps}}\right\}$, where Ber is the Bernoulli distribution.

- **end while**
- Add samples generated and accepted in this step to D: $D \leftarrow D \cup D_{step}$
- Train surrogate model f^* based on D.

end for
return D

Where

$$\text{gate}_{g,p}(x) = \frac{\sigma_{g,p}^*(x) - \sigma_{g,p}^*(0)}{\sigma_{g,p}^*(1) - \sigma_{g,p}^*(0)} \text{ with } \sigma_{g,p}^*(x) = \begin{cases} \sigma_g\left(\frac{x-p}{1-p}\right), & if\ x > p \\ \sigma_g\left(\frac{x}{p} - 1\right), & otherwise \end{cases} \text{ and } \sigma_g(x) = \frac{1}{1+e^{-gx}}$$

The role of the probabilistic performance gate is to balance exploration and exploitation of the initial surrogate model as well as the ones built at every subsequent step. Since the initial surrogate model is built with a limited number of samples, it is not expected to be very accurate in many regions of the design space. However, it is used to evaluate the next batch of samples because it is extremely cheap to query compared to the true objective function.

If the sample filtering used a hard threshold and was completely deterministic, there would be a risk that, based on the incomplete picture provided by the surrogate model, only samples in a narrow area may be accepted. These would then be evaluated with the true objective function and added to the initial set of samples to train a new surrogate model, which, compared to the first one, will have only gained information about that narrow area. This may cause the next sampling steps to drill down (i.e. exploit the model) in that one area without exploring other wells of performance that may have been missed by the initial sampling. The probabilistic gating strategy solves this problem by allowing samples that are predicted to be performing worse than the specified performance threshold to be accepted, albeit with a lesser chance. This results in a wider exploration of the design space and smaller odds that good regions are missed. How much weight is given to exploration versus exploitation is controlled by the growth rate g, as is explained in the previous section: the larger the growth rate, the more onus is put on exploitation, and vice-versa. In fact, if $g \to \infty$, the gate is a step function and the filtering is fully deterministic.

As the algorithm progresses, more samples are collected, and the quality of the surrogate model improves and hopefully identifies regions of good performance correctly. When it does, it makes sense to put more weight on exploitation. In other words, the first sampling steps require more exploration while the later ones demand more exploitation. To achieve this, an additional parameter β introduced and is used to increase the growth rate at each new step by multiplying its previous value by $1 + \beta$. For example, for $\beta = 0.2$, a starting growth rate of 5 translates into a growth rate of about 31 after 10 iterations.

The sampling algorithm depends on 6 hyperparameters: the number of initial samples n_{init}, the minimum number of desired samples, the number of sampling steps, the percentile threshold, the growth rate g, and the parameter β. The first four hyperparameters are readily interpretable and may be chosen with reasonable engineering judgment. The number of initial samples may be increased or decreased depending on the complexity of the objective function for which designers generally have a good working intuition. The choice of the number of desired samples would likely be based on the requirements of the data-driven method they may be used (in our case, it is reasonable to assume that a deep neural network will require thousands of samples to effectively train).

Choosing the number of sampling steps essentially amounts to choosing how many surrogate models will be built throughout the sampling process. Choosing as many steps as the desired number of samples would mean building a new surrogate model for every new sampled design, surely an inefficient strategy since any single point does not contribute so much new information as to warrant training a new model, so keeping the number of steps relatively low is reasonable. This is confirmed by experimental results discussed in Section 3.1 which demonstrate that the quality of collected samples quickly improves only after a few steps. The number of sampling steps and the minimum number of desired samples constitute rigid stopping criteria that are commonly used for sampling algorithms. Convergent behavior is clearly observed in experimental results, especially at the distribution level (Figure 7), but basing a stopping criterion on a relative metric like relative convergence of sample mean would not be practical because the algorithm's primary objective is to provide a desired number of high-quality samples. The percentile threshold explicitly controls the targeted percentile of the sampled designs compared to the performance distribution of a uniformly sampled population of designs.

The influence of the last two hyperparameters is also significant as shown experimentally in Section 3.2. The growth rate g controls how lax the performance filtering is at each step of the sampling process. Because trust in the surrogate model is low at the beginning of sampling, using a relatively low growth rate during the first steps of the algorithm prevents it from getting stuck in a limited region of the design space. However, as the algorithm progresses, the surrogate model improves and the stringency of filtering can be increased. To do so, the initial growth rate is progressively increased at each sampling step with a rate controlled by the parameter β. Figure 8 shows the influence of g and β on the algorithm's progress on the long-span roof case study used to illustrate this research.

One of the objectives of the proposed sampling algorithm is to be able to control the diversity and the quality (performance) of the generated samples. The trade-off between sample diversity and quality is usually hard to navigate with existing sampling algorithms, but, here, it can be explicitly mediated by the growth rate and the performance threshold. It is also worth noting that, while the probabilistic gating looks to ensure a good balance of exploitation and exploration at each step, additional unfiltered LH samples may be collected at each step to further increase the odds of widespread coverage.

2.2 Performance-conditioned VAE

This research uses a conditional VAE instead of a standard based on two observations. First, conditioning the generative model on performance scores improves the model by helping it encode and decode designs more easily. Instead of having to learn projections that work for all sampled designs at once, the performance-conditioned VAE (Figure 2) can adapt its mappings to different performance values: this facilitates disentangling performance contours that may otherwise not be discernable. Second, the performance condition can be used after the model is trained to control the decoder's output. This is particularly useful when designers are looking to trade-off performance for other qualitative design attributes: they can start by generating high-performance design options and can progressively relax their performance condition to explore suboptimal solutions that may fulfill unformulated objectives.

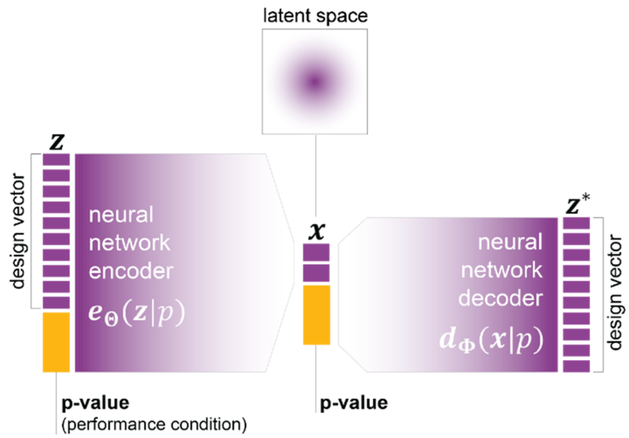

Figure 2: Performance-conditioned VAE. The performance condition is fed to both the encoder and the decoder.

2.2.1 Performance condition: the p-value

The performance-conditioned VAE (PVAE) is trained on a dataset of design samples (represented by their design vectors) and their corresponding performance scores evaluated based on engineering simulation results. However, absolute performance values are not directly used. Instead, we reuse the concept of the p-value, which was introduced in the last chapter and can be seen as a normalized rank of sorts. Given a set of N design samples $\{(\boldsymbol{x}_i, y_i), i = 1, \dots, N\}$ with design vectors $\boldsymbol{x}_i$ and scores y_i, the p-value of design i is computed as $p_i = \frac{|\{y_j \le y_i, i=1,\dots,N\}|-1}{N-1}$. The definition of the p-value is slightly modified compared to the definition in the sampling algorithm such that lower p-values represent designs with lower performance scores. Again, using this strategy allows to map the scores of all designs used for training the PVAE, which can vary widely in magnitude, to the fixed interval $[0,1]$. In addition, compared to the absolute performance scores, the design p-values are distributed evenly on the unit segment.
Using the p-value as the performance condition also makes exploiting the PVAE more intuitive after it is trained: the performance condition can then be seen as a simple knob or slider that can be tuned from 0 to 1 to generate designs with lower or higher objective values.

2.3 Latent space dimensionality

One of the key decisions to make when building a VAE is to choose the dimensionality of the latent space onto which data will be projected. Latent spaces with fewer dimensions generally extract more meaningful directions of change from the data, but they also incur a larger compression loss, which means that it is harder to reconstruct input data from its latent representations. In the context of design space exploration, this means that lower-dimensional latent spaces are likely to generate less diverse design candidates. On the other hand, the lower the dimensionality of the latent space, the easier it is to explore.

In practice, designers may explore the trade-off between ease of exploration by varying the dimensionality of the latent space. This research chooses to build two-dimensional latent spaces, which heavily compress input data, because it allows for the representation of complex design subspaces as two-dimensional landscapes that are easy to explore and provide designers a global view of the design candidates meeting a prescribed performance level. In the long-span roof case study used here, the latent spaces built displayed good diversity such that the benefits brought about by their low-dimensionality outweighed the diversity trade-off. With that said, the proposed method can be readily applied to latent spaces with higher dimensions that are nonetheless significantly easier to explore than the original.

2.4 Decoding the latent space

Once the PVAE is trained, we can detach the decoder and use it to generate designs by moving through the latent space, whose directions are effectively synthetic and reduced design variables. With a two-dimensional latent space, this navigation can be done through a computer screen by moving a point across a two-dimensional slider. Because the latent space dimensionality is small enough, it is also possible to precompute many design candidates and evaluate their performance to build two-dimensional performance maps (Figure 3) that can be overlaid directly on the latent space.

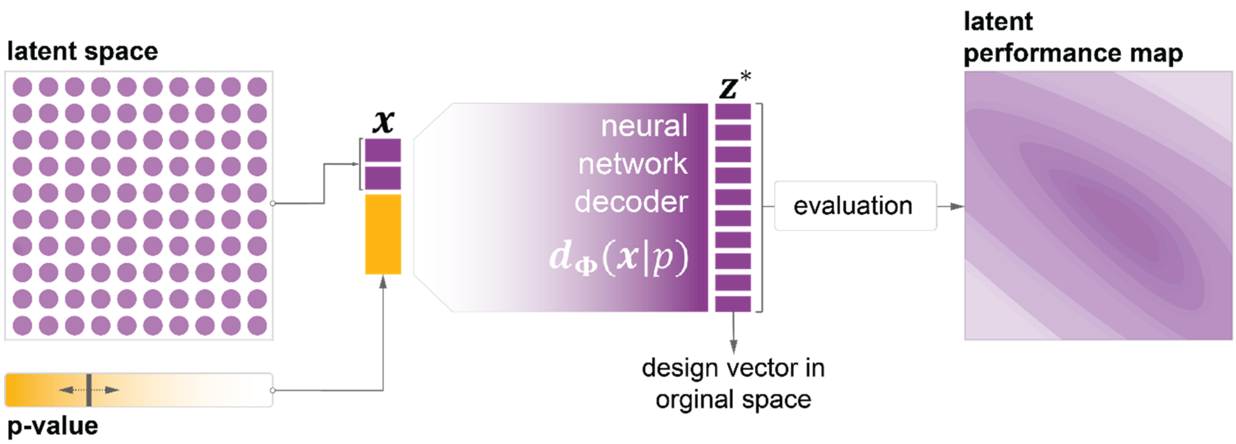

Figure 3: Thanks to the low-dimensionality of the latent space, explorable performance maps can be precomputed by decoding a grid of designs in the latent space and computing their performance scores.

These performance maps can then be navigated with a cursor whose location in the latent space is decoded to generate designs (Figure 4). This allows designers to intently explore diverse, high-performing structures. The p-value constitutes an additional control that allows designers to relax performance requirements and modify the performance landscape to explore other design options.

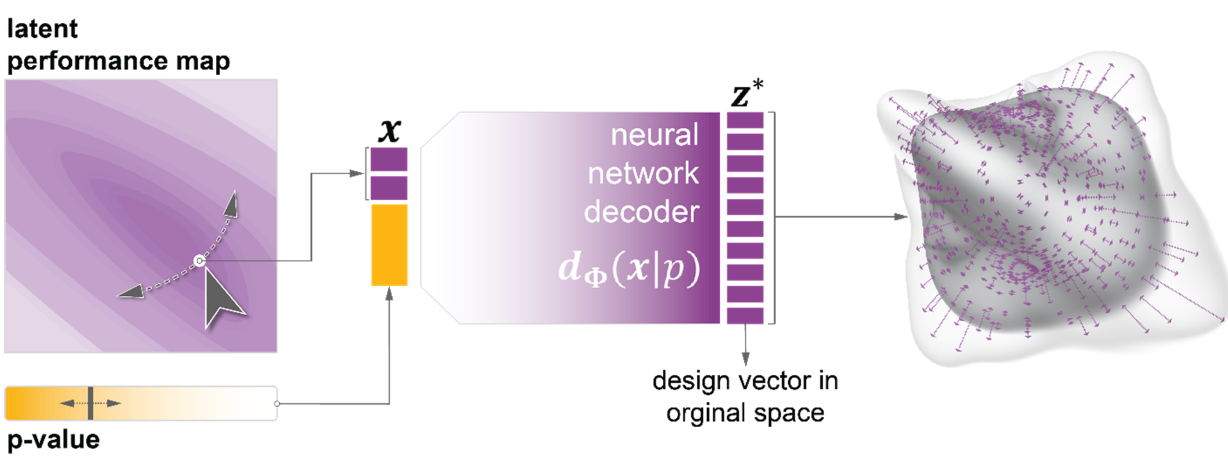

Figure 4: Navigating the latent performance map. A designer can move through the performance map—adjusting it by modifying the p-value—with a cursor, whose location is decoded to generate its corresponding design (represented here by an abstract shape with arrows suggesting the morphing that occurs as the cursor moves across the performance map).

3 Results

To demonstrate the effectiveness of the algorithm on a high-dimensional design space, we use a long-span roof example controlled by 36 design variables (Figure 5). The roof geometry is defined by two surfaces, whose control points can be moved vertically by to modify the depth of the space truss generated between them. Space trusses or space frames are often used in structural design to span long distances: in addition to their inherent structural performance, they can be easily shaped for greater structural efficiency or to conform to an architectural designer's vision or both.

Shaping the space truss in this example is thus an exercise in both structural and architectural design since its shape will affect its structural performance, its architectural form, and the space it spans. The proposed space truss intentionally presents a substantial cantilever to exacerbate the architectural and structural impact of the design variables.

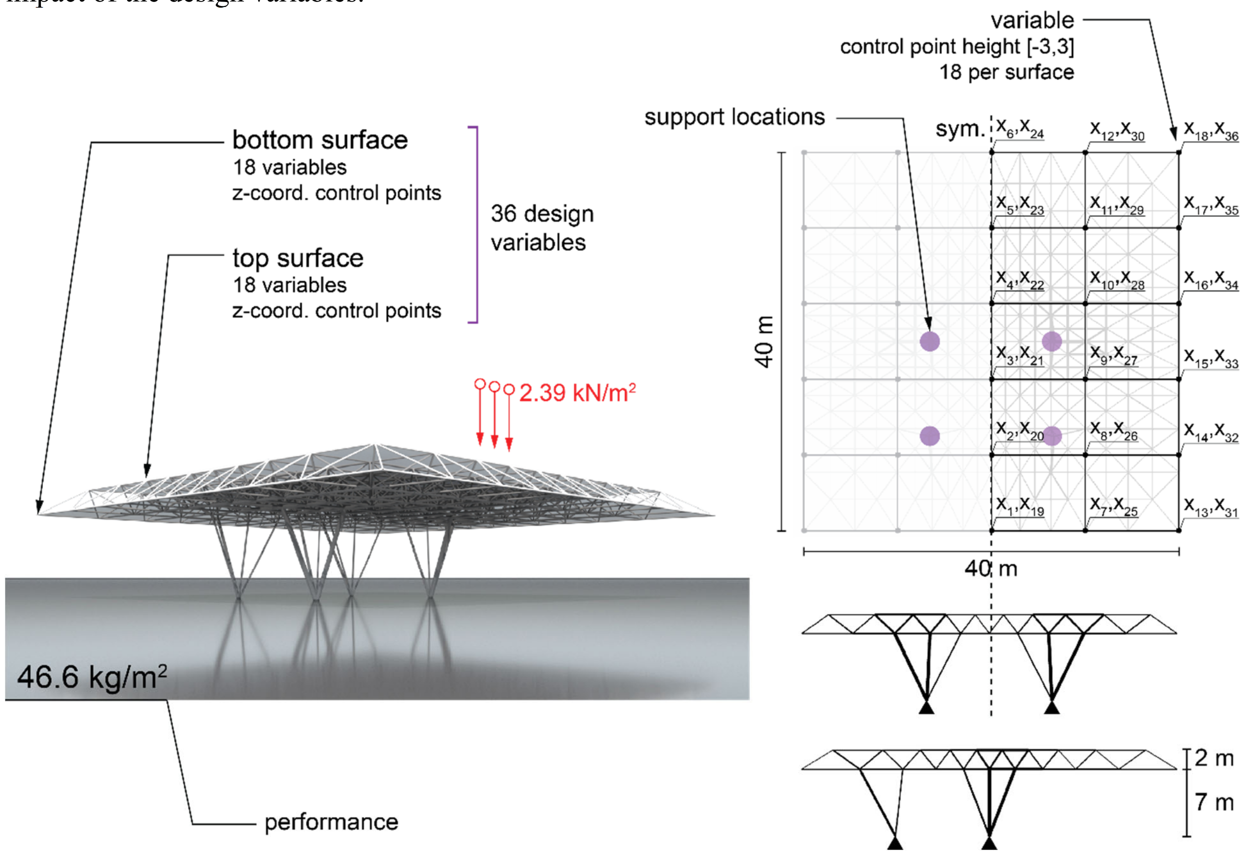

Figure 5: Long span roof example: summary of initial geometry, design variables, and performance measure. Design variables 1 through 18 control the bottom surface and variables 19 through 36 control the top surface.

Controlling the shape of the space truss with two NURBS surfaces (each of degree 2) allows a level of control that can be arbitrarily increased or decreased by introducing additional control points without needing to parametrize the location of every single node in the space truss. For this example, we deliberately use surfaces with a moderate number of control points (25 per surface) to generate a design space that contains diverse and potentially surprising solutions. Because the geometry is constrained to be symmetric and only the z-coordinates of the control points are in play (the roof footprint is fixed), this results in a total of 36 design variables (18 per surface). By parametric design standards, this is a high-dimensional design space, which is likely to contain good solutions both aesthetically and structurally as well as grotesque ones.

3.1 Structural modeling and performance metric

The spatial truss is connected to 4 pin supports by a total of 16 columns and is subject to a load of 50 psf or 2.39 kN/m^2 applied as individual downward point loads on each node of the space frame (17.3 kN per node) in addition to its self-weight. The structure is modeled as a perfect truss (elements only deform axially) built with steel (S355; $\sigma_{yield} = 355\ MPa$) circular hollow sections. Each truss member is sized automatically using the cross-section optimizer of Karamba [31], a structural analysis plug-in. Based on a user-provided catalog of cross-sections, the cross-section optimizer starts by assigning each member with the smallest cross-section possible and analyzes the structure—using linear static FEA—accordingly to obtain its internal forces and displacements. These are then used to resize each member by searching the smallest cross-section possible among the specified catalog that satisfies the strength requirements of the Eurocode EN-1993 (Design of Steel Structures). Because this modifies the self-weight of the structure, the structure is analyzed again to ensure that the cross-sections can resist the updated loads and that the structure does not deform excessively. If these requirements are not met, the process is repeated until they are. For this example, the optimizer sizes the design candidates by picking sections from a catalog of 46 circular hollow tubes with diameters ranging from 10 to 100 cm—very large sections are required to be able to characterize poor-performing designs well—in increments of 2 cm and wall thicknesses equal to 5% of the tube diameters or 20 mm, whichever is smallest. Members are sized against yielding (elastic design) with a safety factor of 1.5 and against buckling with a safety factor of 3. The structure is also subject to a displacement limit of L/300 (=13.3 cm), albeit for a smaller load of 33 psf or 1.58 kN/m^2. The sizing process is used to automatically evaluate design candidates to the amount of material they require to resist the imposed load, which are normalized by the structure's footprint (1600 m^2) for easier interpretation.

3.2 Performance-driven sampling

The proposed performance-driven algorithm is run on the space truss design space for 10 steps with $n_{init} = 1000$, $N = 5000$, a performance threshold $p = 0.8$, a growth rate $g = 2$, and a growth increase rate $\beta = 0.3$. The surrogate model is a support vector regressor with a Gaussian kernel whose parameter γ and C are searched among 5 log-spaced values between 10^3 and 10^{-3} using 3-fold cross-validation.

In total, 6771 samples (including 1000 initial LH samples) are collected at the end of sampling. Figure 6 shows a kernel density estimation (Gaussian kernel; bandwidth=2) of the structural mass—the performance metric of interest—of the samples resulting from the performance-driven sampling and compares it against the same estimation obtained for 6771 Latin hypercube samples, which can be seen as an estimate of the true performance distribution of the design space, and the 1000 LH samples collected to initialize the algorithm. Such comparison shows the benefits brought by the performance-driven algorithm compared to a standard sampling scheme in terms of sample quality. The proposed method leads to a much denser sampling of designs with higher performance as the shift in density toward higher-performing scores (low mass) for the performance-driven samples demonstrate.

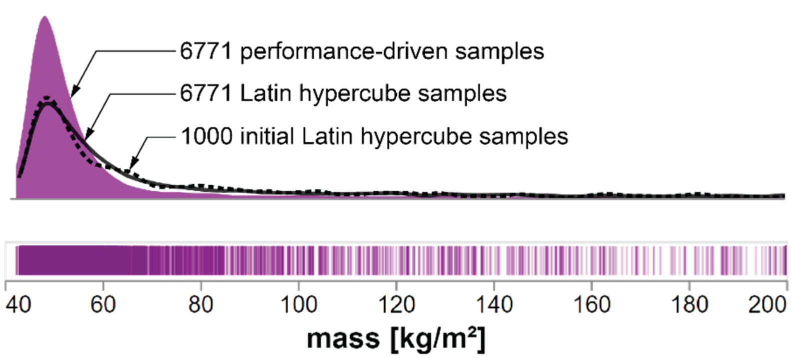

Figure 6: Kernel density estimations of 6771 performance-driven samples scores vs. 6771 and 1000 LH samples scores with strip plot of the scores of the performance-driven samples

To assess the effectiveness of the algorithm on this high-dimensional example, we can also look at the evolution of the sample quality. Figure 7 shows two graphs that shine a light on the algorithm's progress and confirm its value for sampling high-dimensional design spaces. The first (on the left) shows how the performance density of the samples collected at each step gradually moves to the left and peaks higher for lower i.e. better mass values. Particularly noteworthy is the big jump between step 0 and step 1: this shows that even a surrogate model relying on little data can significantly help sample much better design candidates. The second graph (right) confirms this trend: the mean sample score progressively decreases and the samples (each represented by a single dot) cluster downward.

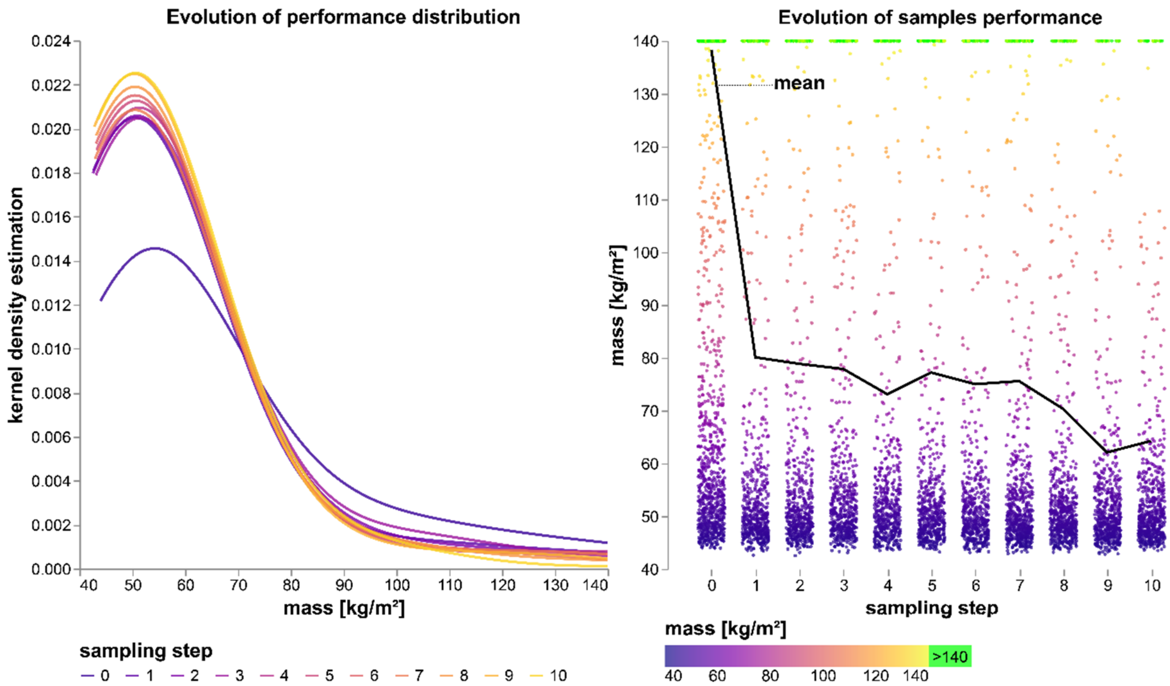

Figure 7: Performance of samples collected for each sampling step. Each point represents a single sample, its vertical position indicates the sample performance, and its horizontal position indicates when it was collected (jitter is introduced to improve readability). The mean performance for each step is shown by the solid line. The performance axis is bounded to [40,140], and samples with larger performance values are crowded at the top of the graph.

The results discussed and shown above were produced for a specific set of growth rate and β parameters, and, as was already discussed in 2.1, these two parameters can alter the result of the sampling process in ways that are worth highlighting. By running the performance-driven algorithm on the long-span roof example for 9 combinations of these two parameters (and using the values listed at the beginning of this section for the other hyperparameters) and tracking the mean of the samples collected at each step, we show that this intuition is confirmed experimentally (Figure 8). Choosing an initial high-value for the growth rate means that the in-step sample mean decreases drastically after the first step with only marginal improvements in subsequent steps. While this yields successive sampling steps with lower sample means, this may indicate that the algorithm is drilling down on a given region and ignoring other promising ones, such that one should not assume that a sudden drop followed by a plateau is necessarily best. A low growth rate and a low β also make the algorithm stall because it is simply not strict enough when filtering proposed LH samples. In summary, too high a growth rate does not yield enough exploration and low values for both parameters yield less exploitation. As Figure 8 shows, a choice of a low initial growth rate (1-5) and a moderate value of β (0.1-0.5) strikes a good balance between exploration and exploitation, but regardless of g and β, the algorithm yields samples with much better performance than the initial LH step.

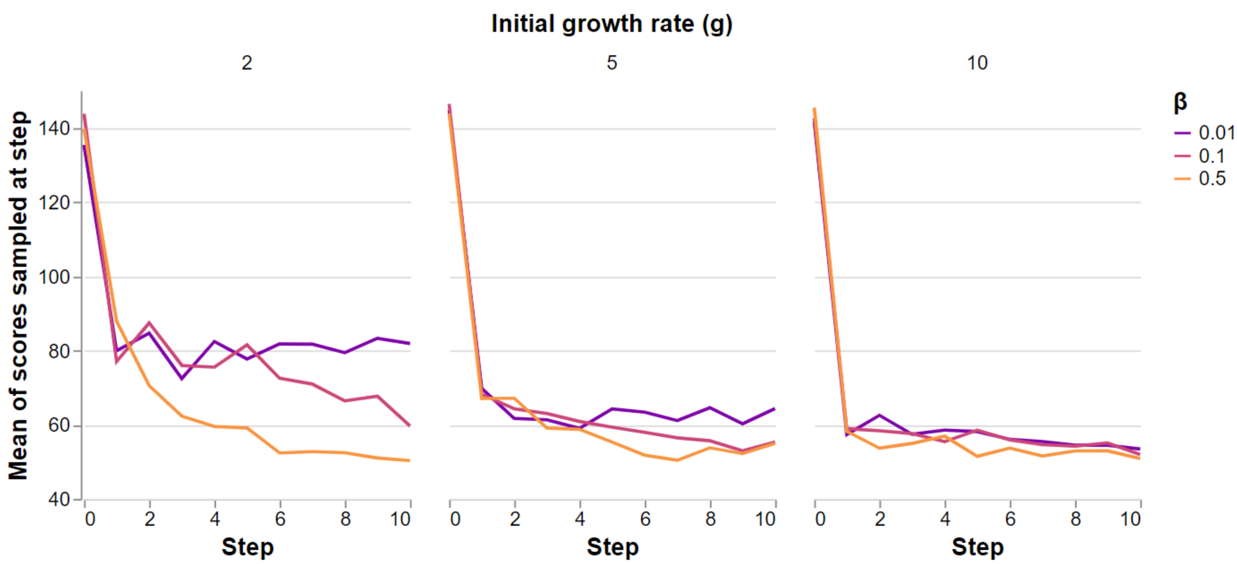

Figure 8: Influence of growth rate and β parameters on the mean performance of samples collected at each step.

3.3 Performance-conditioned VAE

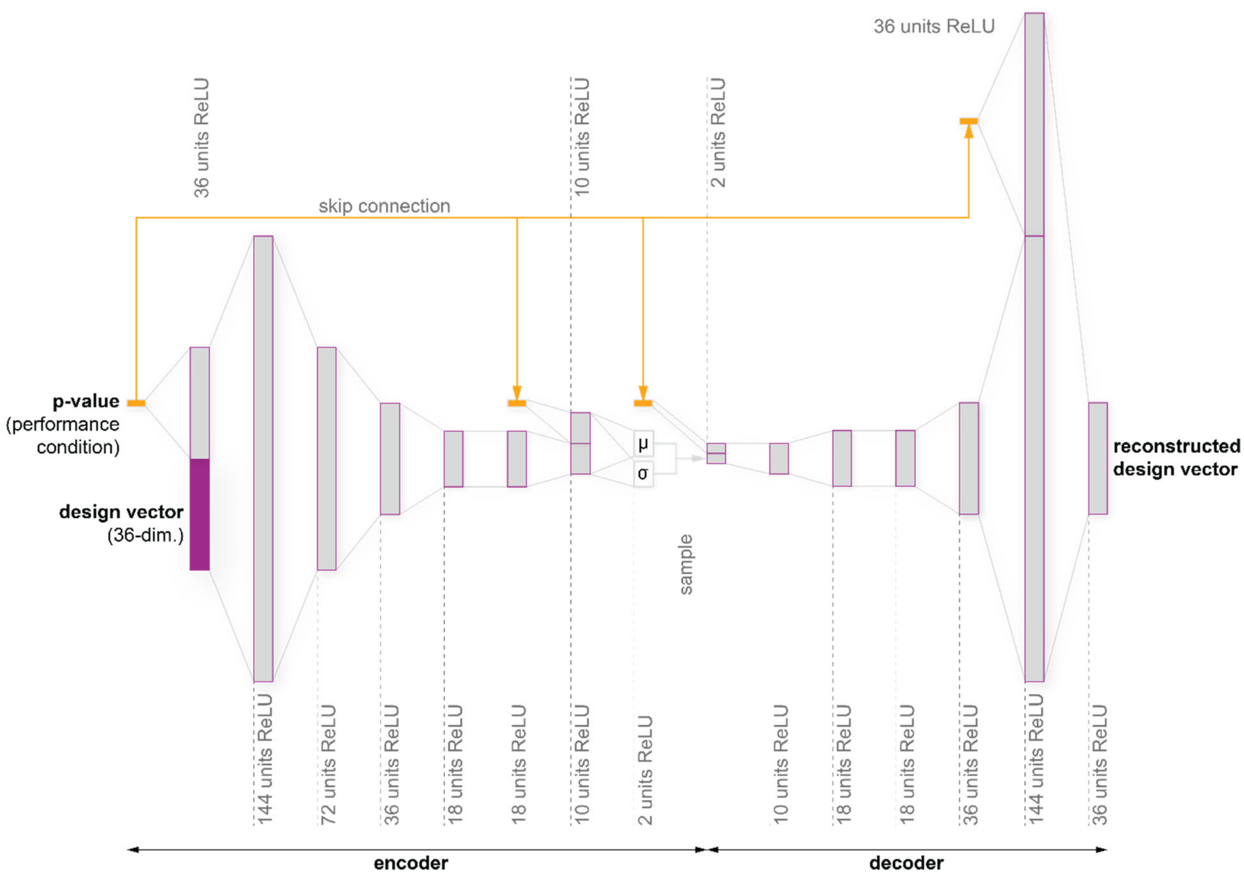

Figure 9: Architecture of performance-conditioned VAE used for the long span roof example.

The performance-conditioned variational autoencoder with 39,516 trainable parameters diagrammed in Figure 9 is trained based on the data collected by the performance-driven algorithm detailed in the previous subsection. Before training, design variables used are normalized from $[-3,3]$ to $[0,1]$. It is worth noting that the final activation of the network is a rectified linear unit activation (ReLU) which, in contrast to a sigmoid activation, does not restrict the output of the decoder to the $[0,1]$ domain, technically allowing decoded samples to have out-of-bound design variables. In practice, this accelerates training, and the bounds of the design variables are softly integrated by the model through learning.

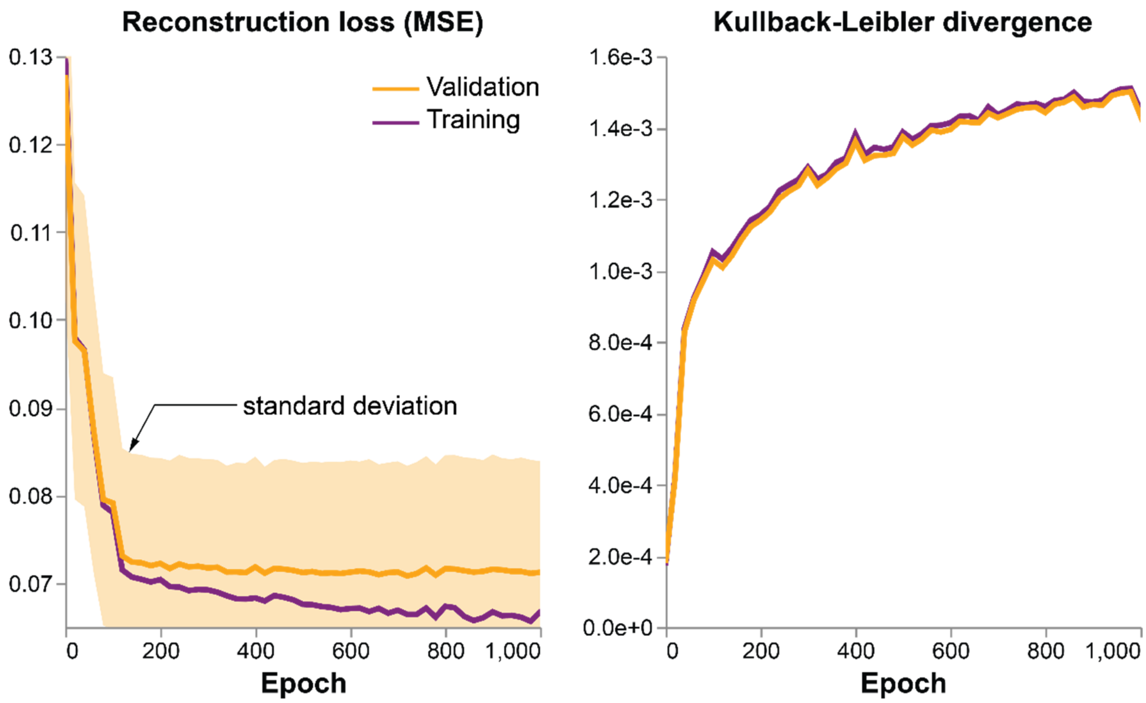

Figure 10: Evolution of reconstruction loss and Kullback-Leibler divergence of the performance-conditioned VAE during training.

The PVAE is trained for 1000 epochs with 5755 samples using the RMSprop gradient descent algorithm [32] to minimize the VAE loss, itself obtained by averaging the MSE reconstruction loss and the Kullback-Leibler (KL) divergence between the encoded samples and the normal distribution. The PVAE is validated with 1016 samples. Figure 10 shows the evolution of the loss components (MSE and KL divergence) for the training and validation sets as training progresses: the MSE is expectedly minimized, but the KL divergence progressively increases to an asymptotic value. The latter result may seem a little counterintuitive given the PVAE is trained to minimize the average of both the MSE and the KL divergence. However, the KL divergence should be understood as a regularization term, which is added to the MSE to ensure that the latent space has a continuous and smooth structure. Without the KL divergence, it would be possible to reduce the reconstruction loss even more, but that would be at the expanse of the latent space continuity. Conversely, minimizing the KL divergence would negatively impact the reconstruction MSE. From that perspective, the asymptotically increasing behavior of the KL divergence is only a reflection of the trade-off between the two terms of the VAE loss.

3.4 Latent space encoding

To assess the impact of the KL divergence on the structure of the latent space, it is useful to look at the projections of the high-dimensional training data points onto the latent space. Figure 11 shows how the 5755 training samples are encoded onto the 2D latent space by the encoder, organized according to a slightly asymmetrical normal-like distribution. This structure is a direct consequence of the use of a VAE (with its KL divergence term in the loss function) over the use a regular autoencoder and is indicative of a latent space that is continuous and smooth, both important features for design exploration.

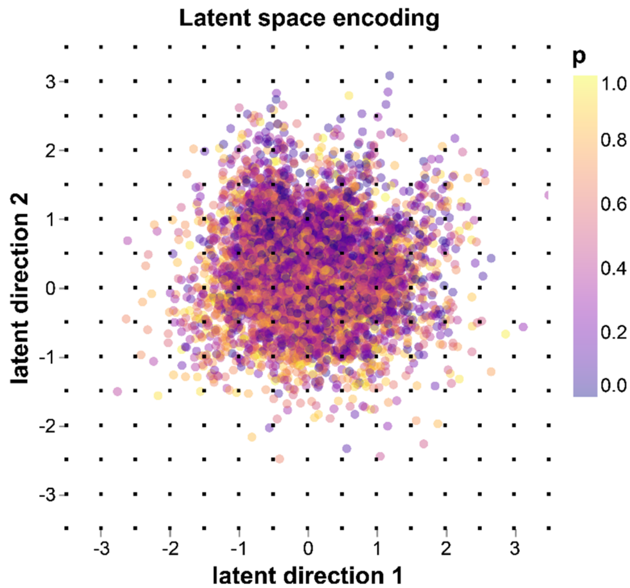

Figure 11: Projection of training samples onto the latent space by the encoder after training.

Figure 11 also shows that samples with different performance scores are superimposed in the latent space. This reflects the role that the performance condition plays: it essentially allows the PVAE encoder to use the same real estate in the latent space to pack multiple strata of the high-dimensional design space. Similarly, the performance condition allows the decoder to unfold the latent space into different manifolds in the high-dimensional design space.

3.5 Latent space decoding: an atlas of design subspaces

The PVAE does not only provide access to a single latent space but rather an atlas of performance maps that designers can sift through by adapting the performance condition. This is particularly useful because it allows designers to investigate potentially interesting trade-offs between performance and other intangible design factors. Figure 12 shows the performance maps obtained by decoding and evaluating the latent space for different p-values.

The maps show that the p-value (the performance condition) has the intended effect on the performance of the latent spaces: the performance scores increase—they get worse in this case—as the p-value is increased. The performance map for $p = 0$ includes designs with excellent performance scores, many of them hovering right above 40 kg/m^2, i.e. the best objective function value in the design dataset used for training the PVAE and obtained using performance-driven sampling. The latent spaces corresponding to larger p-values unsurprisingly contain only slightly worse designs that demonstrate the potential of the proposed approach to explore design candidates that are slightly suboptimal but qualitatively better. Interestingly, the performance contours are not smooth everywhere, even though the decoder mapping is continuous, because the objective function, the structural mass required to support the imposed loads, is itself non-smooth with respect to the original design variables. This is particularly salient for $p = 1$ where the weight of some designs in the latent space skyrocket: these are designs where the space truss has areas with small structural depths, resulting in large internal axial forces and section sizes.

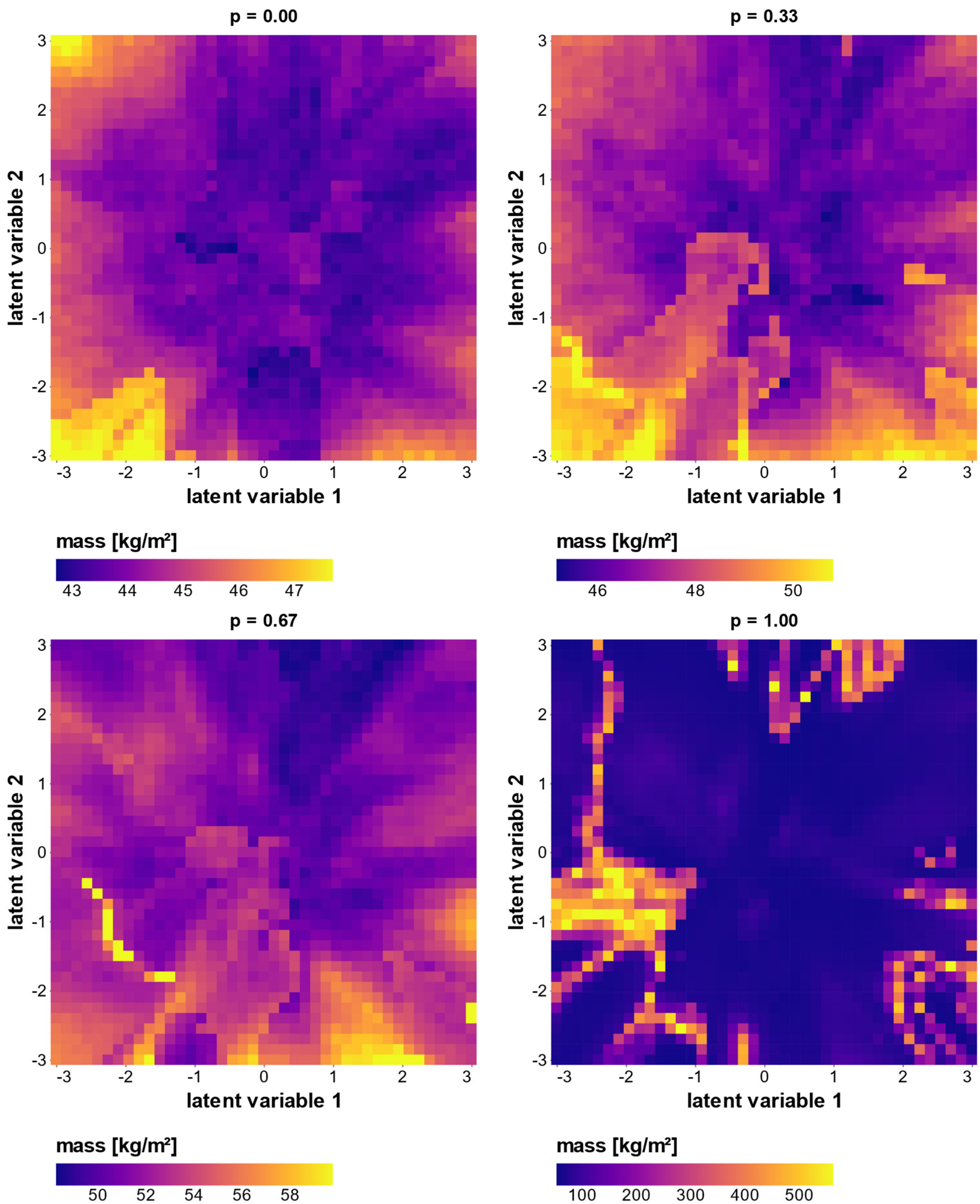

Figure 12: Objective function heat maps in the latent spaces learned by the performance-conditioned VAE for different p*-values.*

The consistent and anticipated link between the performance condition displayed in Figure 12 is even better illustrated by Figure 13, which shows how the performance of 100 random samples in the latent space changes as the p-value is increased. Figure 13 also highlights an interesting trend: for p-values smaller than 0, performance continues to improve (decrease) until around $p = -0.3$. Even though the PVAE is not trained with any samples associated with negative p-values, it learns to extrapolate beyond $p = 0$. These extrapolated trends broadly correspond to an increase of the space truss depth beyond the original bounds of the design space. As the depth of the space truss increases further for values lower than -0.3, the negative impact of the increased length of the structural members starts to outweigh the benefits of the larger structural depth, and the structural mass increases slightly.

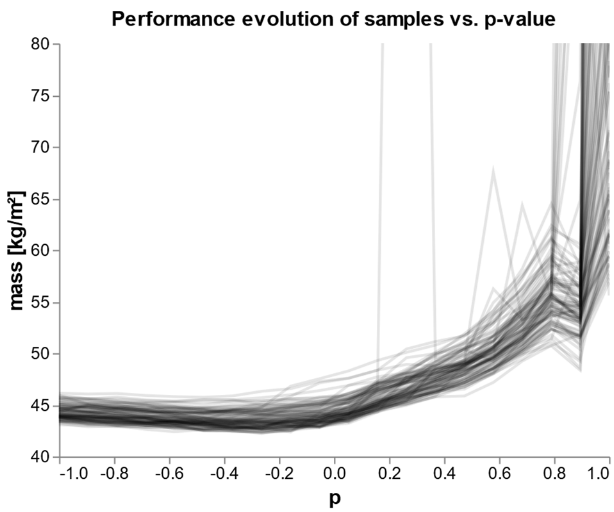

Figure 13: Evolution of the performance score of 100 random designs sampled from the latent space ([-3,3]×[-3,3]) (left) as the performance condition is increased. For values of p lower than 0, the VAE successfully extrapolates trends linked with an improvement (decrease) of the performance score beyond the original bounds of the design space to make the space truss deeper.

To understand the structure of the latent space learned by the PVAE, it is useful to look at how each point in the 2D latent space is mapped to each design variable of the original, high-dimensional design space. Figure 14 shows the values that each of the original design variables takes in different locations of the latent space for $p = 0$. It illustrates the non-linearity of the latent space is, which allows to pack more complex distributions of designs than linear encoding techniques like principal component analysis. It also demonstrates that the latent space is smooth and that any exploration path in the latent space continuously morphs or interpolate between designs.

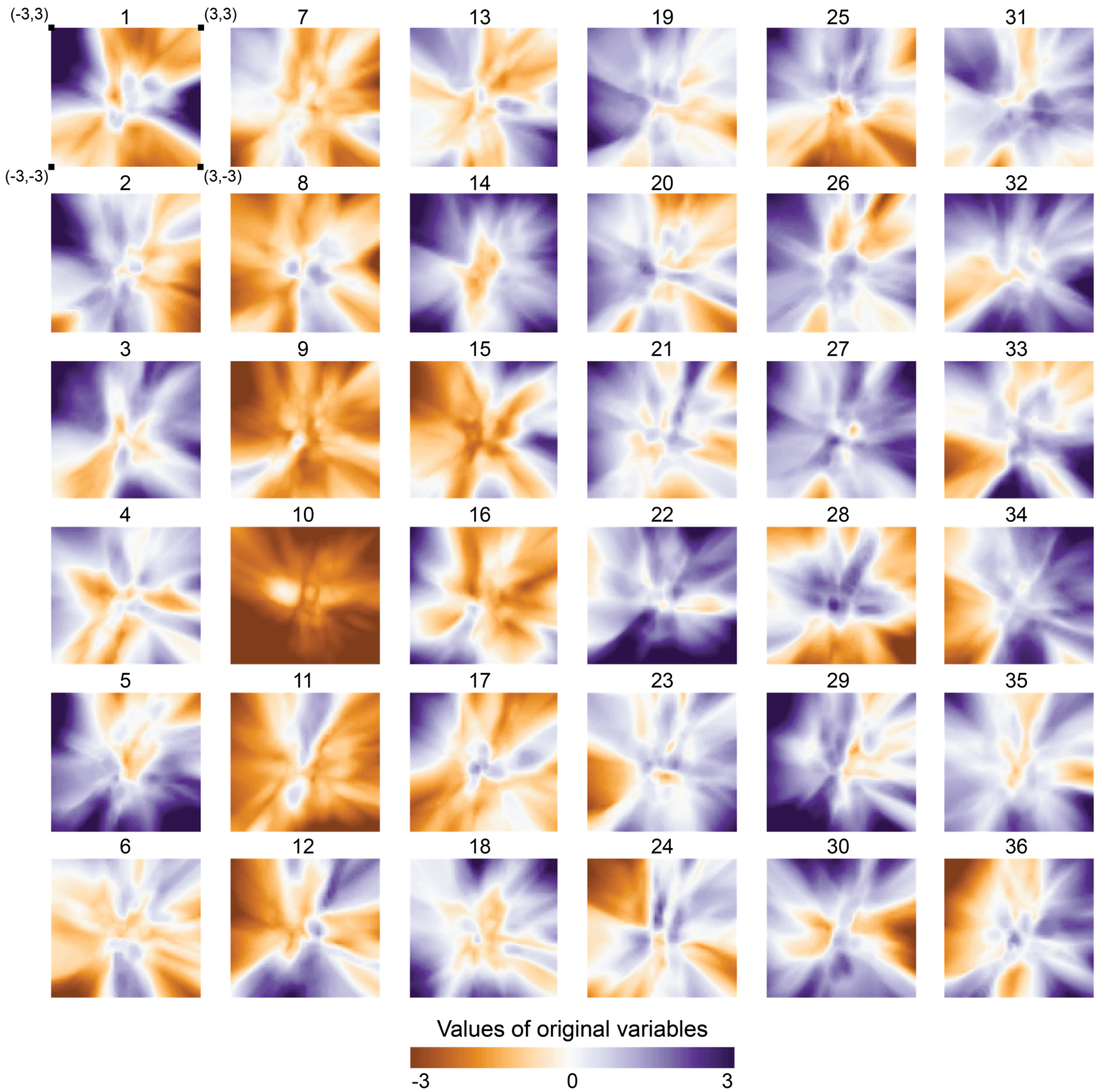

Figure 14: Mapping from latent space ($[-3,3]^2$) to original variables for $p = 0$. Numbers indicate the indices of each design variable as defined in Figure 5.

An interesting question is whether we can derive any meaning from the latent directions. Sometimes, dimensionality reduction techniques yield lower-dimensional representations with directions to which humans can ascribe meaning *a posteriori*. For example, research on lighting control has shown human-derived criteria for sensor lighting control can be extracted using PCA [33]. Interpreting the meaning of the directions of a learned latent space is typically easier for linear dimensionality reduction methods like PCA. Because the PVAE encoding and decoding are nonlinear, interpreting each latent direction globally is not as straightforward. Nevertheless, despite its apparent complexity, the latent space is intuitive to explore because it can be rendered at once on a 2D computer screen.

Of course, the latent space maps also change as the performance condition is modified: Figure 15 shows the evolution of the design variable maps for different p-values. The evolution of the latent space is smooth and shows that any individual design in the latent space is continuously morphed to match a prescribed performance condition. Figure 15 also highlights how the PVAE extrapolates trends for p-values under 0 or above 1.

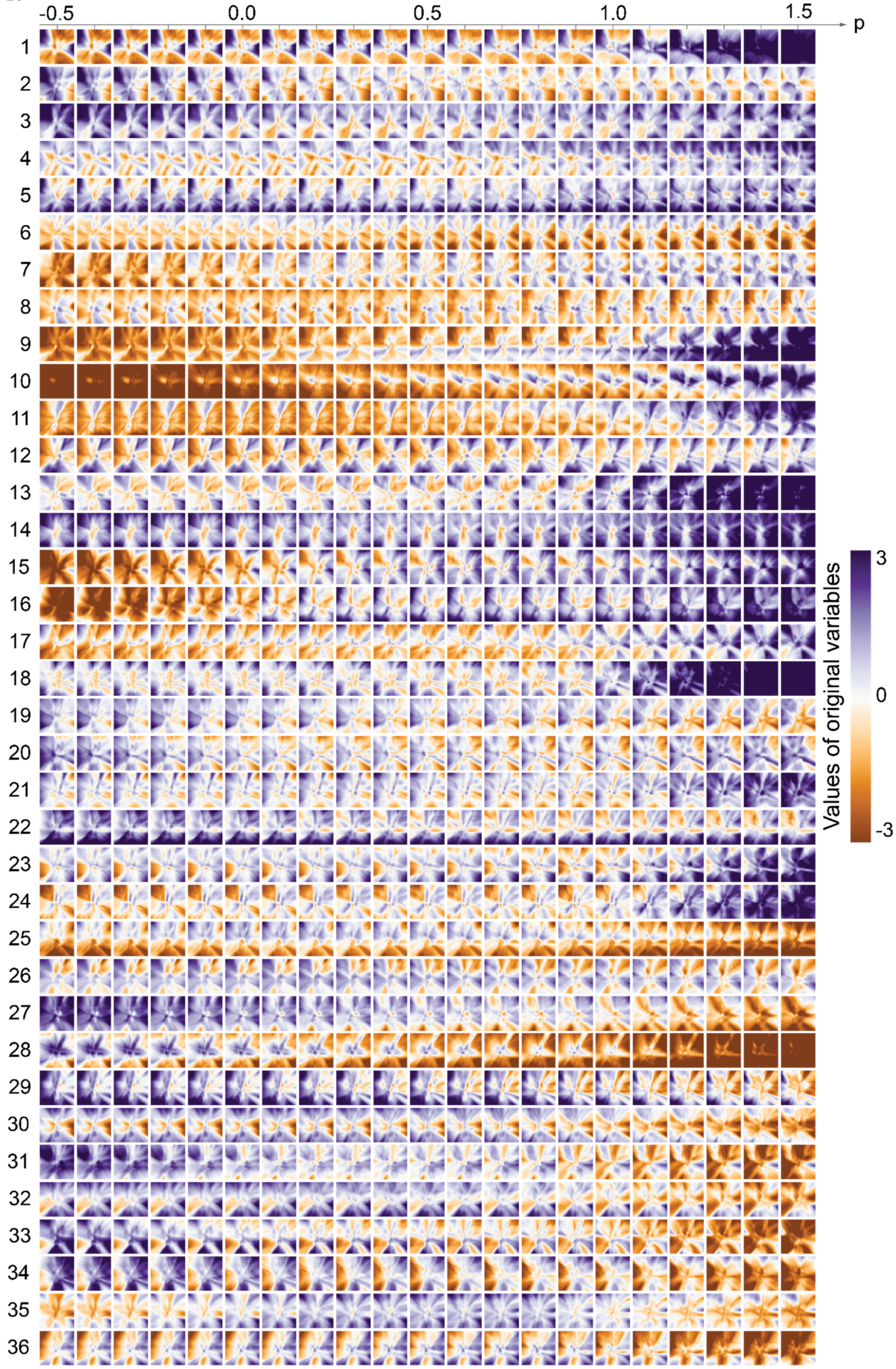

Figure 15: Evolution of latent space maps of original variables for increasing (left to right) values of the p-value.

In addition to the performance and variable maps, it is also important to look at the actual design geometries corresponding to different points of the latent space to understand the design subspace learned by the PVAE. Figure 16 shows 36 designs decoded from a regular grid of 6-by-6 samples in the latent space (for $p = 0$) and their respective performance scores. These designs demonstrate that the latent space contains geometrically and visually diverse. All of these designs perform very well, and the low-dimensional design subspace can be exhaustively searched by designers to find high-performance options with different qualitative properties.

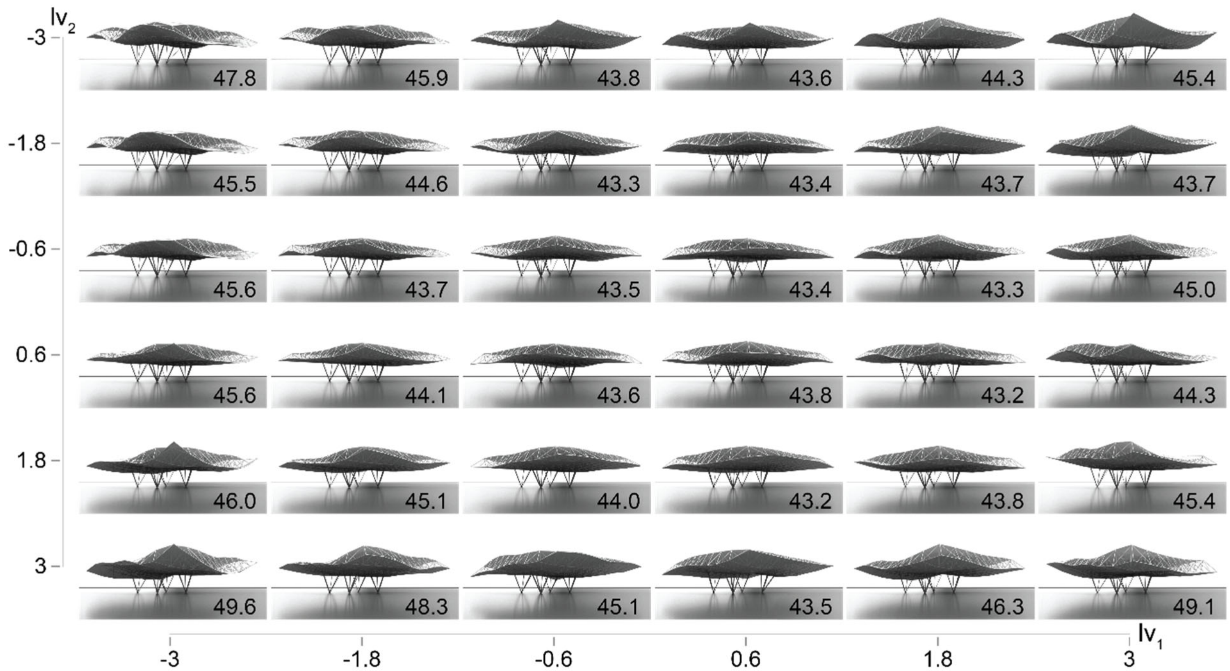

Figure 16: Renderings of a grid of designs in the latent space for $p = 0$ and their corresponding structural mass [kg/m^2]. The vertical and horizontal axes indicate the position of each design in the latent space.

While Figure 16 offers a snapshot of a grid of designs contained in the latent space for a specific value of the performance condition, the influence of the p-value on design geometry and performance is even better highlighted by looking at individual locations in the latent space with varying performance conditions. Figure 17 shows the morphological evolution of 4 designs at 4 different locations of the latent space as the performance condition is increased, and it illustrates the impact of the performance condition on both design geometry and performance: generally, designs in the latent space with $p = 1$ are essentially shallower versions of the ones in the latent space with $p = 0$. It is noteworthy that much of the latent space with $p = 1$ still contains many well-performing designs (see Figure 12) with much shallower morphologies. This demonstrates once again the potential of the proposed approach to explore design options that are sub-optimal quantitatively but potentially better fits for a host of other reasons.

These results confirm that the performance condition preserves most of the broad design characteristics of each design and mostly participates in making the space truss shallower. For experienced structural designers, this result is not surprising: structural depth in areas with high-bending moments, in this case over the supports, is almost always conducive to greater structural efficiency. However, the fact that the PVAE inferred it from data is non-trivial, and the PVAE does not simply operate by moving up the lower surface and moving down the upper surface. For example, some areas of the space truss become shallower more quickly than others.

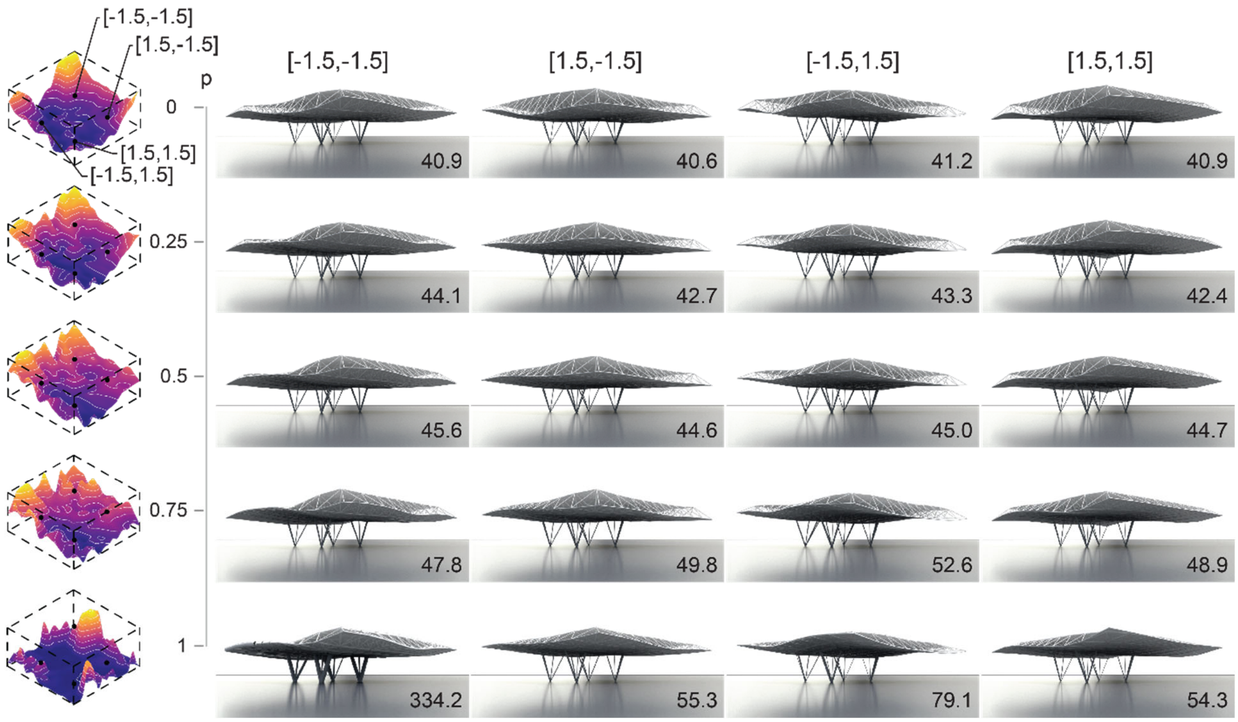

Figure 17: Morphological and performance evolution of 4 designs (right) in the latent space (left) as p increases from 0 to 1. Each column corresponds to a single point in the latent space (the location of which is shown at the top). As the performance condition increases, so does the structural mass, given in kg/m^2.

In addition, Figure 17 further shows the usefulness of the performance condition for exploring non-optimal designs. For example, the second and fourth columns of designs show that it can be used to dramatically change the geometry of near-optimal designs while remaining in acceptable territories from a performance perspective.

3.6 Latent space navigation

As the last section shows, it is possible to build visualizations that summarize the latent space and show both the geometric diversity and the performance of the designs it contains. Such visualizations offer great immediate snapshots of the high-performance regions compressed by the PVAE. However, the PVAE needs not be constrained to the production of static visualizations but can also be explored interactively. This is particularly easy given the 2-dimensionality of the latent space, and designers can navigate the latent space using a simple interface with a three-dimensional view showing the current design geometry, an interactive performance map that designers can move through using with the mouse cursor, and a knob, slider, or even text field to adjust the performance condition. Figure 18 shows a prototype of such an interface with the latent space represented by a 2.5D surface and an additional parallel coordinate plot indicating the decoded original design vector.

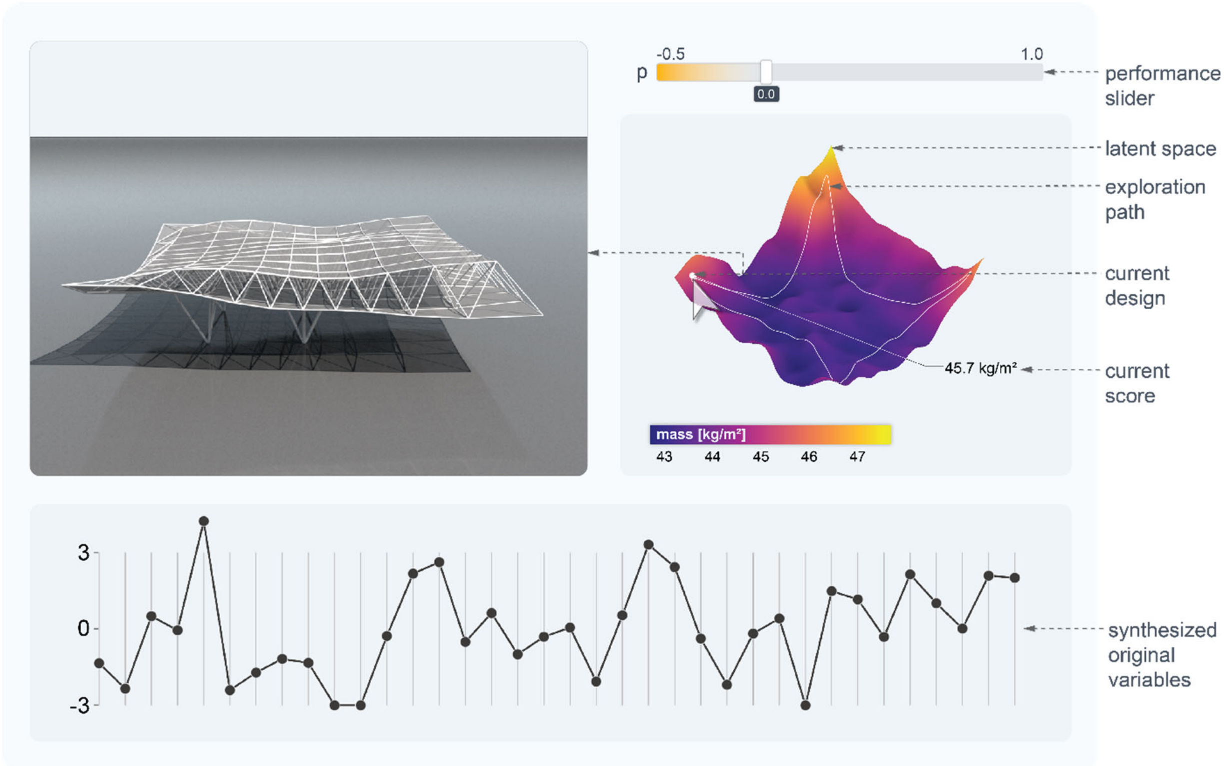

Figure 18: Prototype interface for latent space exploration.

Figure 19 shows a potential exploration path that a designer may take through the latent space in such an interface and the designs that would be explored along such a path. The visualization highlights how navigating through the latent space yields diverse high-performing designs, and the parallel coordinate plot in particular further shows the smoothness and high nonlinearity of the latent space. In addition, it shows the influence of the tunable performance condition. Compared to a manual exploration of the original design space, which would require sequentially adjusting 36 sliders or knobs, this mode of exploration allows designers to generate design variations by controlling only 3 variables, the first of which, the performance condition, allows them to tighten or relax their performance requirements.

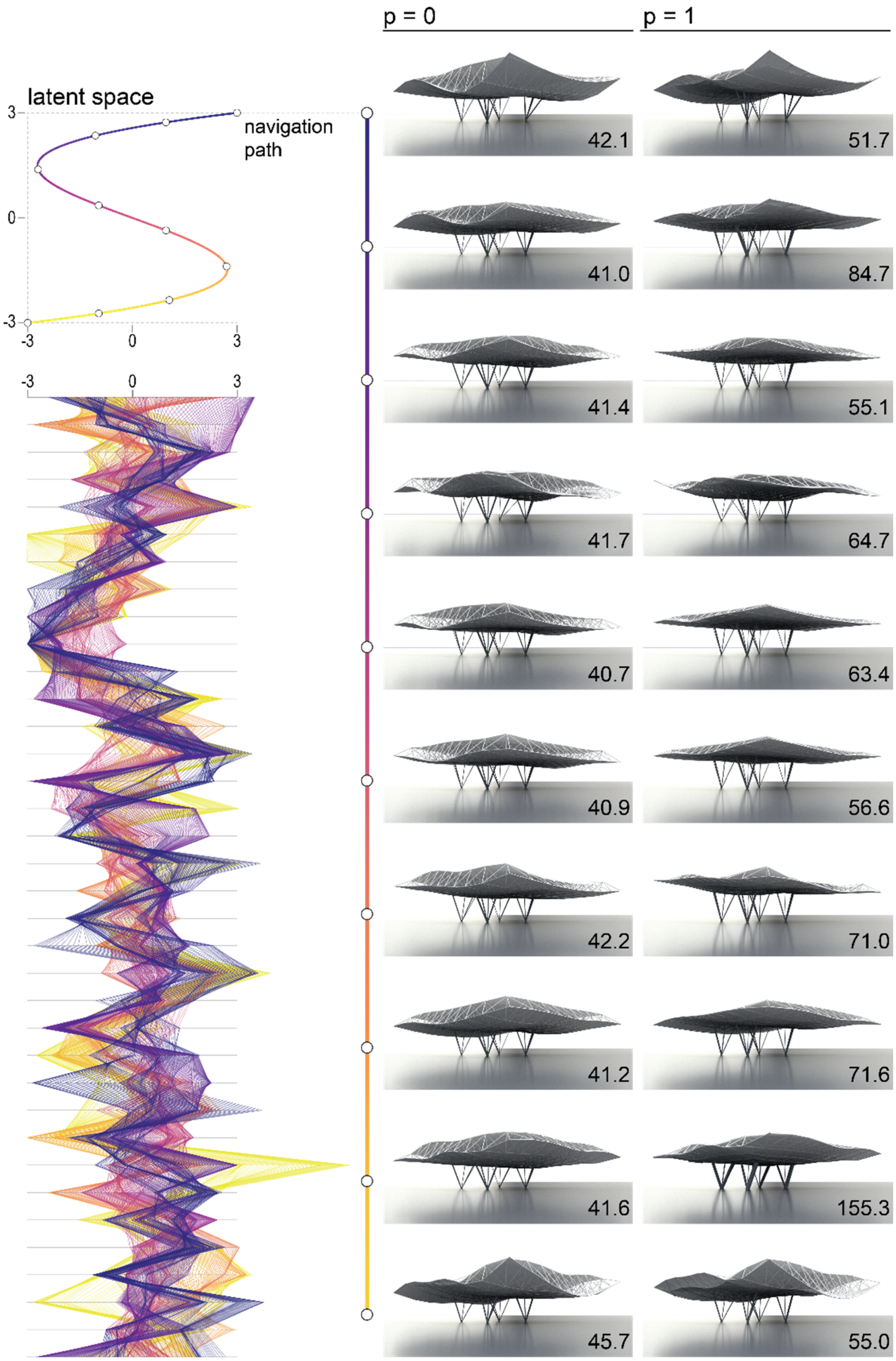

Figure 19: Exploration path in the synthetic latent design space. This visualization highlights a specific exploration path on the performance landscape of the latent design space. Renderings on the right display designs (accompanied by their performance scores in kg/m²) on the exploration path for $p = 0$ *and* $p = 1$*. The parallel coordinate plot on the left shows how the original design variables change as one navigates along the S-shaped exploration path for* $p = 0$*, with colors indicating the corresponding location of each of the designs on the path.*

4 Conclusion

This research contributes a method to build and train deep, performance-conditioned latent variable models that pack complex design spaces into continuous, smooth, low-dimensional design subspaces. Design subspace learning offers a new paradigm for performance-informed design exploration that is neither optimization nor random or undirected and that provides a navigable cartography of otherwise unwieldy design spaces.

4.1 Future Work

There are many interesting and potentially impactful directions for future work related to this research. First, it would be compelling to adapt design subspace learning to rule-based design spaces which have a variable number of parameters and are notoriously hard to control. Second, it would be powerful to illustrate the use of the contribution in multi-objective design contexts, where structural considerations might conflict with other performance objectives typical to architecture, such as daylight autonomy or energy use intensity. In some ways, this can be straightforward: the research presented here can be directly on custom composite objective functions that combine and weigh multiple and potentially divergent performance metrics in a single score, though there are probably more nuanced ways to marry the proposed method with existing multi-objective design approaches. Finally, this research discusses ways low-dimensional design subspaces may be integrated simply as explorable and interactive maps in design tools. Previous work has shown that the nature of the tools we use influences the way we design and that better tools, with more integrated and interactive interfaces, yield better design outcomes [34], [35]. Future work will be devoted to further studying how the interfaces proposed here impact design outcomes in user studies compared to undirected design exploration.

4.2 Concluding remarks

Design subspace learning offers a new paradigm for performance-informed design exploration that is neither optimization nor undirected search and that provides a navigable cartography of otherwise unwieldy design spaces. Design subspace learning provides ways to explore more design solutions that perform well and explicitly gives designers control over how much performance matters and allows them to negotiate the tension between functional requirements and intangible human factors. It can be used to power intelligent interfaces that foster the exploration and discovery of high-performing designs. Because these interfaces can act as a more natural and intuitive layer of understanding between human designers and complex design spaces, design subspace learning overcomes several important and fundamental limitations to many existing computational design methods and has the potential to broaden the adoption of performance-informed design processes.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1854833.

References

[1] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, Jan. 1982, doi: 10.1007/BF00337288.

[2] T. Kohonen, *Self-Organizing Maps*. Berlin, Germany: Springer-Verlag Berlin Heidelberg, 2001, ISBN: 978-3-642-56927-2.

[3] J. Harding, "Dimensionality Reduction for Parametric Design Exploration," in *Advances in Architectural Geometry*, 2016, pp. 274–286, doi: 10.3218/3778-4.

[4] E. Fuchkina, S. Schneider, S. Bertel, and I. Osintseva, "Design Space Exploration Framework: A modular

approach to flexibly explore large sets of design variants of parametric models within a single environment," in *Proceedings of the 36th eCAADe Conference*, A. Kepczynska-Walczak and S. Bialkowski, Eds. Lodz, Poland, 2018, pp. 367–376.

[5] L. Fuhrimann, V. Moosavi, P. O. Ohlbrock, and P. Dacunto, "Data-Driven Design: Exploring new Structural Forms using Machine Learning and Graphic Statics," Sep. 2018, Accessed: Jun. 15, 2020. [Online]. Available: http://arxiv.org/abs/1809.08660.

[6] N. C. Brown and C. T. Mueller, "Design variable analysis and generation for performance-based parametric modeling in architecture," *International Journal of Architectural Computing*, vol. 17, no. 1, pp. 36–52, Mar. 2019, doi: 10.1177/1478077118799491.

[7] Z. X. Conti and S. Kaijima, "Enabling Inference in Performance-Driven Design Exploration," in *Humanizing Digital Reality*, 2017, pp. 177–188, doi: 10.1007/978-981-10-6611-5.

[8] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive Growing of GANs for Improved Quality, Stability, and Variation," *arXiv preprint arXiv:1710.10196*, Oct. 2017.

[9] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations (ICLR), Conference Track Proceedings*, 2014, Accessed: Feb. 04, 2020. [Online]. Available: http://arxiv.org/abs/1312.6114.

[10] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proceeding of the 31st International Conference on Machine Learning*, 2014, pp. 1278–1286, Accessed: Jan. 24, 2020. [Online]. Available: http://proceedings.mlr.press/v32/rezende14.pdf.

[11] K. Sohn, H. Lee, and X. Yan, "Learning Structured Output Representation using Deep Conditional Generative Models," in *NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Montreal, Canada: MIT Press, 2015, pp. 3483–3491.

[12] N. Umetani and Nobuyuki, "Exploring generative 3D shapes using autoencoder networks," in *SA '17: SIGGRAPH Asia 2017 Technical Briefs*, 2017, pp. 1–4, doi: 10.1145/3145749.3145758.

[13] A. Burnap, Y. Liu, Y. Pan, H. Lee, R. Gonzalez, and P. Y. Papalambros, "Estimating and Exploring the Product Form Design Space Using Deep Generative Models," in *Proceedings of the ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2016, pp. 1–13, doi: 10.1115/DETC2016-60091.

[14] S. Carter and M. Nielsen, "Using Artificial Intelligence to Augment Human Intelligence," *Distill*, Dec. 2017, doi: 10.23915/distill.00009.

[15] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum, "Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling," in *30th Conference on Neural Information Processing Systems (NIPS 2016)*, 2016, pp. 82–90.

[16] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning Representations and Generative Models for 3D Point Clouds," in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 40–49.

[17] J. Wu, Y. Wang, T. Xue, X. Sun, W. T. Freeman, and J. B. Tenenbaum, "MarrNet: 3D shape reconstruction via 2.5D sketches," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 540–550.

[18] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Nov. 2017, pp. 1125–1134, doi: 10.1109/CVPR.2017.632.

[19] S. Chaillou, "ArchiGAN: a Generative Stack for Apartment Building Design," Harvard Graduate School of Design, 2019.

[20] D. Ritchie, K. Wang, and Y. Lin, "Fast and Flexible Indoor Scene Synthesis via Deep Convolutional Generative Models," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nov. 2019, pp. 6175–6183, doi: 10.1109/CVPR.2019.00634.

[21] K. Wang, M. Savva, A. X. Chang, and D. Ritchie, "Deep convolutional priors for indoor scene synthesis," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 1–14, Jul. 2018, doi: 10.1145/3197517.3201362.

[22] M. E. Yumer, P. Asente, R. Mech, and L. B. Kara, "Procedural Modeling Using Autoencoder Networks," *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*, pp. 109–118, 2015, doi: 10.1145/2807442.2807448.

[23] T. Wortmann and T. Schroepfer, "From Optimization to Performance-Informed Design," in *SIMAUD '19: Proceedings of the Symposium on Simulation for Architecture and Urban Design*, 2019, pp. 1–8.

[24] G. E. P. Box and K. B. Wilson, "On the Experimental Attainment of Optimum Conditions," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 13. Wiley, pp. 1–45, 1951, doi: 10.2307/2983966.

[25] S. A. Renganathan, R. M. and, and J. Ahuja, "Enhanced data efficiency using deep neural networks and Gaussian processes for aerodynamic design optimization," *arXiv*, Aug. 2020, Accessed: Jan. 31, 2021. [Online]. Available: http://arxiv.org/abs/2008.06731.

[26] H. Ma, X. Hu, Y. Zhang, N. Thuerey, and O. J. Haidn, "A Combined Data-driven and Physics-driven Method for Steady Heat Conduction Prediction using Deep Convolutional Neural Networks," *arXiv*, May 2020, Accessed: Jan. 31, 2021. [Online]. Available: http://arxiv.org/abs/2005.08119.

[27] M. A. Mohamad and T. P. Sapsis, "Sequential sampling strategy for extreme event statistics in nonlinear dynamical systems," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 115, no. 44, pp. 11138–11143, Oct. 2018, doi: 10.1073/pnas.1813263115.

[28] T. P. Sapsis, "Output-weighted optimal sampling for Bayesian regression and rare event statistics using few samples," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 476, no. 2234, p. 20190834, Feb. 2020, doi: 10.1098/rspa.2019.0834.

[29] P. Pandita, I. Bilionis, and J. Panchal, "Bayesian Optimal Design of Experiments For Inferring The Statistical Expectation Of A Black-Box Function," *Journal of Mechanical Design, Transactions of the ASME*, vol. 141, no. 10, Jul. 2018, doi: 10.1115/1.404393.

[30] R. Agrawal, C. Squires, K. Yang, K. Shanmugam, and C. Uhler, "ABCD-Strategy: Budgeted Experimental Design for Targeted Causal Structure Discovery," Feb. 2019, Accessed: Mar. 22, 2020. [Online]. Available: http://arxiv.org/abs/1902.10347.

[31] C. Preisinger and M. Heimrath, "Karamba—A Toolkit for Parametric Structural Design," *Structural Engineering International*, vol. 24, pp. 217–221, 2014, doi: 10.2749/101686614X13830790993483.

[32] G. Hinton, "Neural Networks for Machine Learning: Lecture 6," 2012. http://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf (accessed Aug. 07, 2020).

[33] N. Zhao, M. Aldrich, C. Reinhart, and J. Paradiso, "A Multidimensional Continuous Contextual Lighting Control System Using Google Glass," in *BuildSys '15: Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, 2015, pp. 235–244, doi: 10.1145/2821650.2821673.

[34] G. Tsai and M. Yang, "How It Is Made Matters: Distinguishing Traits of Designs Created by Sketches, Prototypes, and CAD," in *Proceedings of the ASME 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2017, vol. 7, doi:

10.1115/DETC2017-68403.

[35] E. Burnell, M. Stern, A. Flooks, and M. C. Yang, “Integrating design and optimization tools: A designer centered study,” in *Proceedings of the ASME Design Engineering Technical Conference*, 2017, vol. 7, doi: 10.1115/DETC2017-68307.