APPLIED GRAPH MACHINE LEARNING in architecture

**Abstract.** Building on previous research on graph machine learning in architecture, this paper investigates topics that so far have been discussed only briefly or not at all. This includes data generation and -representation, readability, understanding, accessibility and use of software. We primarily focus on understanding how the graph autoencoder deals with differently structured datasets as input, the look and distribution of the latent space and further the interpretation of the resulting, newly generated 3D models.   
We ask how to generate a dataset that, compared with previous results, will be distributed more heterogeneously in the latent space to enable the graph autoencoder to produce geometrically logical and original results. Better performance is achieved by improving the input rather than changing the architecture of the graph autoencoder. A new method of data generation as well as multiple options for data representations are tested and analyzed. Generated results are compared with each other and with the dataset for maximal understanding of the graph auto encoder.   
A second goal is improving the accessibility of the research. Instead of keeping the application only inside the Rhino/Grasshopper environment, we also use web-based tools to guarantee easy access for researchers and the public to understand the system further.

**Keywords.**  generative 3D architecture, generative graph machine learning, graph-based architecture, human-computer interaction, graph autoencoder, latent walk

1. Introduction

Talk about how old paper is continued with a shift in perspective, now putting efforts into the latent space distribution by working on data preparation.

Maybe also mention web based application in here and then not again to make everything clearer.

## 1.1 STATE OF THE ART

Current research in generative AI in architecture can be divided into 2D generative models based on images or vector graphics (floor plans), and 3D generative models based on stacked images, point clouds, voxel- or mesh geometries. 2D image generation has successfully been used the longest, due to its great accessibility and generalizability towards architecture. In addition, it also offers an almost seamless integration into the contemporary workflow of architectural design, requiring a minimum of postprocessing before the result is fully usable (i.e. a quick visualisation of a design idea only needs to be edited in photoshop if anything to be presentable as the designer's idea). No special research for the architectural domain was necessary, but architects can use models like DALL-E (Ramesh et al., 2021) and Stable Diffusion (Rombach et al., 2022). Only some big architecture offices like Zaha Hadid Architects (Ackerman, 2025) and Coop Himmelblau (Himmelb(l)au, n.d.) developed their own, in-house models being able to further control the output.

3D generation models are rarer due to the difficulty of implementation and need of custom solution for architectural purpose. In addition, “[..] personal barriers often restrict [the architects’] access to the latest technological developments, thereby causing the application of generative AI in architectural design to lag behind.” (Li et al., 2024). Another problem to solve is the complexity of any building and how to encode its information into any AI model as well as the lack of specific datasets to train on. Generic text-to-3d models (Lin et al., 2023; Poole et al., 2022) can be used, but they do not adapt as good in producing specific 3d architecture models than the 2d models do in generating images of buildings thanks to less complexity.

Most successful methods of encoding and generating 3d building information rely at least partially on 2d information. That could be floor plans, sections and depth information from which a 3d model is generated (Del Campo, 2022; Zhang & Blasetti, 2020). Other models rely on fully 3d based training, either encoding voxel data (Koh, 2020; Rasoulzadeh et al., 2024) or point clouds (Wei et al., 2023), which have improved in realism and logic recently but still tend to generate rather conceptually 3d models.

## 1.2 Previous research

Here we present an extended and improved version the graph-based generation method of 3d architectural models presented originally in Bauscher et al. (2024).The original paper presents a generative AI system that outputs new spatial configurations of architectural elements in three dimensions. The graph based autoencoder can read 3d buildings as training input and does not rely on any sort of conversion between 2d images or plans and a desired three-dimensional output in post. It presents a very intuitive approach to generate 3d models for architects, that although only simple, already shows some understanding of important architectural features that are used in generation. That makes it very well rooted in the modern idea of the architectural design process. In the following we discuss and improve its main weaknesses which mainly lay in data generation and representation.

1. Original Methodology

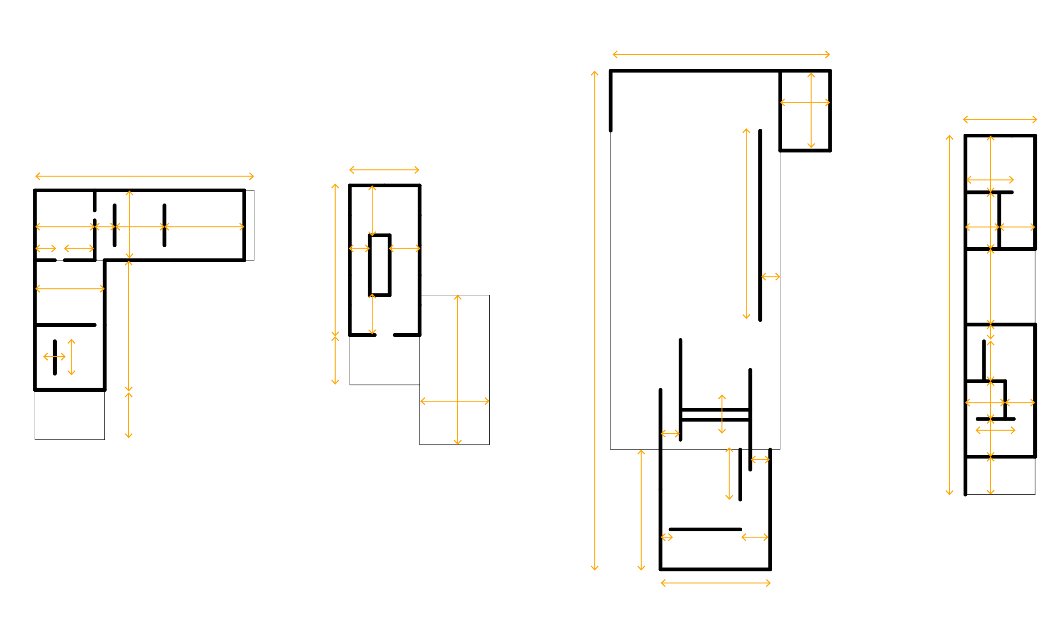
## 2.1 DATASET

The custom dataset originally used consists of four modernist architectures, that have been individually remodeled parametrically in Grasshopper inside Rhino. That gave the opportunity to easily augment each building in its geometry to create more data for training the autoencoder. All houses are from the same architectural period, modernism, for the ease or remodelling and a geometric continuity across all data. The buildings are:

* Mies van der Rohe's Barcelona Pavilion (1929)
* Ray and Charles Eames' Eames House (1949)
* Mies van der Rohe's Farnsworth House (1951)
* Pierre Koenig's Stahl House (1960)

Those houses all follow the design principle of only using orthogonal walls, as well as mostly having full height openings for windows. Yet they vary greatly in size, proportion and location. They are also all very well documented, two of them being part of the case study houses series (Eames, Koenig) (Smith, 2006) while the other being two of the best-known examples of modernist architecture (both Mies van der Rohe projects).

To avoid overcomplication in the 3D model the buildings were remodelled as a surface model. That means that walls, floors and ceilings are all seen as individual, rectangular surfaces without any thickness, while doors and openable windows are just empty left spaces. This supports the idea of trying to model space defining elements only, so there is no need for i.e. any structural considerations.

Figure 1 – Previous method of geometry augmentation in plan, orange highlights the parametric parameters.

This approach fails in providing the autoencoder with a heterogeneous dataset mainly due to the method of augmentation. Starting always from the original model, elements are only slightly moved and scaled randomly in one direction (Fig below), making the variation not being enough to allow for a smooth geometrical translation between the original buildings. The augmentations still always keep the same layout, element count, division between horizontal and vertical elements, overall size and ratio.

The focus of change mainly lies on geometrical proportion. In addition, the augmented geometries are randomly rotated around their central Z axis in increments of 90 degrees, as well as randomly mirrored in X and Y direction. On one hand this creates a more diversified and interesting space for generation, on the other hand however it introduces unnecessary noise into the latent space and thus makes the results very hard to read and understand.

## 2.2 DATA REPRESENTATION

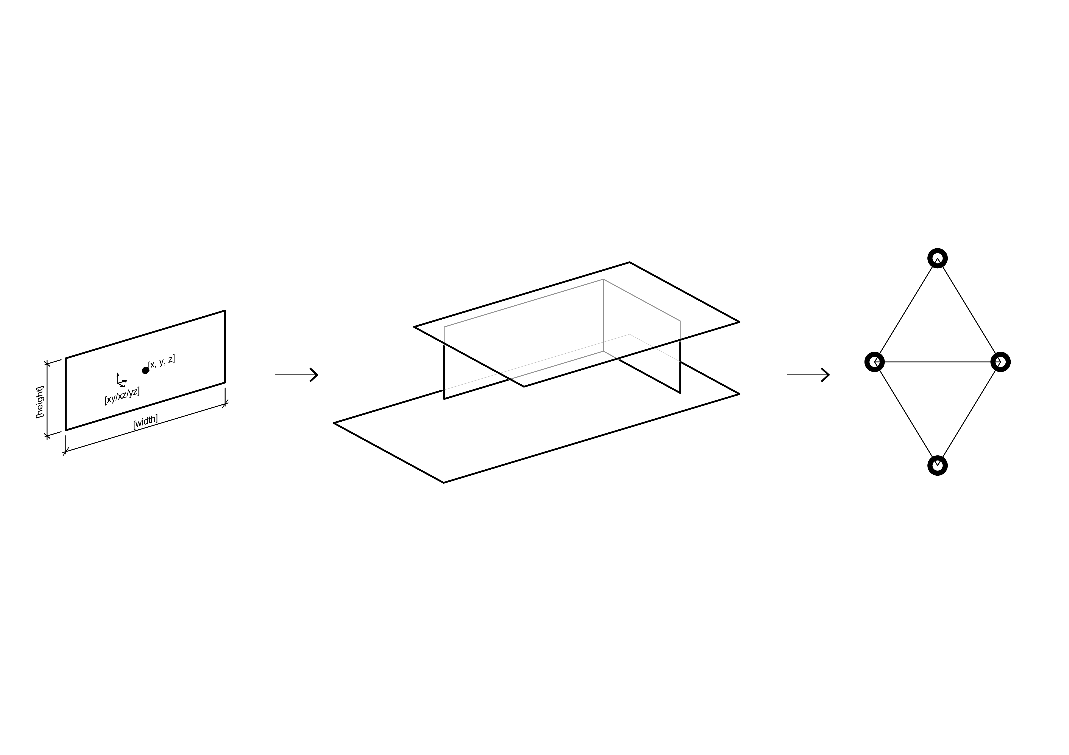
To make a 3D building machine-readable, it is converted into a graph. Graphs have been used for representing architectural space and buildings for a long time (Alexander, 1977; Hillier, 1996) and are still used today in the databased environment of architecture (Elshani et al., 2022; McGlinn & Pauwels, 2022; Rasmussen et al., 2020). They can capture more information than most other data formats due to their non-discursive nature, thus provide the perfect basis for making 3D models of architecture machine readable.

Figure 2 – Graph representation of a surface based 3D model, as used in Bauscher et al. (2024)

Each surface-based building in the dataset is translated into an undirected graph, where all geometrical information is held in the node features. Each node represents one surface, and nodes are connected in the graph if they physically touch in the 3D model (Fig. 2). To encoder all necessary information, the node features contain the coordinates of the centre point, the orientation and the width and length of the surface. Here we must point out that this specific method of conversion only allows for orthogonally positioned elements. The orientation value equals one of the three global planes (XY, XZ, YZ) and thus limits the geometrical options for input and output.

## 2.3 GRAPH AUTOENCODER

Figure 3 – Model architecture, as applied in Bauscher et al. (2024)

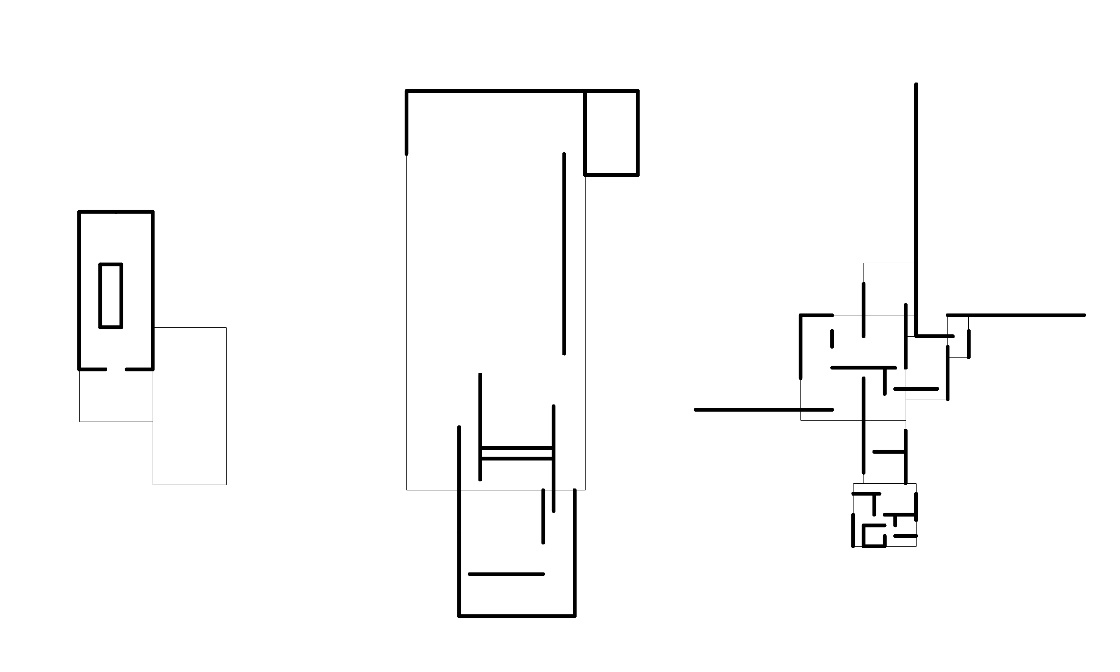
The auto encoder model (Fig. 3) is defined by encoder and decoder model. Here the encoder is a graph-based encoder as first described by Kipf and Welling (2016) consisting of message passing layers (Hamilton et al., 2018), while the decoder consists of linear layers that do not create or use any graph structure. That means all input data must have the same count of elements, because reconstruction the features though linear layers flattens the data into a one-dimensional vector with always the same length. While this poses some limitations to the system, the advantages of simplicity in implementation still outweigh them (Guo & Zhao, 2022).

Another limitation is the chosen dimension of the latent space - three - which restricts the performance of the model. Yet again, the simplicity and possibility of visualization outweigh the disadvantages, also considering that the model itself might be the least deciding factor on the performance of the whole system, and data collection and preparation are far more important (Jarrahi et al., 2023).

1. Adapted Methodology

## 3.1 DATASET

Aroyo et al. (2021) point out: “Data is potentially the most under-valued and de-glamorized aspect of today’s AI ecosystem” and “Benchmark datasets are often missing much of the natural ambiguity of the real world”. Both points can be applied to our original research on graph autoencoders in architecture. The second point is harder to approach due to the limiting number of three-dimensional datasets of buildings ready to use. The usage of a 2D floor plan dataset as they are widely available (de las Heras et al., 2015; Kalervo et al., 2019; Wu et al., 2019) seems to not only defeat the conceptual idea of the research project but also still pose multiple questions on the quality and origin of the data.

Figure 4 –Plans Mies van der Rohe. Left: Farnsworth House, Centre: Barcelona Pavillon, Right: Brick Country House. Original plans redrawn by the author.

Therefore, the first point quoted can be addressed by further developing the existing dataset towards a more heterogeneous landscape for better overall performance. We shrank the original number of houses from four to three, continuing development with Mies van der Rohe’s Farnsworth House and Barcelona Pavillon, as well as a new addition of the conceptional plans drawn around 1923 for the Brick Country House (Fig. 4). These buildings now also function more as conceptional assistants rather than direct input for augmentation, as the method relies now on an iterative process that starts from scratch rather than modifying existing geometry. By choosing architectures by the same architect as guidelines, which differ in size, typology, proportion and layout but follow similar geometrical design principles (open floor plan, grids, horizontal planes), we open the room for a more seamless transition between one another.

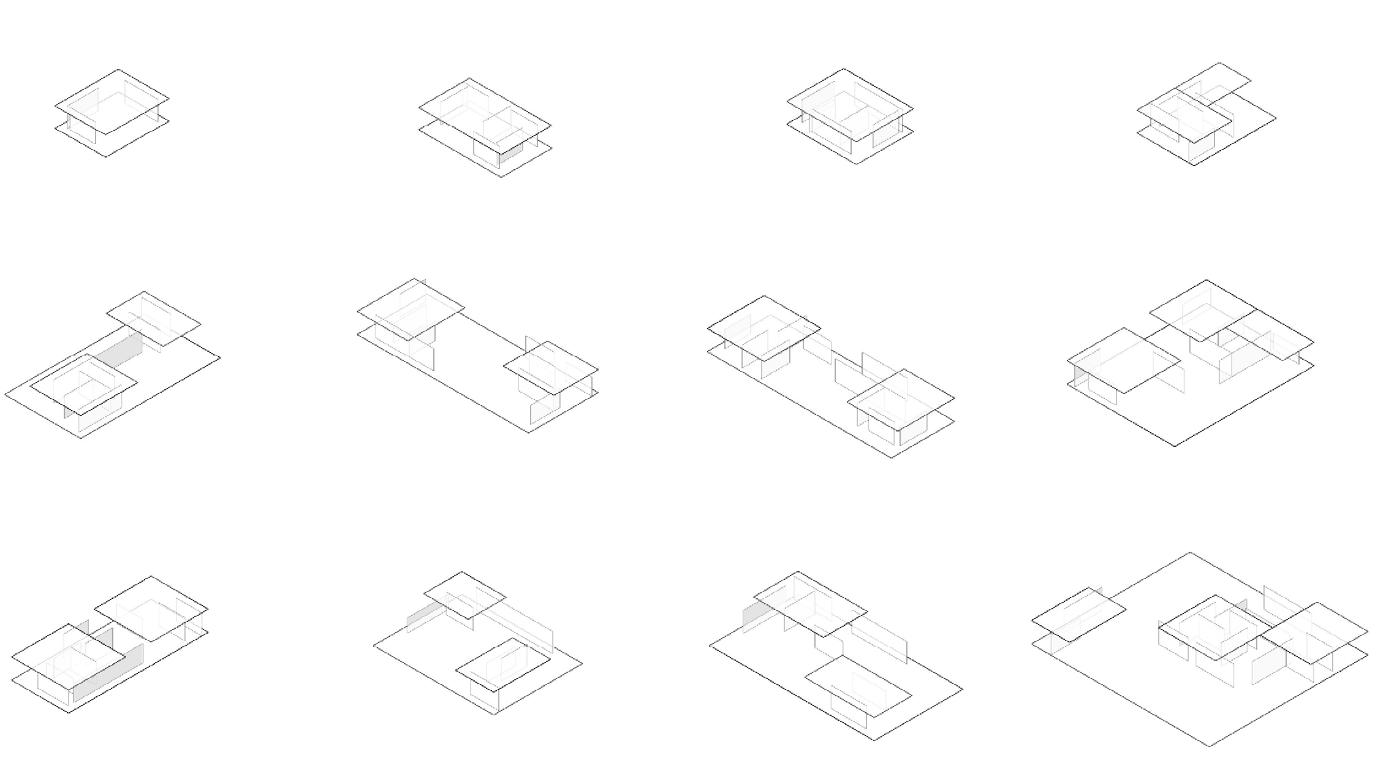
As a main driver for the definition of space in Mies van der Rohes geometries we identified the roof shapes. These rectangles are initially placed in the generative algorithm which is implemented in the Grasshopper/Rhino environment. The roof rectangles are then further subdivided into the subspaces over which they lay. For each edge of the subspace, a wall is placed in a random position with a random length. Finally, a floor geometry is generated by creating a boundary rectangle around the wall and roof projections (Fig. 5).

Figure 5 – Example of generated models in the style of Mies van der Rohe

## 3.2 DATA REPRESENTATION

Figure 6 – Options for representation of a 3D model as a graph

As the used conversion process from 3D model to graph and back has proved to be highly influential on the generated outcome from the AI system, we introduce now three options (Fig below) then can be compared against each other for more transparency and understanding.

Option A is what was used before and generated controllable and valuable results, but with limitations. Each surface in the model thus node in graph is represented by its center point, width and length, and orientation regarding the global world planes. While movement in X, Y, Z as well as changes in width and height is continuous, the rotation values are discrete and allow for three different cases.

Option C defines the surface by its corner points instead of width – height – center. This option offers most geometrical freedom but also has the highest potential of creating faulty and illogical geometries if the concept of the corner points is not learned properly be the autoencoder.

## A diagram of a diagram of a structure Description automatically generated with medium confidence2.3 GRAPH AUTOENCODER

Figure 7 - Graph Autoencoder Model with variable node count in training data

The graph autoencoder model (Fig. 7) stays mostly the same as described in paragraph 1.3. The main limitation before was due to the decoder architecture, the buildings in the dataset must all be made up by the same number of surfaces, which now is the only change to the model. We introduce masking to the decoder, which now decodes the latent vector to be a one-dimensional vector as big as the input data with the greatest number of elements. If less elements have been input, we use a mask to decode only as many elements as needed. This mask is predicted via a second branch in the decoder with three linear layers. The final model used the following variables:

- Epochs: 1000

- Learning Rate: 0.001

- Optimiser: Adam

- Loss function: Mean Squared Error

- Activation function: ReLu

- Train, Validate, Test: 80%, 10%, 10%

1. Results

A cube with different colored blocks

Description automatically generatedA transparent box with small black dots

Description automatically generated with medium confidenceThe geometries in the old dataset were augmented including mirroring and rotating the entire building, which highly influenced the generation from the latent space. This noise was eliminated in the current model, creating a more readable and unsurprising latent space (Fig. 8). The latent variables can further be interpreted, while before those variables were seemingly random.

Figure 8 – Latent Spaces, left: Old Model, right: New Model

1. Accessibility

Instead of relying purely on the Rhino and Grasshopper environment as done before, we use web-based tools to display the trained model quickly in the browser. Before, accessing the model, which is implemented using the Python machine learning framework PyTorch (Paszke et al., 2019) and its library PyTorch Geometric (Fey & Lenssen, 2019), was done inside Grasshopper using the HOPS component. While this works, the process is laborious, needs a lot of attention to make sure the right Grasshopper-Spaghetti are where they are supposed to be, and requires multiple software to run at the same time.

This is now streamlined, where the python script executing the training of the ML model saves the model, its performance statistics and latent space information into a specific location from where it can be accessed by the web-based application. This application (Fig. 8) is using three.(*Three.Js – JavaScript 3D Library*, n.d.) to display generated 3d models and latent space, gives control for the latent space variables and lets the user quickly switch between the different trained models (which before was the hardest part to do in Grasshopper).

As already proved previously by the physical application involving a MIDI controller communication live with Grasshopper and HOPS, this improves general access and understanding for both developer and architect immensely and should be persuaded more by academic research for architectural applications.

1. Conclusion

* Better readable results thanks to less noise in the dataset from bad augmentation.
* While the deterministic model already provides us with new geometries that follow our rule set to create “Miesian” 3d models, we argue that using a generative autoencoder gives us multiple advantages:
  + It can adapt to large datasets, while a deterministic model would need to be rewritten
  + It offers a smooth interpolation between known designs, which can lead to unexpected new forms. A deterministic model might struggle to create “meaningful hybrids”
  + More parameters can be included in the future, training the model to prefer optimized designs
* Next steps could include:
  + further testing, optimizing and publishing the web-based application of the model
  + redesigning the model to be a neurosymbolic AI system, combining a graph-based NN with a rule-based NN
  + a fact based evaluation method for training and evaluating of the model.

1. References

Ackerman, N. (2025, January 2). Zaha Hadid Architects builds ‘winner proposals’ with AI. https://www.thetimes.com/business-money/entrepreneurs/article/zaha-hadid-architects-builds-winner-proposals-with-ai-enterprise-network-qs7m7txwz

Alexander, C. (1977). A Pattern Language. Oxford University Press.

Aroyo, L., Lease, M., Paritosh, P., & Schaekermann, M. (2021). Data Excellence for AI: Why Should You Care (No. arXiv:2111.10391). arXiv. https://doi.org/10.48550/arXiv.2111.10391

Bauscher, E., Dai, A., Elshani, D., & Wortmann, T. (2024, November 18). (PDF) Learning and Generating Spatial Concepts of Modernist Architecture via Graph Machine Learning. ResearchGate. https://doi.org/10.52842/conf.caadria.2024.1.159

Del Campo, M. (2022). Deep House—Datasets, estrangement, and the problem of the new. Architectural Intelligence, 1(1), 12. https://doi.org/10.1007/s44223-022-00013-w

Elshani, D., Lombardi, A., Fisher, A., Hernandez, D., Staab, S., & Wortmann, T. (2022, May). Knowledge Graphs for Multidisciplinary Co-Design: Introducing RDF to BHoM. ESWC - LDAC 2022.

Fey, M., & Lenssen, J. E. (2019). Fast Graph Representation Learning with PyTorch Geometric (No. arXiv:1903.02428). arXiv. https://doi.org/10.48550/arXiv.1903.02428

Guo, X., & Zhao, L. (2022). A Systematic Survey on Deep Generative Models for Graph Generation (No. arXiv:2007.06686). arXiv. https://doi.org/10.48550/arXiv.2007.06686

Hamilton, W. L., Ying, R., & Leskovec, J. (2018). Inductive Representation Learning on Large Graphs (No. arXiv:1706.02216). arXiv. http://arxiv.org/abs/1706.02216

Hillier, B. (1996). Space Is The Machine: A Configurational Theory Of Architecture.

Himmelb(l)au, C. (n.d.). Deep Himmelblau. Coop Himmelb(l)Au. Retrieved 2 February 2025, from https://coop-himmelblau.at/method/deep-himmelblau/

Jarrahi, M. H., Memariani, A., & Guha, S. (2023). The Principles of Data-Centric AI (DCAI). Communications of the ACM, 66(8), 84–92. https://doi.org/10.1145/3571724

Kipf, T. N., & Welling, M. (2016). Variational Graph Auto-Encoders (No. arXiv:1611.07308). arXiv. https://doi.org/10.48550/arXiv.1611.07308

Koh, I. (2020). Voxel Synthesis for Architectural Design.

Li, C., Zhang, T., Du, X., Zhang, Y., & Xie, H. (2024). Generative AI Models for Different Steps in Architectural Design: A Literature Review (No. arXiv:2404.01335). arXiv. https://doi.org/10.48550/arXiv.2404.01335

Lin, C.-H., Gao, J., Tang, L., Takikawa, T., Zeng, X., Huang, X., Kreis, K., Fidler, S., Liu, M.-Y., & Lin, T.-Y. (2023). Magic3D: High-Resolution Text-to-3D Content Creation. 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 300–309. https://doi.org/10.1109/CVPR52729.2023.00037

McGlinn, K., & Pauwels, P. (Eds.). (2022). Buildings and Semantics: Data Models and Web Technologies for the Built Environment. CRC Press. https://doi.org/10.1201/9781003204381

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library (No. arXiv:1912.01703). arXiv. https://doi.org/10.48550/arXiv.1912.01703

Poole, B., Jain, A., Barron, J. T., & Mildenhall, B. (2022). DreamFusion: Text-to-3D using 2D Diffusion (No. arXiv:2209.14988). arXiv. https://doi.org/10.48550/arXiv.2209.14988

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., & Sutskever, I. (2021). Zero-Shot Text-to-Image Generation (No. arXiv:2102.12092). arXiv. https://doi.org/10.48550/arXiv.2102.12092

Rasmussen, M. H., Lefrançois, M., Schneider, G., & Pauwels, P. (2020). BOT: The Building Topology Ontology of the W3C Linked Building Data Group. Semantic Web. https://doi.org/10.3233/SW-200385

Rasoulzadeh, S., Bank, M., Wimmer, M., Kovacic, I., Schinegger, K., & Rutzinger, S. (2024). ArchComplete: Autoregressive 3D Architectural Design Generation with Hierarchical Diffusion-Based Upsampling (No. arXiv:2412.17957). arXiv. https://doi.org/10.48550/arXiv.2412.17957

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-Resolution Image Synthesis with Latent Diffusion Models (No. arXiv:2112.10752). arXiv. https://doi.org/10.48550/arXiv.2112.10752

Smith, E. A. T. (with Gössel, P.). (2006). Case study houses: 1945-1966 : the Californian impetus. Taschen.

Three.js – JavaScript 3D Library. (n.d.). Retrieved 26 January 2025, from https://threejs.org/

Wei, Y., Vosselman, G., & Yang, M. Y. (2023, August 31). BuilDiff: 3D Building Shape Generation using Single-Image Conditional Point Cloud Diffusion Models. arXiv.Org. https://arxiv.org/abs/2309.00158v1

Zhang, H., & Blasetti, E. (2020). 3D ARCHITECTURAL FORM STYLE TRANSFER THROUGH MACHINE LEARNING. MACHINE LEARNING.

NOTES ON TRAINING:

* Changing the latent variables to 6: better performance by 0.5 in loss, but seems to be a shift only, so if it was at 3 before its gonna end at 2.5. it doesn’t change anything fundamentally
* Changing the latent variables to 32: performance worse than 6, while still a bit better than 3: might be already overfitting
* Changing to a different pooling method in the encoder (set2set) improves performance as well by around 0.5 in loss, also making the model learn longer before converging. **Might be good to try more there….**
* Changing the processing steps of set2set from 2 to 4 makes model training very slow
* Changing pooling from 2nd dimension to 1st. as in instead of 29x128 -> 29x3 -> 1x3, doing 29x128 -> 10x3 -> 1x3, so using both dimensions, confuses the model and learning is very jiggle and nowhere near good performance (around 10 loss)