Augmented Intelligence for Architectural Design with Conditional Autoencoders: Semiramis Case Study

Luis Salamanca[1](✉), Aleksandra Anna Apolinarska[2], Fernando Pérez-Cruz[1], and Matthias Kohler[2]

[1] Swiss Data Science Center, ETH Zürich, Zürich, Switzerland
luis.salamanca@sdsc.ethz.ch
[2] Gramazio Kohler Research, ETH Zürich, Zürich, Switzerland

Abstract. We present a design approach that uses machine learning to enhance architect's design experience. Nowadays, architects and engineers use software for parametric design to generate, simulate, and evaluate multiple design instances. In this paper, we propose a conditional autoencoder that reverses the parametric modelling process and instead allows architects to define the desired properties in their designs and obtain multiple predictions of designs that fulfil them. The results found by the encoder can oftentimes go beyond what the user expected and thus augment human's understanding of the design task and stimulate design exploration. Our tool also allows the architect to under-define the desired properties to give additional flexibility to finding interesting solutions. We specifically illustrate this tool for architectural design of a multi-storey structure that has been built in 2022 in Zug, Switzerland.

1 Introduction

1.1 Motivation

Design tasks in architecture, engineering, and construction often entail many parameters, multiple constraints, and contradicting objectives. In a traditional design process, the architects rely on experience to craft a handful of candidate solutions or can use parametric modelling tools to easily create many variations of the design. However, to find designs that fulfil predefined performance requirements, the user needs to find the right combination of the parameters, which can be difficult and time-consuming.

We aim to invert the design paradigm by using machine learning, so that the user can specify the required attributes and in return be presented with different designs that fulfil them (Fig. 1, bottom). The purpose of our approach is not optimization but design exploration. Since the mapping from the geometric design parameters to the desired performance attributes is not injective, there may exist many designs with similar performance. This way, we expect the machine learning tool to augment the early design process, and enhance the architect's experience through a fast and intuitive exploration of the solution space.

L. Salamanca and A.A. Apolinarska—Contributed equally.

C. Gengnagel et al. (Eds.): DMS 2022, *Towards Radical Regeneration*, pp. 108–121, 2023.
https://doi.org/10.1007/978-3-031-13249-0_10

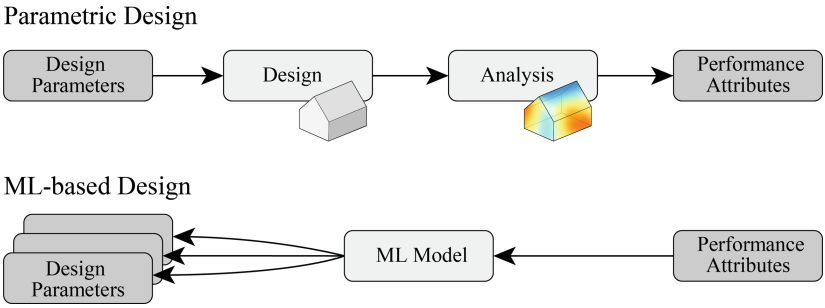

Fig. 1. In the parametric design process (top), the designer first chooses input parameters, creates a design instance, and simulates it (e.g. structure, solar gains) to evaluate its performance. Our proposed paradigm inverts this process (bottom).

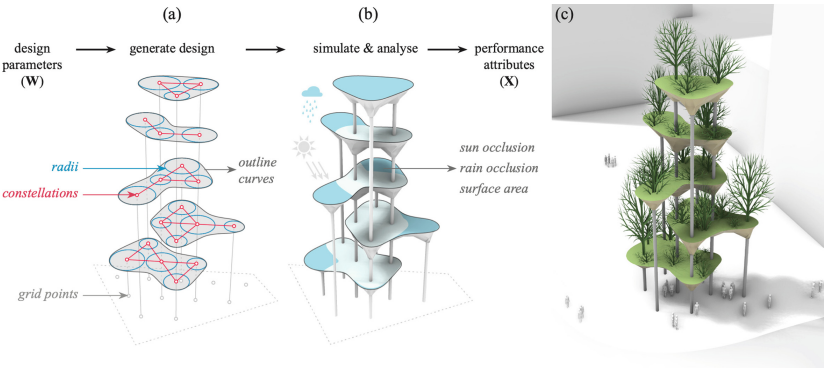

Fig. 2. The Semiramis project: a) setout and geometry constructed from design parameters **W**, b) analysis to obtain performance attributes **X**, c) the final design visualized in context and scale.

1.2 Case Study: Semiramis

Although the presented approach is generalizable to any design, to demonstrate this idea we apply it in a real architectural project. Our case study is a multi-storey structure of stacked plant platforms (an urban vertical garden project called Semiramis (Gramazio Kohler Research 2022), see Fig. 2c), that has been installed in 2022 in Zug, Switzerland. The particular design problem we address is to determine the outline curves of these platforms so that the total planting area and exposure to sun and rainfall satisfy requirements defined by the client and the landscape architect.

The five bowl-shaped platforms are placed at fixed heights and supported by columns on a triangular grid of 11 points. To design the platforms, we parametrize their outline shapes using a signed distance function that draws a smooth blend around the support points of each platform given the specified radii (see Fig. 2a). Each platform is supported by 3 to 5 columns in one of the ten hand-picked *base shapes* depicted in Fig. 3. All the possible positions, rotations and reflections of these base shapes within the given grid result in 115 *constellations*, i.e. subsets of the grid points that define support points for a platform. In summary, the *design parameters* are, for each platform: the choice of

a constellation and the corresponding radii. The *performance attributes* are: the area of each platform, the combined sun occlusion and the combined rain occlusion for all platforms.

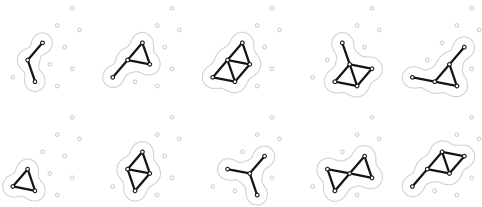

Fig. 3. Ten base constellation schemes. By including different positions and orientations of them within the 11-point grid, there are overall 115 possible constellations to choose from for each platform.

1.3 Proposed Approach

We set up a simple parametric *design+analysis* (DA) model that creates and analyses design instances from given input parameters. To generate designs that fulfill certain properties, we propose to use a *conditional autoencoder* (AE) (Sohn et al. 2015). This AE model is trained using a dataset consisting of tuples of design parameters and the associated performance attributes, obtained using the DA model. During training, in the latent space the performance measures are regressed together with the estimated latent variables to be able to propose multiple designs. Once the model is trained, the encoder emulates the parametric model, and the decoder maps from requested performance attributes to suitable designs, i.e. sets of design parameters. This tool enables the architect to explore different equally performing solutions to select their preferred design. When under-specifying the performance measures, the decoder can explore more freely the potential solutions.

2 Related Work

In architectural design and engineering, machine learning (ML) techniques are rapidly gaining interest in two main applications: for generative design and for design optimization. In the latter case, ML methods promise to solve many complex, black-box problems (Costa and Nannicini 2018) where gradient information is unavailable or impractical to obtain. Prominent examples employ evolutionary algorithms (Hornby et al. 2006), or convolutional neural networks (Takahashi et al. 2019), (Banga et al. 2018). In generative design, which we address in this paper, the role of ML methods is to help create many variations of designs. Models based on GAN or autoencoder architectures show considerable potential for this application, by encoding high-dimensional design variables in a low-dimensional design space. Notable work in this field includes generation of 2D floor plans (Hu et al. 2020; Chaillou 2020; Nauata et al. 2020), 3D shapes of family houses (Steinfeld et al. 2019) and of other building typologies (Miguel et al. 2019), or design

of automotive parts (Oh et al. 2019). Other works provide general methodologies that allow encoding varied objects (Talton et al., 2012; Mo et al. 2019), using a hierarchy of the constituent parts. A common challenge in these applications is the parametrization/representation of the design, which impacts the flexibility of the design process and the quality of the solution (Brown and Mueller 2019).

Against this background, our method allows tailoring the trainable variables to the specific design question, and generating new designs with the same quality as those in the dataset. Hence, our approach can be applied to different design problems by adapting the type and dimensions of input parameters and of performance attributes. And finally, the user receives not one but multiple solutions that approximate the desired performance.

3 Methods

3.1 Implemented Solution

We choose a conditional autoencoder (AE) (Sohn et al. 2015) neural network architecture because, by having as inputs the design parameters **W**, we can enforce some performance attributes **X** in the hidden layer, and also the reconstruction at the output (Fig. 4a). After training, the decoder allows to, given some desired performance attributes $\mathbf{X^d}$, generate new sets of design parameters $\mathrm{W^d}$ (Fig. 4b).

The design parameters $\mathbf{W}_j$ for each platform j are: the constellation defining the support points, and a radius for each grid point (see 1.2). The constellations are encoded as a C-dimensional one-hot vector $\mathbf{W}_j^{\mathrm{C}}$ ($\mathrm{C} = 115$). The radii are represented in an R-dimensional vector $\mathbf{W}_j^{\mathrm{R}}$ ($\mathrm{R} = 11$) with non-negative real values for supporting grid points and otherwise zero. Thus, 630 variables as input W for $\mathrm{P} = 5$ platforms:

$$\mathbf{W} = [\mathbf{W}_1^{\mathrm{C}}, \ldots, \mathbf{W}_{\mathrm{P}}^{\mathrm{C}}, \mathbf{W}_1^{\mathrm{R}}, \ldots, \mathbf{W}_{\mathrm{P}}^{\mathrm{R}}]$$

In the latent space we have the following. **X** is a $\mathrm{D} = 7$ dimensional vector comprising the performance attributes (the surface area of each platform, and the sun and rain occlusions). **Z**, a DL-dimensional latent vector, encodes the input information, and allows during the generation phase to explore different solutions given some performance attributes.

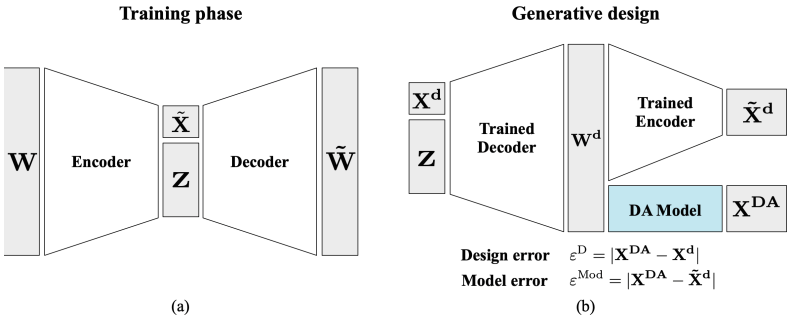

Fig. 4. In (a), a schematic of the architecture, with both **Z** and **X** in the latent space. In (b), the use of the trained encoder and decoder for the generation of new designs $\mathbf{W^d}$.

Further, $\tilde{\mathbf{W}}$ and $\tilde{\mathbf{X}}$ refer to the output during training of decoder and encoder respectively, that may differ from the fed training tuples $\mathbf{W}$ and $\mathbf{X}$. During generative design, $\mathbf{X^d}$ are the performance attributes values requested for the new designs $\mathbf{W^d}$.

3.2 Model Architecture and Training

The specific layers that comprise the neural network, dimensions used, and losses, are depicted in Fig. 5. For each one-hot vector $\mathbf{W}_j^{\mathrm{C}}$ we use different softmax blocks, and a sigmoid block for the output corresponding to the radii $\mathbf{W}^{\mathrm{R}}$. The losses on $\mathbf{Z}$ are regularization terms to enforce zero-mean and unit-variance. Each loss is weighted differently when computing the total loss.

Once the model is trained, all W in the training set are passed through the encoder. Given that $\tilde{\mathbf{X}}$ and $\mathbf{Z}$ in the hidden layer are approximately Gaussian, we model their joint distribution as a multivariate Gaussian with zero-mean and covariance matrix $\Sigma_{\tilde{\mathbf{X}}\mathbf{Z}}$, also beneficial for sampling purposes.

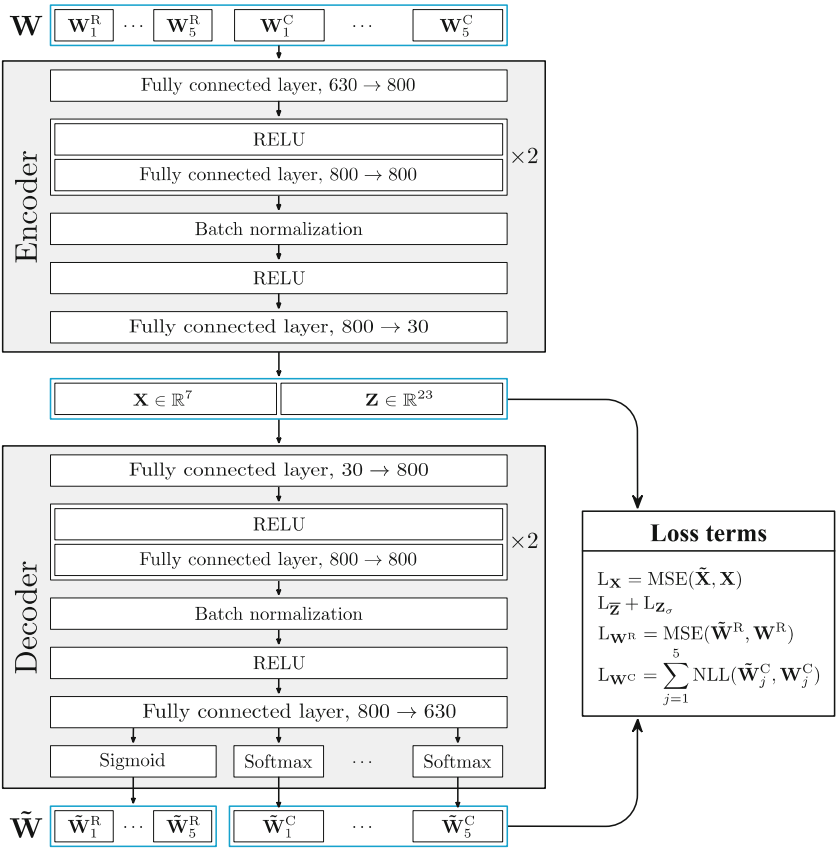

Fig. 5. Architecture of the AE model and losses.

3.3 Generative Design

To generate new designs, the designer specifies the desired performance attributes $\mathbf{X^d}$, or a subset $\mathbf{X^d_{sub}}$ (for a more flexible exploration of the solution space), and the number $\mathtt{Nd}$ of requested solutions. Now, the model can provide precise designs because:

1. It can sample $\mathbf{Z}$ (or the non-specified performance attributes and $\mathbf{Z}$), using the distribution of $\tilde{\mathbf{X}}$ and $\mathbf{Z}$ (Sect. 3.2). Sampling and evaluation can be run in parallel, hence are computationally light processes, enabling the quick generation of any number of designs (decoder in Fig. 4b).
2. The trained encoder acts as a surrogate model of the DA model and can be used to quickly estimate the performance attributes $\tilde{\mathbf{X}}^{\mathbf{d}}$ for all generated designs (encoder in Fig. 4b).

Hence, to further improve the performance of the generated designs, our strategy is to internally generate 100x more solutions and then use the encoder to choose the best $\mathtt{Nd}$ where $\tilde{\mathbf{X}}^{\mathbf{d}}$ is closest to $\mathbf{X^d}$ (or $\mathbf{X^d_{sub}}$). More details in Fig. 6.

Input: $\mathbf{X^d_{sub}}$, $\mathtt{N_d}$
Output: $\mathtt{N_d}$ sets $\mathbf{W^d}$ of generated design parameters
$\mathtt{N_{Tot}} = 100 \cdot \mathtt{N_d}$
$\varepsilon = [\,]$
for $n = 1$ **to** $\mathtt{N_{Tot}}$ **do**
  Different scenarios for the computation of $\mathbf{X^d}$ *and* $\mathbf{Z}$
  if No performance attributes requested **then**
    Sample $\mathbf{X^d}$ and $\mathbf{Z}$ from a Gaussian of mean-zero and covariance $\Sigma_{\tilde{\mathbf{X}}\mathbf{Z}}$
  else if All the performance attributes are indicated **then**
    $\mathbf{X^d} = \mathbf{X^d_{sub}}$, and sample $\mathbf{Z}$ conditioned on $\mathbf{X^d}$
  else
    if Total area is requested **then**
      Sample first from a Dirichlet to split the remaining area (total area minus other areas indicated)
    end if
    Sample all the remaining unspecified performance attributes to form vector $\mathbf{X^d}$,
    Sample $\mathbf{Z}$ conditioned on $\mathbf{X^d}$
  end if
  Computation of designs using the trained decoder
  $\tilde{\mathbf{W}}^{\mathbf{d}} = \mathbf{decoder}(\mathbf{X^d}, \mathbf{Z})$
  One-hot vectors from the logits of $\tilde{\mathbf{W}}^{\mathbf{d}}$:

$$\mathbf{W^d} = [\mathbf{argmax}\left(\tilde{\mathbf{W}}^{\mathrm{C}}_1\right), \ldots, \mathbf{argmax}\left(\tilde{\mathbf{W}}^{\mathrm{C}}_{\mathrm{P}}\right), \tilde{\mathbf{W}}^{\mathrm{R}}_1, \ldots, \tilde{\mathbf{W}}^{\mathrm{R}}_{\mathrm{P}}]$$

  Assessment of designs performance using the trained encoder
  $\tilde{\mathbf{X}}^{\mathbf{d}} = \mathbf{encoder}(\mathbf{W^d})$
  Computation of the performance attributes' errors:

$$\varepsilon_n = \sum_{i \in \mathbf{X^d_{sub}}} \frac{1}{\mathrm{std}(\mathbf{X}_i)} |\tilde{x}_i^{\mathbf{d}} - x_i^{\mathbf{d}}|$$

  Append ε_n to ε
end for
Return the $\mathtt{N_d}$ sets of parameters $\mathbf{W^d}$ with smallest values in ε

Fig. 6. Algorithm for prediction of new designs using the AE model

4 Evaluation

To evaluate the AE model we use the following sets of performance attributes (as observed in Fig. 4b): those requested by the user ($\mathbf{X^d}$), and the ones predicted by the encoder ($\tilde{\mathbf{X}}^{\mathbf{d}}$) and calculated through the DA model ($\mathbf{X}^{\mathrm{DA}}$) for some generated designs $\mathbf{W^d}$. For attribute i we then calculate the *design error* as $\varepsilon_i^{\mathrm{D}} = \left|\mathbf{X}_i^{\mathrm{DA}} - \mathbf{X}_i^{\mathrm{d}}\right|$, and the *model error* as $\varepsilon_i^{\mathrm{Mod}} = \left|\mathbf{X}_i^{\mathrm{DA}} - \tilde{\mathbf{X}}_i^{\mathrm{d}}\right|$.

4.1 Novelty of New Designs

To test the ability of the model to generate novel designs (never seen during training), we request 50 000 designs for a given $\mathbf{X^d}$, and select the best 100, i.e. with $\tilde{\mathbf{X}}^{\mathbf{d}}$ closest to $\mathbf{X^d}$. The values chosen for $\mathbf{X^d}$ are close to the mean of the distribution, which facilitates finding precise designs also in the training set. Then, we select the best 100 samples from the training set, i.e. closest to $\mathbf{X^d}$.

In Fig. 7 we depict the histograms of the average error distributions. For the designs generated through the AE model, we compute the design error (black) and the error $\left|\mathbf{X}_i^{\mathbf{d}} - \tilde{\mathbf{X}}_i^{\mathbf{d}}\right|$ (blue). Both are more shifted to the left, i.e. towards smaller errors, compared to the error for the designs drawn from the training set (red). This means the AE model delivers designs closer to the request than when selecting from the training set, which demonstrates the generation of novel, and more accurate, designs. This is exploited further thanks to the model ability to generate more designs and return the N best ones.

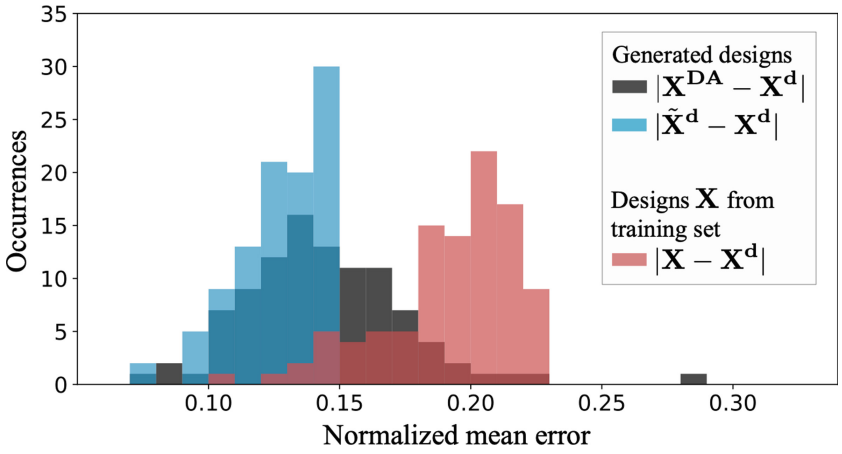

Fig. 7. Errors' distribution for generated designs and for samples from the training set.

4.2 Exploration of Design Space

By interrogating the AE model, the designer can understand the space of solutions: constraints, correlations, feasibility, etc. We illustrate this through seven different requests

$\mathbf{X}^{\mathbf{d}}$ of rain occlusion and total area (red dots in Fig. 8). For each, we randomly pick three designs from the best 1% out of 1 000, and plot $\mathbf{X}^{\mathrm{DA}}$ (black diamonds) and the corresponding design errors (black line).

The designs generated for requests B, C and E approximate accurately the intended values, as they fall in an area where the model has observed more training samples. Conversely, the large error for designs associated to G suggests that such request is unachievable: low rain occlusion (scattered platforms) and large total area (big platforms stacked vertically) are clearly contradictory requirements.

For requests A, D and F, in areas with few training samples (<5), the AE model was still able to discover new feasible and accurate solutions. In Fig. 9 we present the richness of proposed solutions by showing 10 exemplary designs for each request.

4.3 Assessment of Model Performance

The validation is performed by assessing how well generated designs match the requested performance attributes, for their full distribution. For each performance attribute, its distribution is split into bins, and for each we request 1 000 designs.

This allows calculating $\mathbf{X}_i^{\mathrm{DA}}$ and $\varepsilon_i^{\mathrm{D}}$ for each sample, leading to a mean design error per bin. Figure 10 shows these errors when requesting only one performance attribute (a, b) and two simultaneously (c, d).

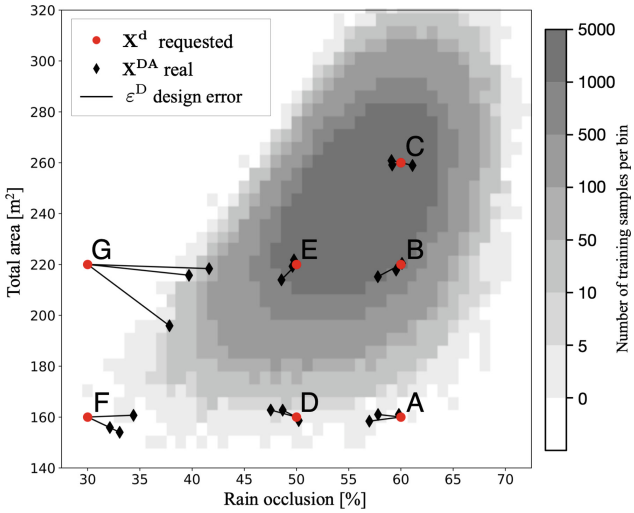

Fig. 8. Design errors for rain occlusion and total area, points A–G, and the distribution of training samples values $\mathbf{X}$ (grey heat map). Out of the best 1% designs provided for each request, we depict 3, randomly selected.

For the rain occlusion, the design error is smaller for performance values for which the model observed more data (Fig. 10a). For the total area attribute (Fig. 10b), however, the error increases as the attribute values grow – the errors per platform accumulate. Nevertheless, both cases lead to accurate designs for the best 10% of the samples per

bin: <2 pp for the rain occlusion and <5 m^2 for total area, for almost all the distribution. Studying multiple performance values allows to detect areas where the attributes' correlations lead to unfeasible requests, e.g. higher error in bottom-right corner in Fig. 10c–d. Besides, it also helps discovering areas that intuitively seem unfeasible, but where the AE model can still generate accurate designs (e.g. <5 pp for a high rain occlusion of 64% and much lower sun occlusion of 21.7%, as observed in Fig. 10c–d).

The trained encoder has an important role as surrogate model: to perform the selection of the best designs, and to allow an agile and time-efficient exploration of the solution space. In Fig. 10a–b we additionally plot the model error (red dashed-line). In both cases its value is small (∼1 pp and 1 m^2 respectively). This helps concluding that the

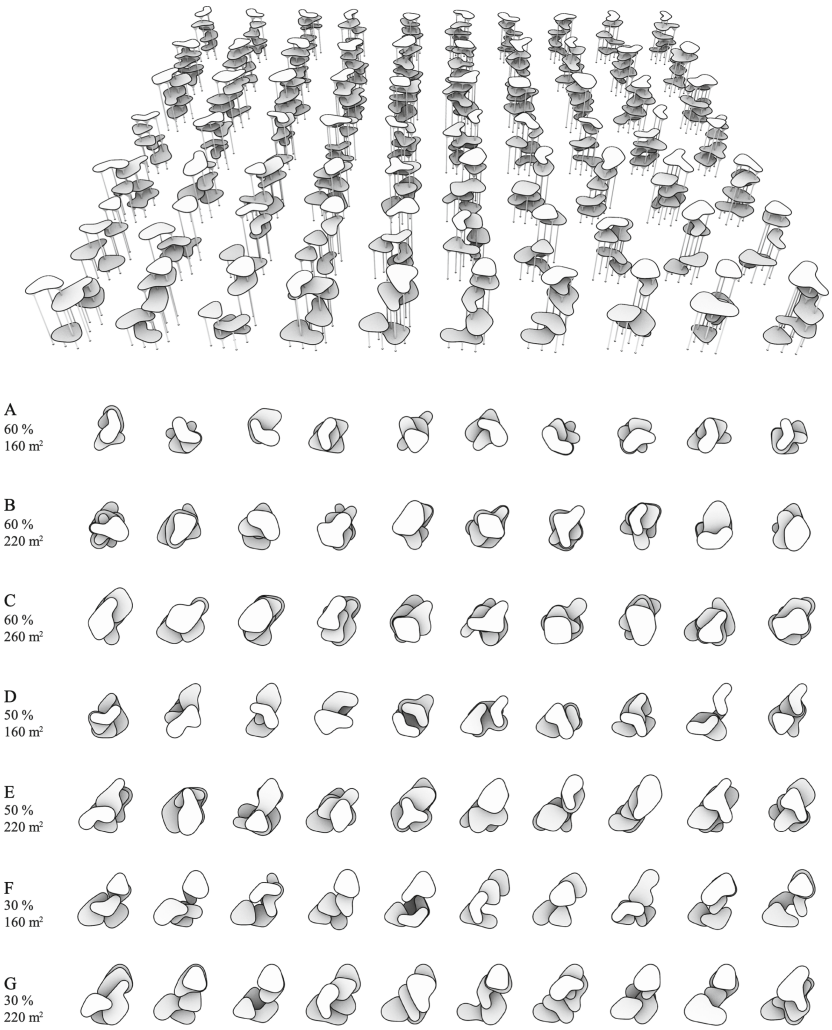

Fig. 9. Exemplary designs for seven different requests of rain occlusion and total area in perspective and top view.

encoder provides on average predictions $\tilde{\mathbf{X}}^{\mathbf{d}}$ similar to those calculated in the DA model, verifying its validity for the selection the best designs.

5 Application in Architectural Design

5.1 Design Workflow

The overall design workflow using the proposed approach consists of the following steps: 1) set up the project- specific parametric design+analysis (DA) model, 2) generate the dataset, 3) train the AE model, and finally 4) deploy the trained model to explore similarly performing design options, as explained in 3.3. The DA model can be used three-fold: a) to create an instance of the design for input parameters $\mathbf{W}$ given explicitly by the user (optional), b) to generate the dataset using input parameters $\mathbf{W}$ sampled randomly in step 2, and c) to construct and analyse design instances from design parameters $\mathbf{W}^{\mathbf{d}}$ predicted by the AE model for the aforementioned design exploration in step 4 (see Fig. 11).

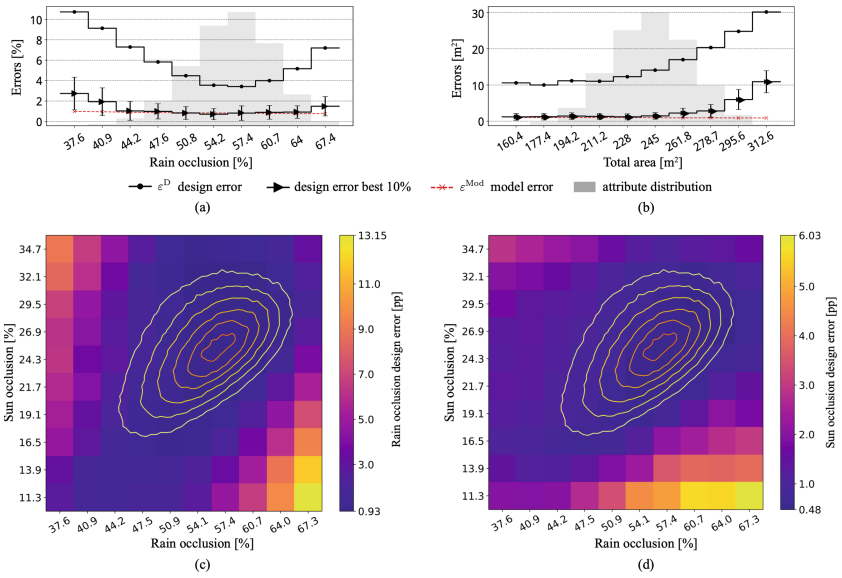

Fig. 10. In (a) and (b), average design and model errors. For a simultaneous request of rain and sun occlusion, rain (c) and sun (d) design errors for the best 10% designs. The isocurves depict the distribution in $\mathbf{X}$ of rain and sun occlusion.

5.2 Implementation

In the presented case study, the project-specific DA model was built using Grasshopper (McNeel and Rutten, 2010) and GhPython. The dataset $D = \{\mathbf{W},\mathbf{X}\}$ consisting of 470 000 random designs was split into a training (90% of samples) and a validation set. The

latter is used for hyperparameters' tuning of the weighting terms and the learning rate using GPyOpt (The GPyOpt authors 2016), process that took approximately 72 h on a machine with a single GPU (Tesla P100). The AE model, implemented using PyTorch (Paszke et al. 2019), is finally trained for 200 epochs with a batch size of 1024, using Adam optimizer with default values. The script for the rollout of the trained model runs as a server outside of the Rhino environment and communicates with the DA model in Grasshopper over a http protocol[1]. The requests are processed quickly, e.g. returning 100 design within <5 s.

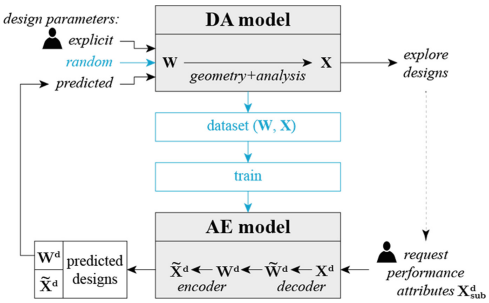

Fig. 11. The overall proposed workflow using the parametric design+analysis (DA) model and the autoencoder (AE) model. The pawn symbol indicates user input.

5.3 Application in the Case Study

The implemented model was used by an architect, who was not involved in the development of the presented tool, in an early design stage of the project. By interacting with it, the initial target performance values from the project brief were revised, as well as observed that sun occlusion is usually sufficiently satisfied and can be omitted in the request. For the selection of the final design, the architect requested 100 solutions, from which they discarded designs unsuitable according to further criteria not considered in the DA model, and finally chose the winning design based on other values such as aesthetics.

5.4 Observations

We observed that the solutions proposed by the autoencoder are very versatile, both geometrically (as exemplified by a selection in Fig. 12), and in terms of strategies to accomplish the requested performance, some of which the architect did not intuitively anticipate. For example, the autoencoder "discovered" that a needed percentage of rain-exposed areas can be achieved by having large platforms at the top and very small ones at the bottom (Fig. 13). Another observation was that requesting a small total area with low occlusions will result in small platforms spread far apart (compare designs A and F

[1] Video demonstrating the design exploration process here.

in Fig. 9). In both cases such solutions might be unfavourable, e.g. for structural reasons, and to curb the first effect, the user can opt to explicitly request a small area for the top platform. Overall, such findings help the designer understand the solution space and how the performance values translate into geometries.

6 Discussion and Outlook

We observed that the AE model undoubtedly can present many versatile solutions, and that this variety can give valuable insights that enhance the architect's understanding of the design task. Additionally, the analysis of the dataset can assist the architect in setting up and revising the DA model, e.g. by revealing hidden correlations.

Fig. 12. Different geometries generated from design parameters produced by the AE model, all for the same requested performance attributes.

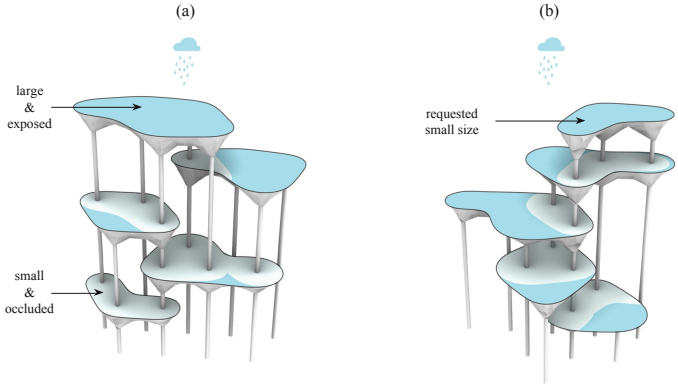

Fig. 13. Example of a strategy found by the AE model to easily achieve a large area exposed to rain though a very large top platform (a). The user can counteract it by requesting explicitly a small top platform (b). Both designs have the same total area and rain occlusion.

Throughout the pipeline, from the way the DA model is implemented to the human exploration and the AE methodology, there are biases. To understand them better, the future work could conduct studies on the data diversity during generation, biases on the encoding process, etc. Nonetheless, we propose a human-in-the-loop interactive approach as we believe is the best way to tackle these biases. By exploring the solutions, the designer can learn from them, tune the parameters accordingly, and finally understand the implicit biases and adjust to them.

For a widespread adoption, the methodology should allow a seamless and intuitive interaction. In our proof-of- concept, the trained model returns up to 100 designs within a few seconds, making it fairly interactive, and enabling a deep and exhaustive exploration of possible solutions.

In future work, we aim to generalize the tool to not only accept vectorial inputs, but all type of data presentations (graphs, images, etc.). This will require using alternative deep learning architectures, such as convolutional neural networks. Besides, other generative models, based on GANs or variational AE, will be implemented and tested on similar use cases.

Acknowledgements. We would like to thank Sarah Schneider of Gramazio Kohler Research, ETH Zürich, and the planning team of the Semiramis project for the valuable user feedback from the architect's perspective. This project was supported by the Swiss Data Science Center (project C20-09).

Contributions. Matthias Kohler developed the basic concept and the initial software prototype for the case study. Luis Salamanca and Fernando Pérez-Cruz improved the methodology beyond this initial idea, implementing the final AE model, and carrying out the analyses for performance, creativity, etc. Aleksandra Anna Apolinarska developed the final DA model, its parametrization and analysis, the interface to the AE model, and contributed to assessment of its performance.

References

Chaillou, S.: Space layouts & GANs Jan (2020). https://towardsdatascience.com/space-layouts-gans-19861519a5e9

Banga, S., Gehani, H., Bhilare, S., Patel, S., Kara, L.: 3D topology optimization using convolutional neural networks (2018). https://arxiv.org/pdf/1808.07440.pdf

Miguel, J.de, Villafañe, M.E., Piškorec, L., Sancho-Caparrini, F.: Deep form finding – using variational autoen- coders for deep form finding of structural typologies. In: Sousa, J.P., Xavier, J.P., Castro Henriques, G. (eds.) Architecture in the Age of the 4th Industrial Revolution – Proceedings of the 37th eCAADe and 23rd SIGraDi Conference, 11–13 Sep, vol. 1. University of Porto, Porto, Portugal (2019). http://papers.cumincad.org/data/works/att/ecaadesigradi2019_514.pdf

Costa, A., Nannicini, G.: RBFOpt: an open-source library for black-box optimization with costly function evaluations. Math. Program. Comput. **10**(4), 597–629 (2018). https://doi.org/10.1007/s12532-018-0144-7

Gramazio Kohler Research: Semiramis (2022). https://gramaziokohler.arch.ethz.ch/web/e/projekte/409.html

Hornby, G., Globus, A., Linden, D., Lohn, J.: Automated Antenna Design with Evolutionary Algorithms. Space AIAA SPACE Forum. American Institute of Aeronautics and Astronautics (2006). https://arc.aiaa.org/doi/ https://doi.org/10.2514/6.2006-7242

Hu, R., Huang, Z., Tang, Y., Van Kaick, O., Zhang, H., Huang, H.: Graph2plan: Learning floorplan generation from layout graphs. In: CM Transactions on Graphics 118-1 (2020)

McNeel, B., Rutten, D.: Grasshopper (2010). www.grasshopper3d.com

Mo, K., Guerrero, P., Yi, L., Su, H., Wonka, P., Mitra, N., Guibas, L.J.: Structurenet: Hierarchical graph networks for 3d shape generation (2019)

Nauata, N., Chang, K.-H., Cheng, C.-Y., Mori, G., Furukawa, Y.: House-GAN: relational generative adversarial networks for graph-constrained house layout generation. In: European Conference on Computer Vision, pp. 162–177. Springer (2020). https://arxiv.org/pdf/2003.06988.pdf

Oh, S., Jung, Y., Kim, S., Lee, I., Kang, N.: Deep generative design: integration of topology optimization and generative models. J. Mech. Des. **141**(11) (2019). https://asmedigitalcollection.asme.org/mechanicaldesign/article/141/11/111405/955342/Deep-Generative-Design-Integration-of-Topology

Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems, vol. 32, pp. 8024–8035. Curran Associates (2019)

Sohn, K., Lee, H., Yan, X.: Learning structured output representation using deep conditional generative models. In: Advances in Neural Information Processing Systems, pp. 3483–3491. Curran Associate (2015)

Steinfeld, K., Park, K., Menges, A., Walker, S.: Fresh eyes: a framework for the application of machine learning to generative architectural design, and a report of activities at smartgeometry 2018. In: Computer- Aided Architectural Design."Hello, Culture", Communications in Computer and Information Science, pp. 32–46. Springer, Singapore (2019)

Takahashi, Y., Suzuki, Y., Todoroki, A.: Convolutional neural network-based topology optimization (CNN-TO) by estimating sensitivity of compliance from material distribution (2019). http://arxiv.org/abs/2001.00635

Talton, J., Yang, L., Kumar, R., Lim, M., Goodman, N., Mech, R.: Learning design patterns with bayesian grammar induction. In: Proceedings of the 25th annual ACM symposium on User interface software and technology, pp. 63–74 (2012)

The GPyOpt authors: GPyOpt: a Bayesian optimization framework in python. (2016). http://github.com/SheffieldML/GPyOpt

Brown, C.T., Mueller, N.C.: Design variable analysis and generation for performance-based parametric modeling in architecture. Int. J. Arch. Comput. **17**(1), 36–52 (2019). https://doi.org/ https://doi.org/10.1177/1478077118799491