

CARNEGIE MELLON UNIVERSITY

School of Architecture

College of Fine Arts

Thesis

Submitted in Partial Fulfillment of the requirements for the degree of

Master of Science in Computational Design

TITLE:

Explore Generative AI Moldes for Vector-based 3D Architecture
Data

AUTHOR:

Yujin Wu

ACCEPTED BY ADVISORY COMMITTEE:

Daragh Byrne
Professor NAME



Principal Advisor

May 6th, 2025

DATE

Deva Ramanan
Professor NAME

Advisor

05/06/25

DATE



EXPLORE GENERATIVE AI MODELS FOR VECTOR- BASED 3D ARCHITECTURE DATA

Abstract

In recent years, we have witnessed a notable surge in generative AI from Large language models to image diffusion models. It appears we are on the brink of an era in which, given sufficient data, generative AI models can learn the data distribution of any dataset and synthesize new, high-fidelity examples. This thesis explores the application of this idea in architecture, focusing on the generation of vector-based 3D architectural data. Emerging state-of-the-art generative AI models—such as MeshGPT—have demonstrated the ability to generate 3D vector data, including meshes, wireframes, and boundary representations (B-reps). To explore whether such models can learn human spatial habits and formal conventions, I trained MeshGPT on the 3DBAG dataset to generate buildings with diverse roof types and morphologies from text prompts. The thesis evaluates the feasibility of this approach in relation to current model capabilities, architectural datasets, and design needs. Evaluation results show that while the model generates accurate outputs for common prompts, it struggles with rare typologies and more complex descriptions.

Yujin Wu

yujinwu@andrew.cmu.edu

Advisors:

Daragh Byrne

Deva Ramanan

Acknowledgements

I am deeply grateful for the continued support I have received throughout my thesis journey. Firstly, I would like to thank my primary advisor, Prof. Daragh Byrne. Without his guidance, it would not have been possible to scale the thesis to a suitable scope and complete it on time. I would also like to thank my secondary advisor, Prof. Deva Ramanan, who helped clarify numerous technical details related to the project.

My sincere thanks also go to Prof. Levent Burak Kara and Ava Pun, whose insights into model data preprocessing and training were invaluable. I am especially thankful to my friends and fellow MSCD cohort members, who have helped sharpen my understanding of my own work through discussions and constructive feedback. Special thanks to Chen Yin, who struggled alongside me throughout the final semester and offered immense emotional support.

Lastly, I would like to thank Jianhan for his unwavering encouragement and support in pursuing my passion.

Contents

Acknowledgements.....	3
Annotated Table of Contents.....	7
Introduction	8
Problem Space.....	8
Research Questions	10
Goals and Expected Result.....	10
Hypothesis	11
Background Review.....	11
2D vector floorplan generation	12
3D data generation	13
Raster Formats	13
Vector Formats	14
Other 3D Representations.....	15
Available 3D Datasets	16
Method	18
Precedent: House Gable Project	18
Comparison of different 3D vector generative models	20
Data Preprocessing	23
Prepare the 3D models.	23
Caption Generation	25
Model Architecture Design	27
Evaluation Method	27
Result	28
Model Caption.....	28
Keywords Analysis.....	29
Embedding model analysis: BGE and Embedding Space Exploration	32
MeshGPT	34
Autoencoder Training	34
Transformer Training.....	35
Trends, Success and Failures	35
LlamaMesh	40

Reflection and Next Step	42
Revisiting Kyle Steinfeld’s theory on AI-human relationship	42
Next Step	43
Bibliography	45
Appendix.....	48
Preliminary Testing of General Model Capabilities	48
Plan for Cross Model Benchmarking Evaluation	49
BrepGen	49

Figure 1 Post-processing steps are required to correct boundary lines that are not perfectly straight.	8
Figure 2: a commercial website, techarch.ai [6], that can generate realistic architecture rendering.	9
Figure 3 Different type of roofs in Atlanta, Georgia	18
Figure 5 The caption evaluation UI. In this case, the generated prompt is "an elongated structure with a central gabled roof and lower flat-roofed extensions on both sides, creating an asymmetrical layout"	28
Figure 6 Normalized keywords frequency diagram. Gabled roof is most common in our dataset.	31
Figure 7 Normalized co-occurrence matrix for keywords	32
Figure 8 Embedding space analysis of embedding generated by bge model	33
Figure 9 Training loss for MeshGPT Autoencoder	34
Figure 10 Training loss for MeshGPT transformer	35
Figure 11 Two meshes generated from the prompt "a compact structure featuring a steep gabled roof with an asymmetrical extension on one side"	37
Figure 12 Examples of exploded mesh during inference	39
Figure 13 Training loss graph of LlamaMesh model.	40
Figure 14 Generated mesh after finetuning	40
Figure 15 An example of houses that are autoregressively generated MeshGPT	48
Table 1 Compare four generative models: MeshGPT, 3DWire, LlamaMesh, BrepGen	20
Table 2 Compare the cost of GPT-4o and GPT-4o-mini	25
Table 3 Examples of caption quality	29
Table 4 Accuracy count of text-mesh consistency	36
Table 5 Accuracy rate of text-mesh consistency.	36
Table 6 Examples of inaccurate captions which should be "easy"	38

Annotated Table of Contents

1. Introduction

This section provides an overview of the integration of AI in architecture. It mentions challenges with raster-based approaches in CAD workflows and highlights the shift towards generative models that output vector data. It also narrows the problem space to building envelope generation. It then proposes the research question, goals, expected result and hypothesis.

2. Background and Literature Review

This section starts with an overview of architectural CAD software. It then provides a review of raster and vector approaches to generate 3D shapes and their limitations. It then compares various 3D datasets.

3. Methodology

This section outlines experiment setup, dataset cleaning and caption generation procedures.

4. Results and Analysis

This section documents the experiment result with 3DBAG dataset and MeshGPT model.

5. Reflections and Future Work

This section reflects the limitations of current approach and outlines the next steps to improve the pipeline.

Introduction

Problem Space

The integration of artificial intelligence (AI) into architecture design has sparked innovative approaches and exciting possibilities. Researchers have explored various AI techniques, from classification models predicting building styles based on images [27] to segmentation methods extracting Building Information Modeling (BIM) data from scanned point clouds [3]. Among many, the task of generating new architecture is especially exciting. Since the rise of generative adversarial networks (GANs) as one of the earliest image generation models, we have seen the application of generative AI (GenAI) models in tasks such as generating 2D building footprints, unit floor plans, and building facades. There are also research projects that aim to synthesis 3D buildings, either directly as voxels representation by making use of 3D convolution [22] or

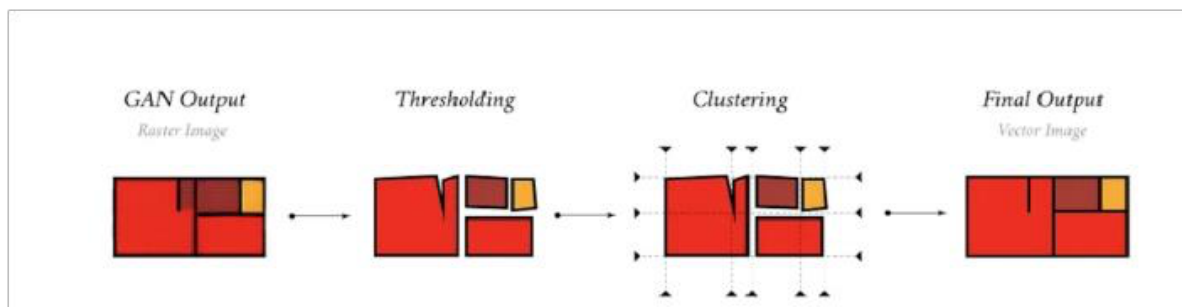


Figure 1 Post-processing steps are required to correct boundary lines that are not perfectly straight.

generate 2D images layer by layer and interpolate between the layers [23].

However, previous work in generative AI for architectural design has primarily relied on raster data formats, such as 2D images or 3D voxels, which represent spatial information as ordered n-dimensional arrays. This reliance stems from the early dominance of convolutional neural networks (CNNs) architecture in deep learning and CNNs can only handle raster data. Nonetheless, raster data poses a significant limitation: it is incompatible with the native workflows of computer-aided design (CAD) software, which primarily uses vector-based data. Vector data formats are hierarchically organized and efficiently represent geometry, making rendering and editing easier. As a result, many AI-driven building generation methods require complex post-processing steps to convert raster data into vector formats, introducing noise and inefficiencies. Fig. 1 illustrates an example of such post-processing (thresholding and clustering) used to convert GAN-generated floorplans into vector data [4].

Arguably the most impactful recent application of AI in design has been Stable Diffusion, a generative model that produces high-quality images from text prompts. While its influence is widespread, particularly in architecture where it's used for quickly visualizing stylistic concepts, its outputs are limited to raster-based 2D images. These images offer only single-perspective views, making it challenging to maintain spatial consistency across multiple angles. For architectural workflows that demand an understanding of form in three dimensions, this is a significant constraint. Generating consistent multi-view renderings often involves complex post-processing or manual intervention, reducing the practicality of these tools beyond early-stage ideation. Consequently, diffusion models like Stable Diffusion, despite their creative power, remain largely disconnected from CAD-centric pipelines that rely on vector-based and volumetric data.

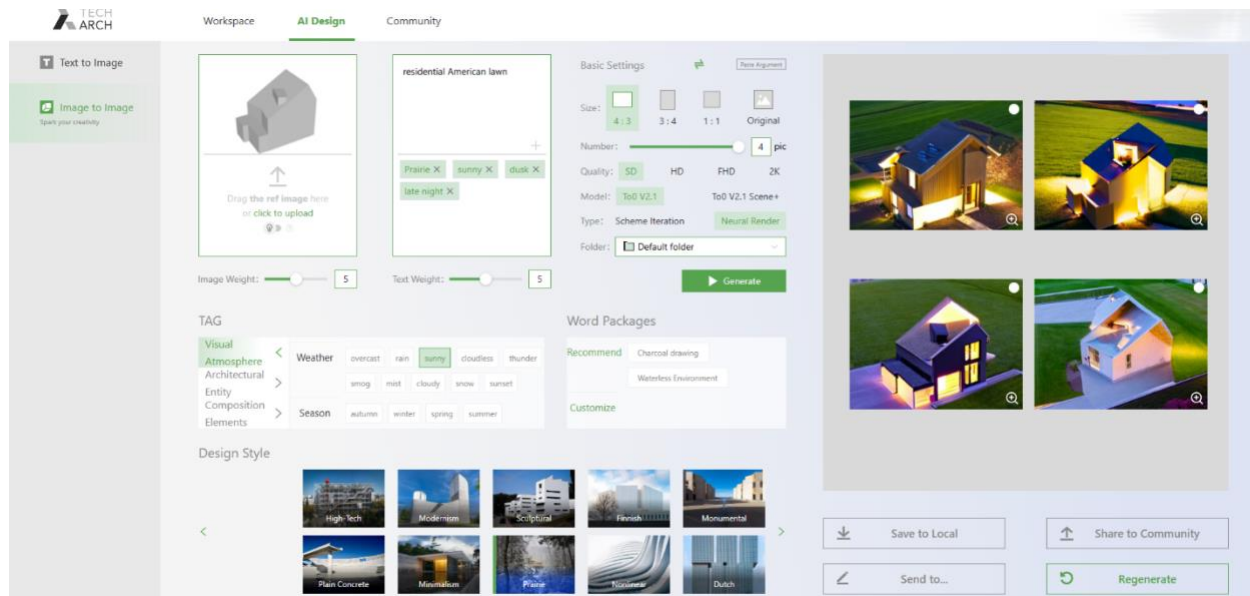


Figure 2 A commercial website, *techarch.ai* [6], that can generate realistic architecture rendering.

There is a growing shift toward generative AI models that output vector data directly, moving beyond traditional image- or voxel-based representations. In the 2D domain, this shift has catalyzed a surge of research into floor plan generation [6, 7] and indoor furniture layout synthesis [15]. The details of these 2D approaches and their applications will be further elaborated in the background review. Parallel to this, a newer strand of research has emerged around the generation of 3D vector geometric data. While not yet architecture-specific, recent advancements in autoregressive models [32, 38] and diffusion-based models [43], have demonstrated the ability to synthesize complex 3D outputs such as triangular meshes, wireframes, and even B-rep representations. However, these models have predominantly been tested on furniture or mechanical parts and are rarely adapted for the compositional complexity of architectural forms.

This thesis focuses on the generation of 3D building models, specifically targeting the building envelope rather than interior design. The building envelope, encompassing elements like walls, roofs, and facades, plays a critical role in architectural design, yet it remains an underexplored area in generative AI research compared to the more well-established domain of interior floor plan generation. While tools for generating detailed layouts and floor plans are available, the emphasis here is on developing AI models capable of synthesizing realistic, coherent exterior designs that align with the needs of architects during early-stage conceptualization. By prioritizing the building envelope, this thesis seeks to address a crucial gap in architectural AI applications, providing a foundation for creating editable 3D architectural models that integrate seamlessly into CAD workflows. By designing and training a GenAI model that can generate 3D vector-based architectural data, this research aims to bridge the gap between AI innovation and practical architectural applications. Such a model has the potential to significantly reduce the time architects spend on repetitive tasks, much like how tools like GitHub Copilot assist programmers in automating code generation. Additionally, it could serve as a powerful ideation tool, enabling architects to explore creative design possibilities more efficiently and with greater flexibility. Not only does this present an exciting opportunity to explore the learnability and explainability of often enigmatic architectural forms and morphology to GenAI models, but it also motivates intriguing discussions on the emerging relationships between architects and AI design tools.

Research Question

Building on the foundation of integrating generative AI into architectural workflows, this thesis focuses on generative AI models and 3D building datasets to address a core research question: *how do generative AI models represent spatial relationships, and to what extent can they generate new building envelopes?* This investigation is driven by the desire to understand how these models encode and replicate spatial and morphological relationships fundamental to architectural design.

Goals and Expected Result

Despite the rapid advances in generative AI, a significant disconnect remains between architects and AI researchers. Generative AI models tailored for architectural applications are still in their infancy, with only a handful available—many of which fail to meet the practical needs of architects. As a result, most architects have neither encountered such tools nor integrated them into their workflows. At the same time, architects often lack a deeper understanding of the mechanisms underlying these models, limiting their ability to engage with or influence their development. On the other hand, AI researchers frequently focus on imaginative use cases, overlooking the nuanced requirements and workflows essential to architectural practice. This thesis seeks to bridge that gap by developing and refining a generative AI model grounded in architectural data, fostering a more productive dialogue between the two disciplines and enabling mutual understanding.

The primary goal of this thesis was to develop a generative AI model specifically tailored for vector-based architectural data. This involved designing and training a system capable of producing 3D vector representations—such as meshes, wireframes, and boundary representations (B-reps)—and exploring how such models encode architectural information, including spatial relationships and forms, in ways that align with human design understanding. The aim was to bridge computational and human design thinking, enabling AI systems to capture the nuanced principles that underpin architectural creativity.

Another key objective was to evaluate the feasibility and performance of existing generative AI technologies, such as MeshGPT, in the context of architectural design. This included assessing the ability of these systems to generate high-quality and contextually relevant architectural outputs, as well as identifying the technical requirements for successful training, including dataset availability and preparation.

This thesis also investigated the potential for collaboration between architects and AI tools. By analyzing how AI can serve as a creative partner, the research proposed innovative modes of interaction that foster productive workflows. Ultimately, the findings provide insights into how architects can leverage AI to expand creative possibilities while maintaining control over design outcomes.

With these goals in mind, the outputs of this thesis include:

1. Technical Prototypes:

- A functional generative AI model tailored to an architectural design task, capable of synthesizing coherent and usable 2D or 3D vector data

2. Insights into Architectural AI:

- A deeper understanding of the strengths and limitations of current generative AI models when applied to architectural data.
- Documentation of the steps required to preprocess existing datasets to ensure compatibility with generative models.

Hypothesis

For my thesis, I hypothesize that generative AI can partially recreate the spatial relationships present in building datasets. However, achieving meaningful output requires careful human intervention—particularly in curating and preprocessing the dataset.

To analyze the evolving relationship between human and machine throughout this process, I adopt Kyle Steinfeld’s framework of human-AI interaction in art and design [34]. His theory categorizes human engagement with machine learning (ML) models into three modes: ML as Actor, Material, and Provocateur. This theory forms the theoretical foundation for my research. Steinfeld explores the different ways architects and designers interact with machine learning (ML) models, categorizing these relationships into three types:

- **ML as Actor:** Designers and machines collaborate, emphasizing the machine’s agency and spatial perspective, valuing the process as much as the outcome.
- **ML as Material:** Designers mold the ML model like a craft material—training and manipulating it to imitate specific aesthetic or behavioral traits.
- **ML as Provocateur:** Designers use AI outputs to provoke thought, acting as stimuli rather than final solutions.

In the context of this thesis, all three of Steinfeld’s relational modes were observed. During dataset creation, the role of ML as Material was most evident, as I had to understand and craft the training strategy to optimize the model’s performance. When engaging with the model outputs, ML as Actor emerged, with the system offering insights derived from the data distribution it had learned. Finally, ML as Provocateur became apparent through the model’s limitations—particularly its inability to adapt to site-specific constraints—which prompted critical reflection and revealed the boundaries of current generative capabilities.

Background and Literature Review

The architectural workflow involves a range of software tools, each tailored to specific tasks and stages of the design process, and each supporting distinct file formats.

The architectural design process relies on a range of software tools—such as SketchUp, Rhino, Revit, and AutoCAD—each tailored to specific stages and tasks, and each associated with its own file format. These tools are used across the workflow, from early conceptual modeling to detailed documentation, and they generate 3D data with varying levels of geometric precision and embedded semantic information. For example, Rhino (.3dm files) is frequently used for complex geometry and parametric modeling, particularly during the conceptual and design development stages, often in conjunction with tools like Grasshopper for algorithmic design. Revit (.rvt files) is widely adopted for Building Information Modeling (BIM), supporting detailed, information-rich

3D models that are used in design development, coordination, and documentation stages. As a result, the architectural data landscape is diverse and fragmented, with formats optimized for visualization, parametric design, BIM coordination, or technical drafting, depending on the role of the software in the design process.

In contrast, machine learning workflows for architecture often rely on software-agnostic file formats. Historically, architectural data used in AI workflows was often rasterized into formats such as images, voxels, or point clouds. These formats became prevalent because early AI models, especially convolutional neural networks (CNNs), were optimized for raster data due to their structured grid-like representation. However, these formats often required complex post-processing steps to integrate with vector-based architectural workflows. As a result, there has been a growing demand for AI models that directly handle vector based. These include mesh-based formats like OBJ and STL, which represent geometry as collections of vertices, edges, and faces, and vector-based formats like STEP, which store precise Boundary Representation (B-rep) geometry.

Following this introduction, the discussion will delve into existing examples and literature that highlight how these file formats and software have been applied in generative AI-driven projects. Some precedents are specifically tailored for architectural applications, such as tools for generating floor plans or 3D building models, while others represent general advancements in generative AI that could be adapted to architectural workflows. This review will examine both categories, providing a detailed understanding of the current capabilities and limitations of generative AI in architecture and laying the groundwork for the proposed generative AI model in this thesis. The discussion will begin with floor plan generation, which, while not the primary focus of this thesis, offers valuable insights and lessons that can inform the development of generative models. It will then transition to 3D model generation, emphasizing its alignment with the thesis's goal of advancing generative AI tools tailored to architectural workflows.

2D vector floorplan generation

Since 2020, we have seen huge development in the recent data-based approaches that output vector floorplan [11, 15, 30, 35, 36, 44]. At least two commercial products, Finch [8] and Plan Finder [25] that already launch their product which focus on residential 2d floor plans generation. Such tools usually create a floor plan based on a predefined house boundary and requirements about how many different types of rooms (e.g. living room, bedrooms, bathrooms) are needed and (optionally) how they are connected.

One of the earliest scholarly works in the field is WallPlan [35] which vectorization occurs during

	Raster	Vector	
		Straight	Curved
2D	images	polygons	Polylines, Bézier curve, B-splines, nurbs
3D	voxels	mesh	
	point clouds		

Figure 3: A summary of different data formats used in architecture and AI research.

the search stage in each iteration. Although the model is based on backbones that employ

convolution and take 2D images as input and output, vectorization occurs at each step. In 2022, LayoutEnhancer [15] represents all furniture as vectors and utilizes transformers as its backbone, capable of handling variable input lengths without CNNs. HouseDiffusion [30] was developed in 2023, is a completely vector-based GenAI model. It encodes rooms as vectors and employs a diffusion model for denoising directly on the room vectors as shown in Fig. 4. One notable aspect of HouseDiffusion [30] is that the number of vertices for each room is predetermined before the diffusion process, based on the distribution observed in the training dataset. This approach simplifies the model's task by avoiding hierarchical decision-making about structural complexity, a challenge that diffusion models are less suited to handle. For 3D applications, ensuring that the number and arrangement of vertices are aligned with real-world architectural components could improve the coherence and usability of generated models. It is an approach that can be extended to 3D generative task to help the model focus on refining geometry and typology relation rather than resolving high-level decisions.

In 2018, Yuqian [16] built a topology-based architectural search engine allows users to search for buildings based on similar "spatial relationships" in building typology. The tool was built on a belief propagation algorithm in graph theory on a custom architecture dataset of 100 buildings and she manually extracted a topology graph based on the wall connectivity.

Also in 2018, Cecilia Ferrando's thesis [7] explores the use of machine learning for extracting and analyzing spatial relationships of buildings, particularly through Space Syntax theory. It addresses key technical challenges, such as the extraction of spatial data and the application of machine learning algorithms, focusing on privacy-based predictions. She tried different classical machine learning algorithms to predict the privacy class based on a small dataset of nine architectural plans of houses (total of 97 rooms) from the online architecture archive.

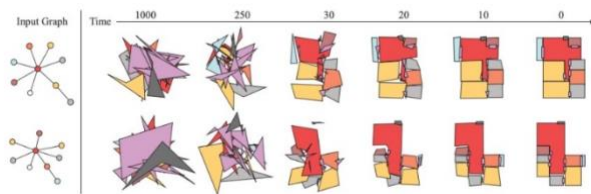


Figure 4: An example of diffusion process on rooms [30]. The number of vertices of each room is fixed before diffusion. The input is a graph representing how different rooms are connected.

3D data generation

Raster Formats

Images: Prior to the development of reliable 3D generative model, one common approach is to slice a 3D model at different height and extract each layer as a 2D image. For example, in the deepir project [6], they slice the skyscrapers at different height.

Voxels: Voxels, the three-dimensional equivalent of pixels, represent values on a regular 3D grid and are frequently used to model volumetric data. Many AI research efforts in architecture utilize voxels to represent the inherently three-dimensional nature of architectural structures [47].

Vector Formats

Point Clouds

Point clouds represent sets of vertices in a three[6]-dimensional coordinate system. These are usually generated through methods like LiDAR scanning and are commonly used in architectural surveys and urban analysis. While point clouds are closer to vector data due to their geometric representation as discrete points defined by spatial coordinates (x, y, z), they lack the explicit structure and connectivity of formats like meshes or B-rep. This unstructured nature, combined with the absence of surfaces or volumes, makes them the 'least vector' data format in some contexts. Moreover, their high-density data representation and lack of inherent order can make them appear raster-like, although they do not follow a fixed-resolution grid. In general, point cloud is not suitable for the task of generating editable architecture models. Nonetheless, there exists research that tries to generate building point clouds directly with diffusion model [39]. One such study used a diffusion model conditioned on a single image, training on relatively small datasets of 321 and 2171-point clouds. The generated point clouds successfully captured the main features of the building depicted in the original image, showcasing potential in AI-driven synthesis of raw architectural forms.

Meshes

In architecture, meshes are used to represent complex surfaces and forms. These meshes are typically composed of vertices, edges, and faces, with triangular meshes being the most common. Triangular meshes are widely used because triangles, as the simplest polygon, can approximate any surface with great flexibility and precision. This makes them particularly useful for representing irregular or curved surfaces in architectural models. Architectural meshes may differ in complexity and detail, depending on the scale and specificity of the design.

Using autoregressive models to generate meshed is a rather recent research direction. In 2023, Siddiqui et al. published MeshGPT [32], which was trained in ShapeNet dataset, and the model can synthesis furniture. It uses an autoregressive model (similar to that of ChatGPT). It outperforms many of the point-cloud or voxels-based models as it can generate clean, coherent, and compact meshes with sharp edges and high fidelity. However, its memory requirement is high for training. Another Diffusion-based method, MeshDiffusion [18] was also proposed in 2023 where they use a diffusion model to deform a tetrahedral meshes. MeshDiffusion first parameterizes mesh into tetrahedral grids and trains a diffusion model on tetrahedral grids and corresponding signed distance functions, based on 2D render supervision. Compared to MeshGPT, its output tends to have less shape edges and corners, and its data preparation is also time-consuming. However, it can output mesh with a higher number of faces.

Multi-modal mesh generation tasks are built on top of previous tasks by allowing certain forms of conditional input from another modality (e.g. text or images). MeshLlama [38] is an example where it is able to unify task generation and mesh generation by representing vertex coordinates and face definition as plain text. The multimodal model demonstrated both potential for mesh understanding and mesh generation. This model has high potential for the architects as it provides the potential of generating building models based on human instruction to potentially increase the efficiency of modeling. However, MeshLlama is not trained on a building specific dataset that currenting the building it generated is not complete.

Wireframe Models

A wireframe model consists of only the edges of a 3D object, represented by lines that form a skeletal outline of the shape. Unlike traditional meshes, which include vertices, edges, and faces, wireframes lack surfaces and can be considered a special type of mesh that has not been triangulated. This simplicity makes wireframes lightweight and ideal for quick previews or early-stage design exploration, where surface details are less critical.

Very recently, a team built on MeshGPT [33] and SolidGen [12] and build an transformer model that generate 3D House Wireframes [19]. Instead of outputting a sequence of encoded meshed, the model chooses to output a sequence of wire to form a wireframe. This choice is a lot more suitable for architecture design. It also outputs the building exteriors and interior room arrangement at the same time, which would have been impossible for MeshGPT [33]. However, it was trained in an augmented version of the 2D RPLAN dataset, with a dummy roof generated based on the room's layout. The capability of this model on a building dataset with more variation remains to be tested.

Boundary Representation (B-rep)

According to BrepGen [43], a Boundary Representation (B-rep) uses interconnected faces, edges, and vertices to define a solid's geometry. Faces represent regions of parametric surfaces bounded by closed loops of edges, while edges correspond to segments of parametric curves trimmed by terminal vertices. By maintaining the adjacency information between these elements, a B-rep provides a detailed and complete representation of the solid's shape. This structured representation is critical in architectural workflows, particularly in CAD tools, where precision and editability are paramount. B-rep models also excel at describing curved surfaces and curved boundaries efficiently, often utilizing parametric surfaces like NURBS to capture intricate and complex geometries with high accuracy.

While B-rep excels at representing curved surfaces and boundaries efficiently, research around its use in AI, particularly in generative design and modeling, is still nascent compared to other data formats like voxels and meshes. However, recent advancements have begun addressing these limitations. SolidGen [12], for example, uses transformers to synthesize B-rep directly by encoding geometry and topology in a sequential manner. BrepGen [43] a diffusion-based model, further extends this capability by generating B-rep structures with less training data, which is particularly advantageous given the limited availability of architectural datasets. These models highlight the potential of B-rep as a robust format for AI-driven architectural design, offering both precision and compatibility with industry-standard workflows. Moreover, diffusion-based methods' lower data requirements make them particularly well-suited for advancing AI applications in architecture.

Other 3D Representations

Neural Radiance Fields (NeRF) [21] and Gaussian Splatting [14] are advanced 3D representation techniques that leverage multiple images from different viewpoints to create detailed 3D models of scenes. NeRF, which uses a neural network to predict color and density, excels at generating photorealistic views for intricate reconstructions. Gaussian Splatting, on the other hand, represents a scene by storing millions of 3D Gaussian blobs, offering a more structured representation. Both methods are particularly valuable for tasks like novel view synthesis, where they generate new perspectives of a scene.

In 2022, Michael Hasey [10] curated a dataset 3-D meshes of 313 wooden churches located within and near the Carpathian Mountain regions of Ukraine using photographic data and 3-D

reconstruction techniques in his MSCD thesis. He trained a Variational Autoencoder so that he can extract embedding from the latent space of the implicit NerS representation. He then ran clustering algorithm in the embedding space and compared the clustering results of that made by humans based on common church characteristics and the machine learning result aid in identifying and analyzing the anomaly. While this task is more analytical than generative, variational autoencoder can be used to generate new data too.

However, despite their ability to produce realistic images, they lack a true understanding of the underlying 3D structure. Converting such volumetric representations into discrete meshes is as, if not more, complex than working with voxels or point clouds. While research like [9] has attempted to generate these representations directly from images produced by text-to-image models, the computational intensity and the resultant inaccuracy involved makes these approaches difficult to integrate directly into CAD tools, limiting their practical application in architectural workflows.

Available 3D Datasets

Training a GenAI model notoriously often requires a huge amount of data. For example, the famous text-to-image model stable diffusion [28] is trained from 400 million image-text pairs. Architecture data is often closed source, and the limited number of open-source datasets constrained the potential task, and model can be trained for. They also must be pre-processed into a format is ingestible by the generative AI model, which in general cannot deal with too many mesh faces.

The current available 3D Building dataset contains data of different scales, some examples including individual residential units, buildings or even on an urban scale. Fig 5 shows a list of publicly available datasets [17, 29, 45], [41]. One such dataset is RPLAN [41], which is a comprehensive collection of over 80,000 two-dimensional floor plans. Another important dataset is Structure3D [45], a 2.5D dataset comprising Computer-Aided Design (CAD) models of more than 3,500 rooms. BuildingNet [29] is a dataset specifically focused on 3D building meshes, where each model is segmented and labeled to provide detailed information about various building components. The models are sourced from Trimble Warehouse and many of them have high number of faces (on average >20k faces per model). UrbanScene3D [17] contains 16 scenes that include both real and synthetic urban environments. The mixture of real-world and synthetic data makes this dataset particularly useful for developing and validating models that need to operate in diverse and dynamic urban settings. Building3D [37] dataset contains more than 160k buildings from 16 cities in Estonia, built from aerial LiDAR point clouds. The models in Building3D are relatively simple with an average of 50 faces for each model. It also simplifies meshes into wireframe model which requires smaller storage space. 3DBAG [1] is another publicly available dataset contains more than 10 million buildings in the Netherlands built from satellite image and Lidar scan. The dataset is of LOD2, which means that it does include slanted roof, however it does not capture architecture details such as position of the door and windows. The models are generally considered simple with 80% of the models having less than 50 faces.

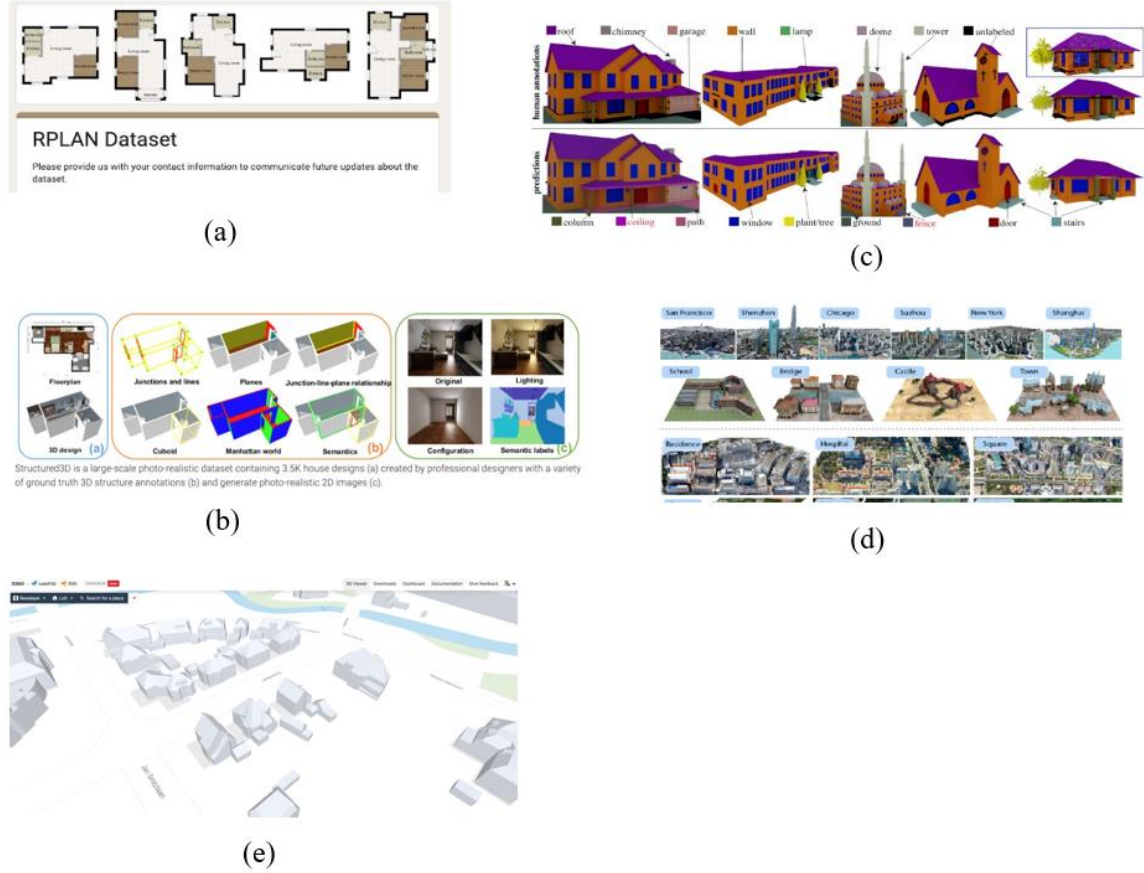


Figure 5: 5 vector-based building datasets. (a) RPLAN is a collection of 80000+ 2d floorplans. [41] (b) Structure3D is 2.5D dataset consisting of CAD models of 3500+ rooms. [40] (c) BuildingNet contained segmented and labeled 3D building meshes. [29] (d) UrbanScence3D contains 16 real/synthetic scenes. [17]

Methodology

In this section, I will document how do I arrive at the final experiment setup to evaluate my hypothesis and the steps involved to prepare the dataset. I decided to frame the problem as a text-to-3D task where I use text description to generate houses of different roof styles and compositions. Overall, like many machines learning research, it involves selecting suitable datasets and comparing multiple baseline generative AI models. However, operating in a unique field of architecture also poses unique challenges due to the lack of open-source text-to-3D architecture dataset. This section covers the details of preparing the dataset and culminates in the training of machine learning models.

Precedent: House Gable Project

One of the main inspiration and architecture comparative of this thesis is the **House Gable Project** by Jennifer Bonner [48], which includes an analysis of all single-family buildings in Atlanta, this research takes a similar approach to studying roof typologies, leveraging computational tools to analyze and generate roof designs. Bonner broadly categorizes roofs into types such as **shed**, **dormer**, **gable**, **mansard**, **hip**, and **butterfly**, emphasizing the geometric variety and cultural significance of each typology. The project is at least meaningful in two aspects: the team use it to identify the relation between roof typologies and social economic factors. It also eventually leads to a single-family residence design that incorporates a hybrid of roof forms, which demonstrates its deep root in the community rich history.

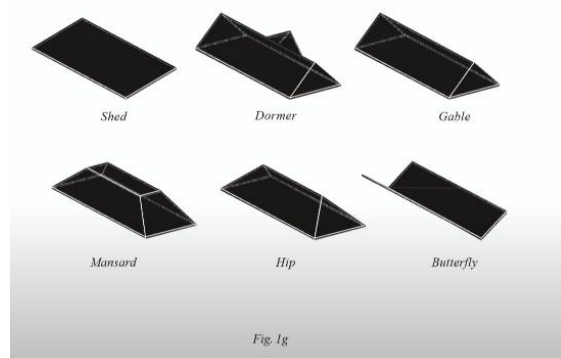


Figure 3 Different type of roofs in Atlanta, Georgia

The 3DBAG dataset offers an opportunity to conduct similar roof research in Netherland. Similarly, many single-family homes exist in the Netherlands feature **gable roofs**, reflecting both functional and regional design characteristics. We can use **3DBAG dataset**, a rich resource of Dutch building geometries, to train generative AI models that explore roof typologies.

While the long-term goal is to compare four generative models—**LLaMA-Mesh**, **BrepGen**, **3DWire**, and **MeshGPT**, as introduced in the Background section—this thesis focuses on MeshGPT due to time constraints. The intention is to create a reusable dataset and evaluation framework, enabling future experiments with the remaining models. A comparative overview of these models is included later in this section for reference.

The experiment is framed as a **conditional generation task**. Ideally, conditioning inputs would include not only text descriptions but also features like building footprint, footprint area, height, and street orientation. However, due to time limitations, I focus solely on **text-to-mesh generation**.

Comparison of different 3D vector generative models

The following table provide a basic comparison between the four potential models that can be used for the test-to-3d task outlined in the previous section.

Table 1 Compare four generative models: MeshGPT, 3DWire, LlamaMesh, BrepGen

Title	MeshGPT	3DWireframe (https://vcc.tech/research/2024/3DWire)	LlamaMesh	BrepGen
Encoder Architecture	We first learn a vocabulary of latent quantized embeddings, using graph convolutions, which inform these embeddings of the local mesh geometry and topology.	SEGMENT AS NODE 1) Coordinates of its endpoints, 2) Length of the segment 3) Directional orientation 4) Angle between it and adjacent segments 5) Coordinates of its midpoint	No special encoder needed quantize to 64 bin per axis, use as normal vocabularies	Two VAEs to compress the shape feature Fs, Es into lower-dimensional latent Fz, Ez
Encoder Training loss	Reconstruction Loss	Reconstruction Loss	N.A.	Reconstruction Loss
Generative Architecture	Auto-regressive (GPT2-medium model, context window of up to 4608 embeddings)	Auto-regressive	Llama-3	Latent Diffusion Module
Training loss	CE loss	CE loss	CE loss	L2-norm regression loss

Metrics reported	Geometric Fidelity by sampling Point Clouds on generated mesh Minimum Matching Distance (MMD ↓), Coverage (COV ↑), and 1-Nearest-Neighbor Accuracy (1-NNA, optimal@50%). Visual Similarity Render to images from multi views and calculate the Fréchet Inception Distance (FID ↓) and Kernel Inception Distance (KID↓) scores for these image sets. Compactness as the average number of vertices and faces in the generated meshes.	Geometric Fidelity same as MeshGPT Vertex-line segment connectivity - Two-Line-Connected Vertex Proportion (2L-CVP) and - Three-Line-Connected Vertex Proportion (3L-CVP) KL Divergence	Not reported	Distribution Metrics randomly sampled 3,000 B-reps from the generated data and 1,000 B-reps from the reference test set, and then sample 2,000 points from the solid surface and measure • Coverage (COV) is the percentage of reference data with at least one match after assigning every generated data to its closest neighbor in the reference set based on Chamfer Distance (CD). • Minimum Matching Distance (MMD) is the averaged CD between a reference set data and its nearest neighbor in the generated data. • Jensen-Shannon Divergence (JSD) measures the distribution distance between reference and generated data after converting point clouds into 283 discrete
------------------	---	---	--------------	---

				voxels. CAD Metrics Use the same 3,000 B-reps • Novel percentage of data that do not appear in training set. • Unique percentage of data that appears only once in generation. • Valid percentage of B-rep data that are watertight solids.
Data format	Triangulated Mesh, obj files	Wireframe (edge only)	Triangulated Mesh, obj files	.step files
Dataset Used	ShapeNetV2 dataset: 28980 shapes with <800 faces post-decimation	RPLAN dataset: <78000 architecture wireframe data	Objaverse dataset: 31k <500 faces mesh, augmented by rotation	DeepCAD dataset & Furniture b-rep dataset & ABC dataset: 259,597 ABC B-reps and 1,198 furniture B-reps with a maximum of 50 faces and 30 edges per face are used for training
Original Training time needed	2 A100 GPUs for ~ 2 days	encoder decoder: 8 RTX 3090 GPUs for ~ 2 days transformer: 5 days on 8 RTX 3090 GPUs	32 A100 GPUs for 21k iterations for 2400 gpu hours (~3 days)	4 NVIDIA RTX A5000 GPUs
Open source	Official weight available	Not available	Official weight available	Official weight available
Control ability (in the paper)	Categorical control	N.A.	Natural language control	Categorical control

Data Preprocessing

In this section, I document the dataset preprocessing part, which involves filtering out the relevant model in the full 3DBAG dataset, optimize the mesh and convert them into desirable format as well as getting the captions using GPT-4o models.

Prepare the 3D models.

Filtering for Residential Buildings

While the 3DBAG dataset does not directly provide information on building use cases, it is built on the **Basisregistratie Adressen en Gebouwen (BAG)** dataset [13], which includes this information using the same building identification numbers. The BAG dataset can be [downloaded](#) in GeoPackage (.gpkg) format, which can be accessed in Python using geopandas or SQLite as a Pandas DataFrame. This allows for efficient filtering operations. Since our focus is on residential buildings, we filter the dataset based on the building use case ('gebruiksdoel'), selecting only those classified as residential ('woonfunctie'). This reduces the usable 3d models from 11166642 to 5457082. In reality, due to the limitation of budget and time, we use 30k models for Autoencoder training and 10k captioned mesh for transformer training.

Planar Decimation using Blender

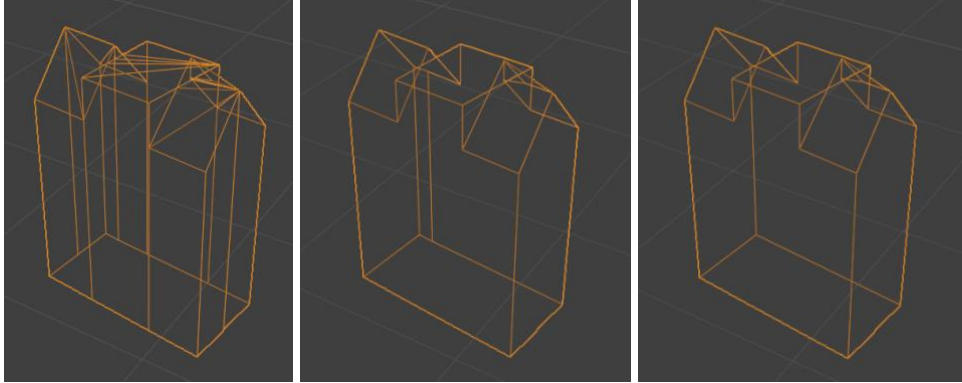
In the original 3DBAG dataset, there are no measures in place to ensure that roof faces are planar. Additionally, neighboring faces that are almost co-planar are not merged into single surfaces, which creates inconsistencies in the data. To address this issue, the primary goal of the cleaning process is to:

- Merge neighboring "almost co-planar" faces into a single face, based on a threshold angle. and
- Ensure all faces are strictly planar.

While there exist multiple face decimation tools and software, the **Decimate-Planar** function in Blender has been identified as a suitable tool for addressing these challenges. One competitor would be the Autodesk Fusion 360 software. However, it does not support scripting which does not meet our requirements of processing thousands of mesh. Specifically, **Decimate-Planar** simplifies meshes by merging faces that fall within a specified angular threshold, thus reducing non-coplanar inconsistencies while retaining the overall geometry.

The following images illustrate the outcomes of the cleaning process:

- Original Model: Displays irregularities, including non-coplanar faces and unmerged neighboring faces.
- Cleaned Model (1° Threshold): Slightly simplified with minimal loss of detail. Suitable for high-fidelity applications.
- Cleaned Model (5° Threshold): More aggressive simplification, reducing complexity while maintaining architectural integrity. **<chosen>**



At the same time, decimate function create non-planar faces as it only merge almost coplanar faces into one face with making sure it is planar. To address this, Blender offers the "Make Planar Faces" tool, which iteratively adjusts the vertices of selected faces to achieve planarity. To use this feature, enter Edit Mode, select all the faces, then navigate to the Mesh menu, choose 'Clean up,' and select 'Make Planar Faces.' This tool provides two key parameters:

- Factor: This determines the distance each vertex moves toward the target plane in each iteration. A higher factor moves vertices more per iteration, leading to faster convergence but potentially overshooting the desired position.
- Iterations: This sets the number of times the operation repeats. More iterations allow for finer adjustments, enhancing the accuracy of the planarization process.

Adjusting these parameters helps in effectively flattening faces. Note that since some vertices are shared by multiple faces, it is possible the some faces will never be planar. We can use blender tools like "Split Non-Planar Faces" to divide complex faces into smaller, planar ones, thereby minimizing distortion.

Geometry Filtering based on shape complexity

We can use python to filter out geometry that is too simple (number of faces present in the mesh < 8) and too complex (> 50 faces).

Converting to .step for BrepGen

While MeshGPT operates using the .obj file format, which represents 3D geometry as a collection of vertices and faces (typically triangular), another model of interest—BrepGen—requires input data in the .STEP file format. The STEP (Standard for the Exchange of Product model data, ISO 10303) format is a neutral data exchange format that describes 3D objects in a way that preserves high-level geometry and topology. Unlike .obj files which can only describe triangular faces These include detailed definitions of solid geometry using mathematical descriptions of surfaces, curves, and edges, making them particularly suitable for CAD and engineering applications where design precision and semantic richness are critical.

To convert our decimated .obj meshes—those reduced in complexity to ensure tractability in generative modeling—into the B-rep format, we use Rhino's MeshToNurb tool. This utility approximates a mesh surface with planar faces and converts it into a set of NURBS (Non-Uniform Rational B-Splines) surfaces, which can then be saved as .STEP files. This conversion is necessary

to interface with BrepGen, which operates directly on solid geometry rather than surface approximations. The pipeline, therefore, involves first simplifying the mesh, ensuring it consists primarily of planar elements, and then using Rhino to generate the corresponding NURBS representation that retains as much geometric fidelity as possible during the translation process.

Caption Generation

To enable language-conditioned geometry generation, we generate descriptive captions for each building model. This process begins with the creation of eight rendered views per model, strategically captured from different angles to highlight key geometric characteristics. These viewpoints are uniformly distributed to ensure comprehensive visual coverage of the building's volumetric and surface features.

For the rendering task, we employ PyTorch3D [26], a flexible and GPU-accelerated rendering library developed by Facebook AI Research. PyTorch3D allows for fast, batched rendering of 3D models directly within the PyTorch ecosystem, which is particularly advantageous when dealing with large-scale datasets. In our case, we need to render approximately 30,000 building models, each with eight views—amounting to 240,000 renders. Using more traditional rendering tools such as Blender or Rhino would be prohibitively slow and resource-intensive due to their emphasis on realism and user interface interactivity, which are not essential for our downstream task of caption generation.

These rendered images are then fed into a language model (e.g., ChatGPT-4o) for caption generation. We apply in-context learning [2], seeding the model with example caption structures to standardize output. The resulting captions encapsulate salient design features, architectural elements, and contextual cues, effectively linking visual geometry with textual semantics. This forms the basis for language-conditional generation, where geometry synthesis can be guided or "steered" using natural language prompts. Cost-wise, the input contains $85 \times 8 = 680$ tokens for images and 300 tokens for text so total of ~ 1000 input tokens, the output tokens are estimated to be around 100 tokens. The table below gives a break down on the cost for each model.

Table 2 Compare the cost of GPT-4o and GPT-4o-mini

	Single input call	Single output	Single call	10 k call
GPT-4o	0.0037	0.0018	0.0055	55 usd
GPT-4o mini	0.0003	0.00006	0.0003	3 usd

After a few trials runs, I decided to proceed with **GPT-4o** as the **GPT-4o-mini** model fails to recognize that all eight renders came from the same 3D models and often give eight different captions rather than one merged caption despite different ways of prompting. The final text prompt template is attached as below.

You are an architectural assistant describing 3D building models based on multi-angle renders. Your goal is to generate concise, informative captions for each model that highlight architectural form, geometry, and roof types. Avoid guessing the function or internal use of the building.

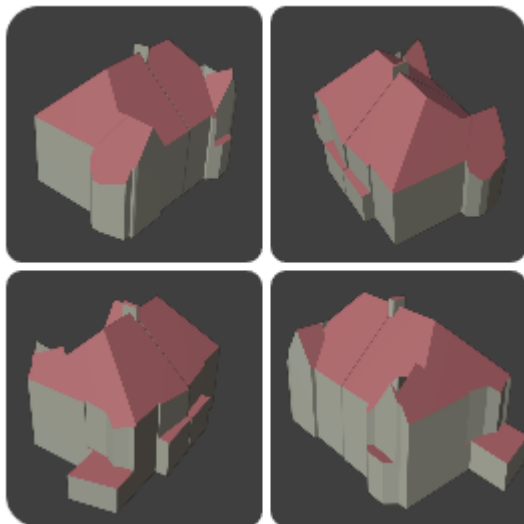
Guidelines:

- Focus on exterior shape and spatial organization.
- Describe roof types precisely (e.g., gable, hip, flat, shed, gambrel, cross-gable, multi-gable).
- If applicable, note spatial arrangements (e.g., "L-shaped layout", "T-shaped plan"), but only when the layout is clearly distinguishable.
- Note contrasts in height, massing, or roof style across building sections.
- Mention elements like dormer-like projections, chimneys, or extensions if clearly visible.
- Do not include any reference to building color or texture.
- Do not infer purpose or use (e.g., do not say "residential" or "barn").
- Use clear, neutral language suited for input to a text-to-mesh model.

Example captions:

- "A compact structure with a tall gabled roof and a flat-roofed side extension."
- "An elongated volume with two intersecting gabled roofs forming an L-shape."
- "A main mass with a steep gable roof and a lower perpendicular section featuring a hip roof."

Now, I will pass you eight images of a building. Please describe the building form in a similar style.



<more views omitted here>

Model Architecture Design

One innovation and a challenge to the thesis is how to effectively include text conditioning in the model. Other than Llama-Mesh which natively supports mesh, other model needs special conditioning mechanism. Here we study how MeshGPT achieve conditional generation based on text information. Other than MeshGPT, although I do not have time to finetune the BrepGen Model, I included the architecture design for BrepGen text conditioning in the appendix.

MeshGPT

MeshGPT extends a generative mesh pipeline with semantic language cues, allowing it to produce 3D geometry guided by a text prompt. It couples a transformer-based mesh generator with a text encoder so that the mesh generation process can “understand” a given description. In practice, MeshGPT first condenses the input text into a single pooled text embedding that captures the prompt’s meaning. In my thesis, an open-source text embedding model bge [5] is used. This text embedding then guides the transformer as it sequentially outputs tokens for the mesh. The model uses a cross-attention mechanism to inject the text information: at each decoding step the mesh tokens attend to **the fixed text embedding**, aligning the geometry with semantic cues from the prompt.

Additionally, MeshGPT incorporates Feature-wise Linear Modulation (FiLM) layers – which apply learned scale-and-shift transformations to features – so that text context can directly modulate intermediate features of the mesh generator. A specialized gating module called the GateLoopLayer further ensures the text guidance is consistently integrated across the sequence. This GateLoop mechanism recurrently blends information (including the text context) as the sequence is generated, helping maintain coherence from the first token (when little geometry context exists) through to the final mesh.

Evaluation Method

The model is evaluated based on how well it can generate 3D meshes that match given textual descriptions. To capture a broad spectrum of difficulty, I designed a benchmark consisting of 30 prompts: 5 categorized as easy, 15 as medium, and 10 as hard. In the results section, I report both the quality of the generated captions and the alignment between text and geometry. To assess the generalizability of the generative pipeline, I also rewrote each caption using paraphrased wording to test whether the model could still produce coherent and meaningful outputs.

Results and Analysis

Overall, I selected a set of 30 examples to evaluate both the quality of the generated captions and the conditional generation of 3D meshes. This set includes 5 simple captions, 15 of medium complexity, and 10 difficult ones. The difficulty level is determined based on the frequency of the caption patterns in the dataset and the semantic complexity of the descriptions. For instance, captions that specify spatial relationships between multiple interacting volumes are generally categorized as “difficult.”

Model Caption

To streamline evaluation, I developed a lightweight web tool that enables efficient browsing of the entire dataset and inspection of captions generated by the GPT-4o model. As shown in the image below, the interface features a rotatable 3D viewer on the left (powered by Three.js). On the right, it shows the generated captions (including outputs from one or more models) on the top, and rating controls in the middle section; at the bottom, multiple rendered views of the mesh provide visual references for evaluation.

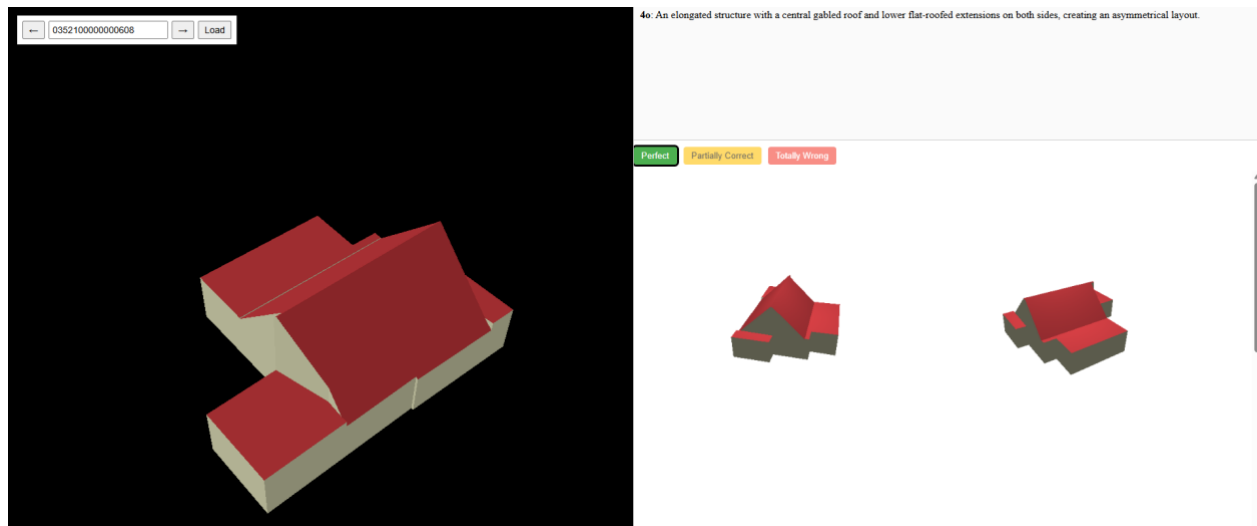
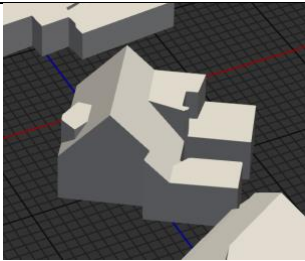
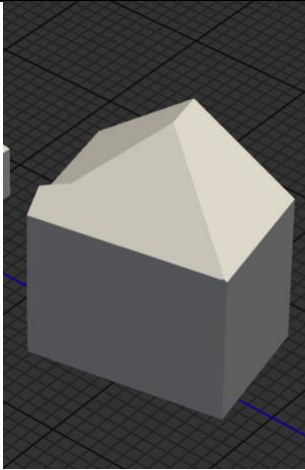
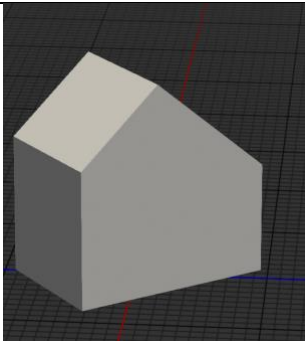


Figure 4 The caption evaluation UI. In this case, the generated prompt is "an elongated structure with a central gabled roof and lower flat-roofed extensions on both sides, creating an asymmetrical layout"

An initial review of the first 30 captions indicates that approximately 70% are accurate, 20% are partially accurate, and 10% are completely inaccurate.

Table 3 Examples of caption quality

A visualization of the 3D mesh	Caption generated	Caption Quality
	A compact structure with a prominent gabled roof featuring a small dormer at the front. At the rear, there is a stepped-down series of flat roofs.	Accurate
	A polygonal structure with an irregular roof featuring a central ridge, where one side slopes inward to form a distinct flattened peak	Partially Correct (The issue with this model is that this is supposed to be a part of a hipped roof but it was identified as two separate buildings in the BAG dataset.)
	A simple structure with a single sloping shed roof.	Completely Inaccurate

Keywords Analysis

To examine the nature of the dataset based on the caption generated for 3DBAG dataset, we defined a set of **25 domain-specific keywords** and observe their distribution. These keywords were selected based on two key criteria:

1. **Domain relevance** — terms that frequently describe architectural geometry and composition (e.g., *gabled*, *flat*, *dormer*).

2. **Descriptive power** — the ability of a term to imply structural change or complexity (e.g., *extension, intersecting, multi-volume*).

The list included roof types (e.g., *gabled, hipped, shed, gambrel, mansard, butterfly*), shape descriptors (*rectangular, polygonal, L-shaped*), and modifiers or features (*dormer, notch, extension, stepped*). Each caption was scanned for all morphological variants of these keywords (e.g., *sloped, slope, sloping* all mapped to *sloped*) as shown below.

```
keyword_map = {
  "flat": ["flat"],
  "gabled": ["gabled", "gable"],
  "hipped": ["hipped", "hip"],
  "shed": ["shed", "shedroof", "shed-roof"],
  "gambrel": ["gambrel"],
  "mansard": ["mansard"],
  "butterfly": ["butterfly"],
  "pyramidal": ["pyramidal"],
  "dormer": ["dormer"],
  "extension": ["extension", "extend", "extended", "extending"],
  "protruding": ["protruding", "protrude", "projection", "projecting"],
  "notch": ["notch", "notched"],
  "asymmetrical": ["asymmetrical", "asymmetric"],
  "intersecting": ["intersecting", "intersect", "intersection"],
  "interlocking": ["interlocking", "interlock"],
  "stepped": ["stepped", "step"],
  "L-shaped": ["L-shaped", "L shape", "L shaped"],
  "rectangular": ["rectangular", "rectangle"],
  "polygonal": ["polygonal", "polygon"],
  "irregular": ["irregular"],
  "multi-volume": ["multi-volume", "multiple volumes", "multi volume"],
  "tiered": ["tiered", "tier"],
  "connected": ["connected", "connect", "connecting", "connection"],
  "compact": ["compact"],
  "sloped": ["sloped", "slope", "sloping"]
}
```

We then computed the frequency of each keyword across the dataset, which is shown in Figure 6. The most frequent labels were:

- **gabled** — 6,831 occurrences
- **flat** — 6,250 occurrences
- **extension** — 4,266 occurrences

These suggest that gabled and flat roofs dominate the dataset's geometric language, along with structural additions (extensions). On the other hand, the **least frequent keywords** were:

- **butterfly, gambrel, and mansard** — each with fewer than 10 mentions

This rarity implies these roof types are either visually rare in the training set or under-described linguistically — which limit the model’s generalization to uncommon typologies.

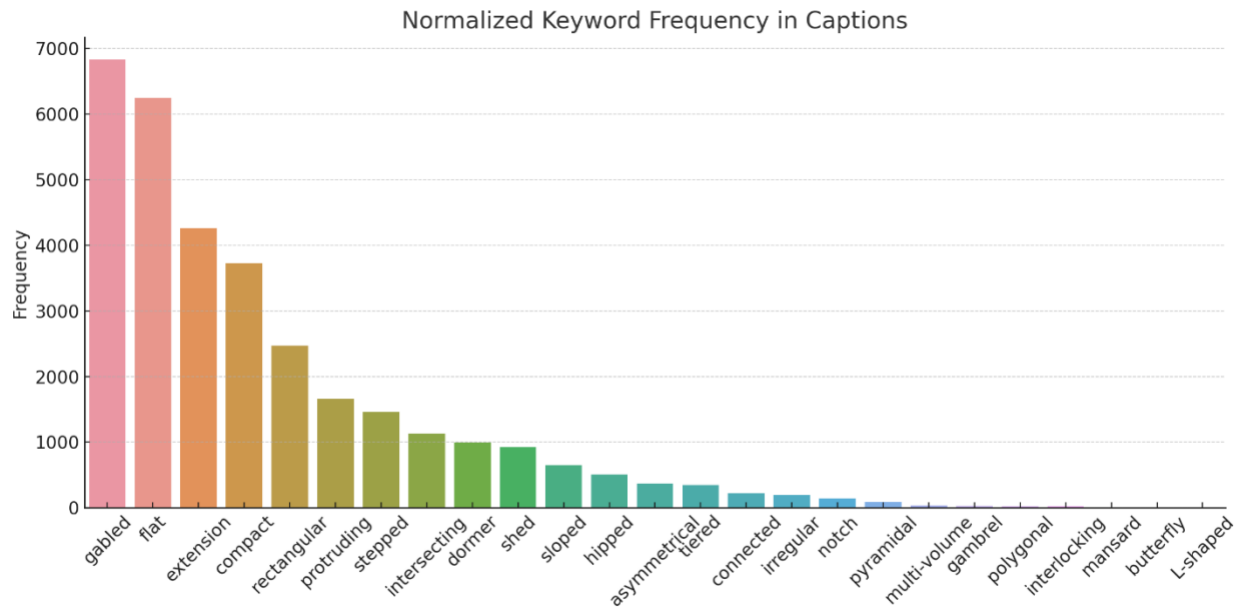


Figure 5 Normalized keywords frequency diagram. Gabled roof is most common in our dataset.

To better understand how different architectural features are described together, we also computed a **co-occurrence matrix** (Figure 7) — a 25×25 symmetric matrix where each cell (i, j) represents how many captions contained both keyword *i* and keyword *j*.

From this, several meaningful associations emerged:

- **flat** co-occurs most with **extension**, **stepped**, and **notch** — indicating that flat-roofed structures are frequently described with additive or subtractive volume modifications.
- **gabled** co-occurs often with **dormer**, **connected**, and **compact**, which are hallmarks of traditional, residential-style forms.
- More complex structures involving **intersecting**, **multi-volume**, and **tiered** showed sparse co-occurrence, consistent with their architectural rarity.

These patterns not only reflect trends in the dataset but also suggest where generative models might overfit (to common pairings) or underperform (on rare combinations).

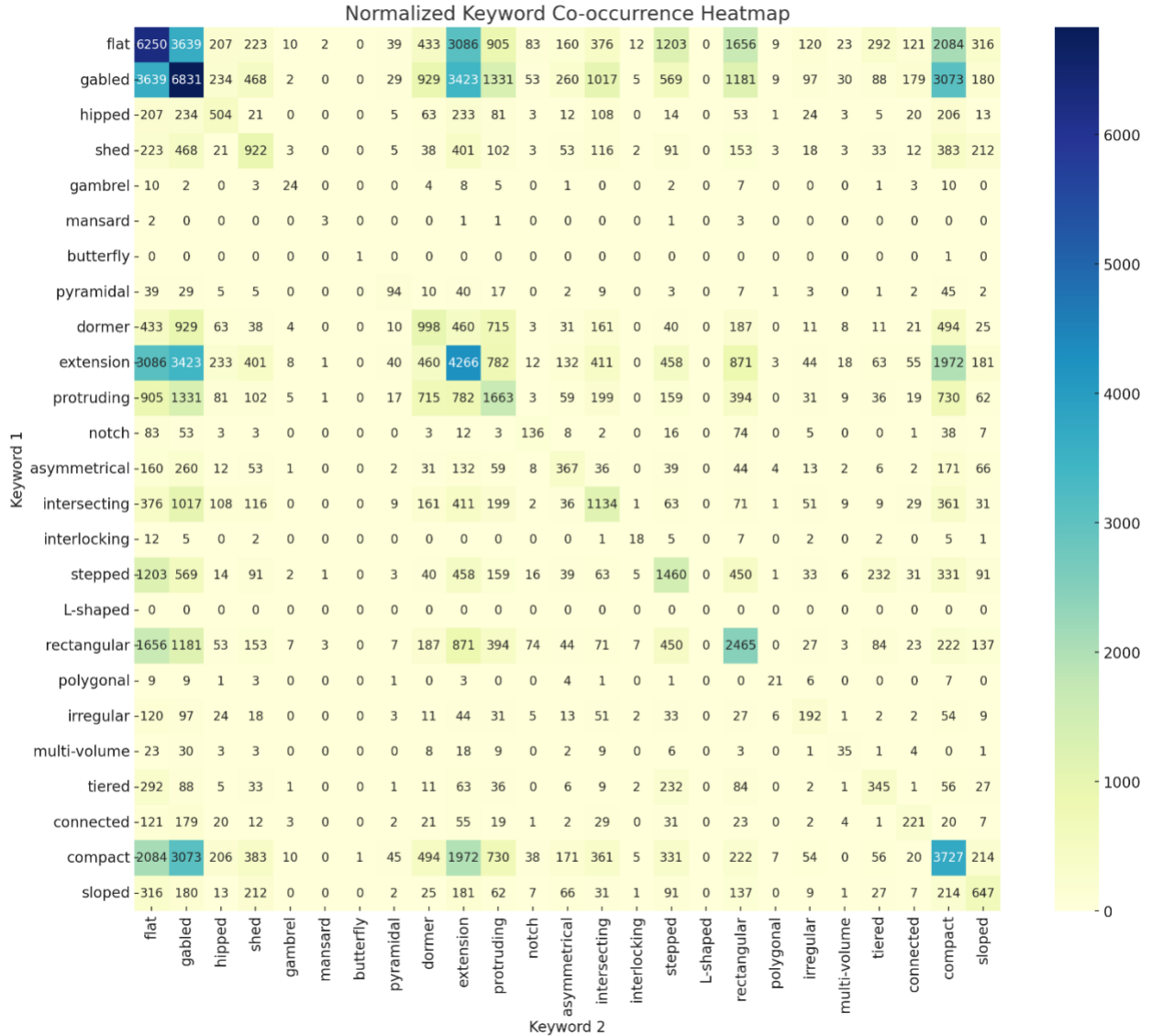


Figure 6 Normalized co-occurrence matrix for keywords

Embedding model analysis: BGE and Embedding Space Exploration

Considering the text embedding is calculated by condensing the input text into a single pooled embedding vector, we perform both embedding space visualization and semantic retrieval tests to evaluate whether captions with similar architectural descriptions are indeed mapped closely together. To explore this, we visualized the 2D projection of sentence embeddings generated from the architectural captions using UMAP [20], a non-linear dimensionality reduction technique suitable for preserving local neighborhood structure. In these plots, points are colored based on the presence of specific architectural keywords or structural patterns.

As shown in Figure 8 below, captions containing the word "dormer" and "L-shaped" exhibit a distinct local clustering (red points), indicating that the model consistently encodes this structural

feature into a semantically coherent region of the embedding space. We can also identify smaller cluster for keywords like "rectangular" and "cube".

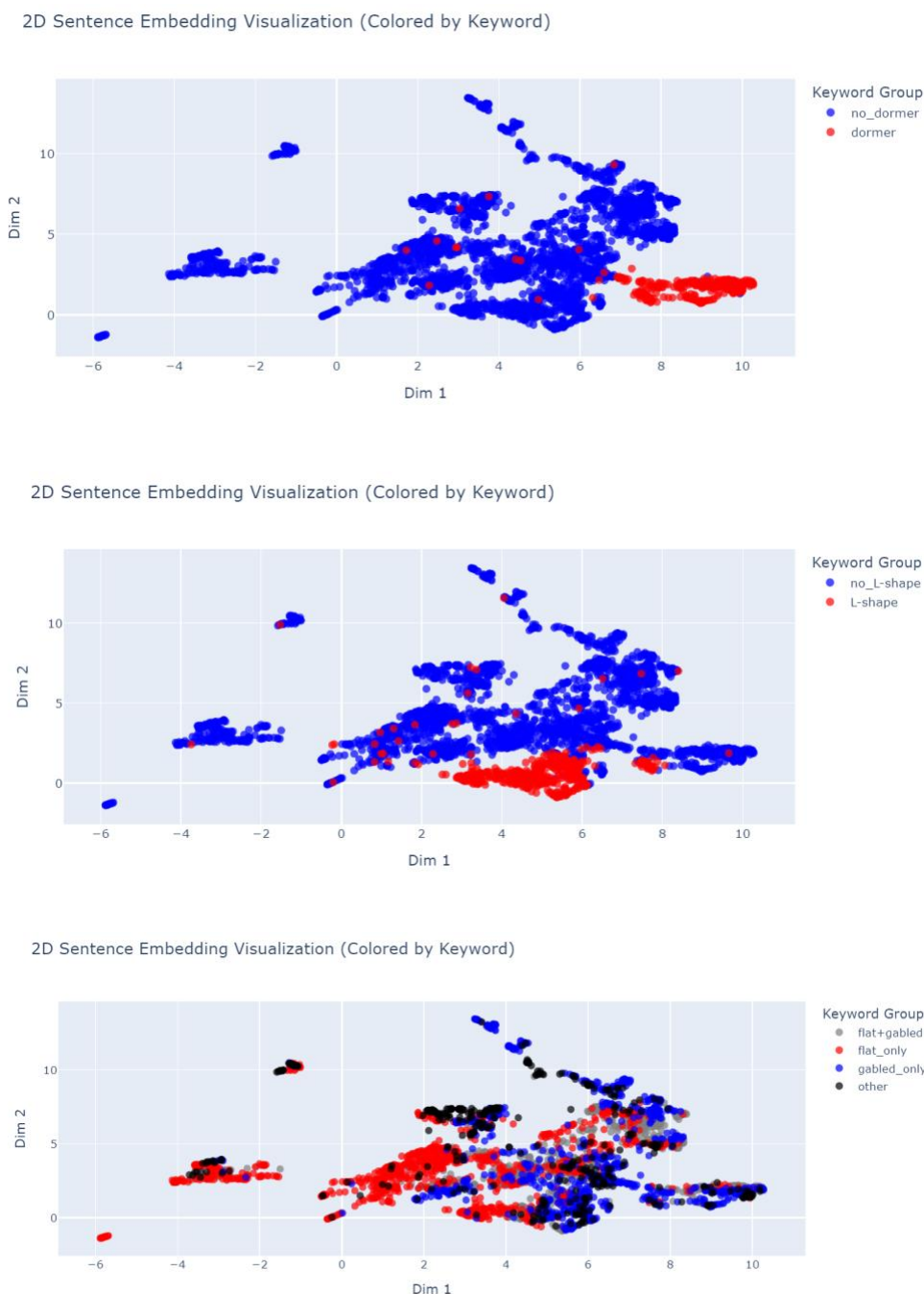


Figure 7 Embedding space analysis of embedding generated by bge model

We further investigated the sentence embedding space by examining how captions distribute according to co-occurring architectural features, such as roof types. We explored the spatial

clustering of captions that mention “flat”, “gabled”, or both. “flat” and “gabled” are the two most frequent keywords and they often co-occur. As shown in Figure 8, captions that include only flat or only gabled descriptors exhibit some clustering tendencies, but they are not as cleanly separable as other features like dormer or L-shape. This could explain why the MeshGPT model do not always scricltly generate flat roof, highlighting the limitation of the approach.

MeshGPT

Autoencoder Training

We train the model Autoencoder and the Transformer on RTX-3080-ti machine. When we are training the model, we use a learning rate of 0.002 until the loss reaches 0.45 and then trained with the model manages to converge to below 0.25 on training dataset.

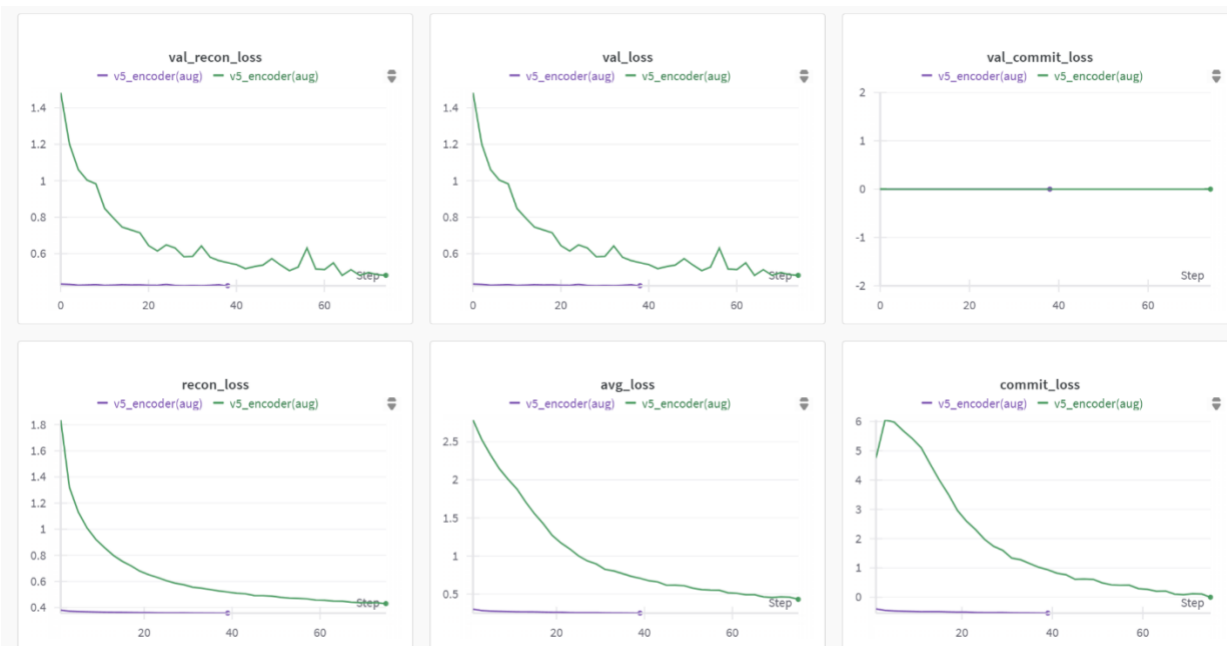


Figure 8 Training loss for MeshGPT Autoencoder

Overall, the autoencoder trains well and generalize well into the test set. The model is able successfully reconstruct the meshes from the face encoding. We observe better test set performance generalizability after we add more jitter on the vertices of the training set.

Transformer Training

After the autoencoder is trained, the transformer is trained with a training loss of 0.01 until the training loss reaches 0.5 before reducing the training rate to 0.001. A batch size of 2 is used (max possible given the 12 GB memory capacity of 3080-ti). Based on the training graph, we observed that the model is clearly overfitted on the training dataset. There are multiple ways to explain this case. First, the language conditioning could clearly be too weak, given there is one to many mapping in the case of a caption to a model.

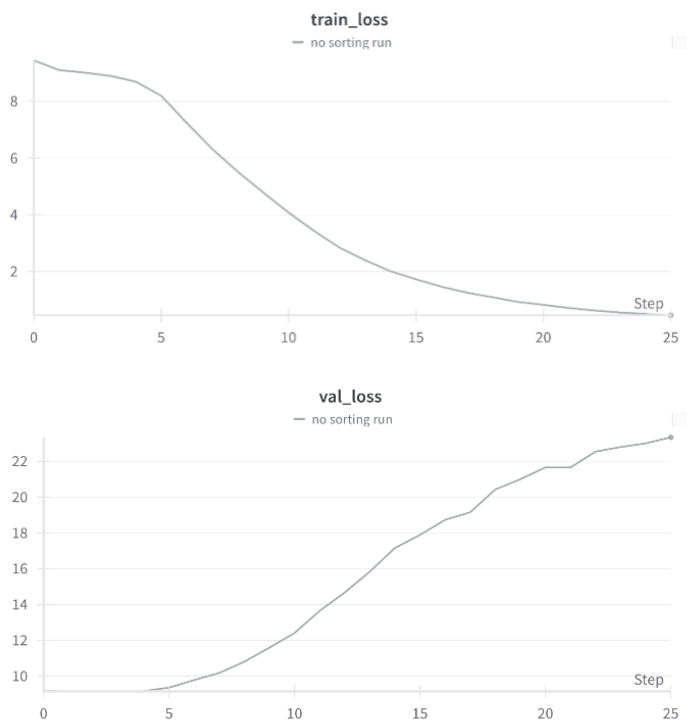


Figure 9 Training loss for MeshGPT transformer

As a next step, some way to reduce overfitting include better augmentation techniques including rewrite the text condition such that the same mesh is matched to multiple captions to reduce overfitting.

Trends, Success and Failures

The following tables reports the text-shape correspondence between the caption and the generated mesh, categorized by different difficulty levels.

Table 4 Accuracy count of text-mesh consistency

Difficulty	T=0		T=0.5		T=0.95		GT Accuracy
	Correct	Decoding Failure	Correct	Decoding Failure	Correct	Decoding Failure	
Simple	20/50	10/50	18/50	11/50	16/50	11/50	2/5
Simple (Rephrased)	30/50	0/50	24/50	1/50	19/50	3/50	N.A.
Medium	100/150	10/150	105/150	6/150	98/150	13/150	10/15
Medium (Rephrased)	60/150	0/150	53/150	11/150	15/150	33/150	N.A.
Difficult	70/100	0/100	70/100	2/100	69/100	6/100	8/10
Difficult (Rephrased)	30/100	10/100	28/100	13/100	28/100	15/100	N. A.

Table 5 Accuracy rate of text-mesh consistency.

Difficulty	T=0	T=0.5	T=0.95	Caption Accuracy
Simple	0.4	0.36	0.32	0.4
Simple (Rephrased)	0.6	0.48	0.38	NA
Medium	0.67	0.7	0.11	0.667
Medium (Rephrased)	0.4	0.353	0.1	NA
Difficult	0.7	0.7	0.69	0.8
Difficult (Rephrased)	0.3	0.28	0.28	NA
Total	0.51	0.49	0.41	0.67

The main observation is due to overfitting of the transformer backbone, it is unable to generate any innovative mesh. Instead, what seems to be doing is to retrieve shapes from the existing trainset. However, in the case their multiple house models match the same captions. Sampling ($t > 0$) will retrieve the different models. An example of such a case is attached below.

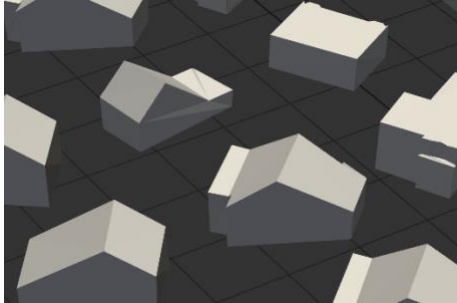
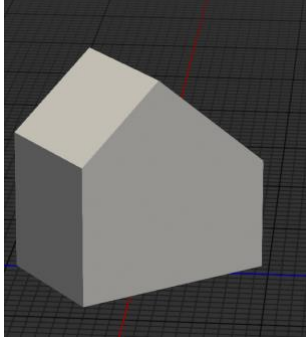
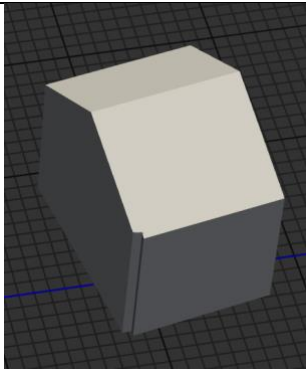
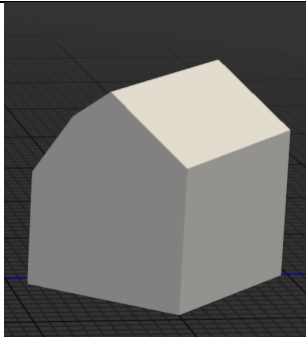


Figure 10 Two meshes generated from the prompt "a compact structure featuring a steep gabled roof with an asymmetrical extension on one side"

As we manually loop through the generated mesh, there are two main reasons for the failure cases during generation. First, the original caption is wrong. For example, in the easy category, three out of the five captions are wrong as shown in the Table 6.

Table 6 Examples of inaccurate captions from the "easy" category

Caption	Why wrong	Mesh
A simple structure with a single sloping shed roof	A shed roof is sloping in one direction only.	
A compact structure with a simple hipped roof	A hipped building has four sloping roofs.	
A simple structure with a symmetrical gabled roof on a rectangular base.	The model roof is not symmetrical.	

Multiple ways exist to improve the caption quality. The most accurate method would probably be manually cleaning and correcting the caption. However, this would be expensive and time-consuming. Alternatively, we may consider finetuning a caption model or using more in-context learning examples to correct the LLM. Another potential area to explore to change the method of rendering, the current rendering method was basic due to the limitation of Pytorch3D. Explore other rendering methods such as wireframe rendering that allow the VLM to see through the mesh may reduce some confusion.

Second, as the temperature increases, the autoencoder failed to decode the codes into sensible structures and hence produced a cluster of exploded mesh.

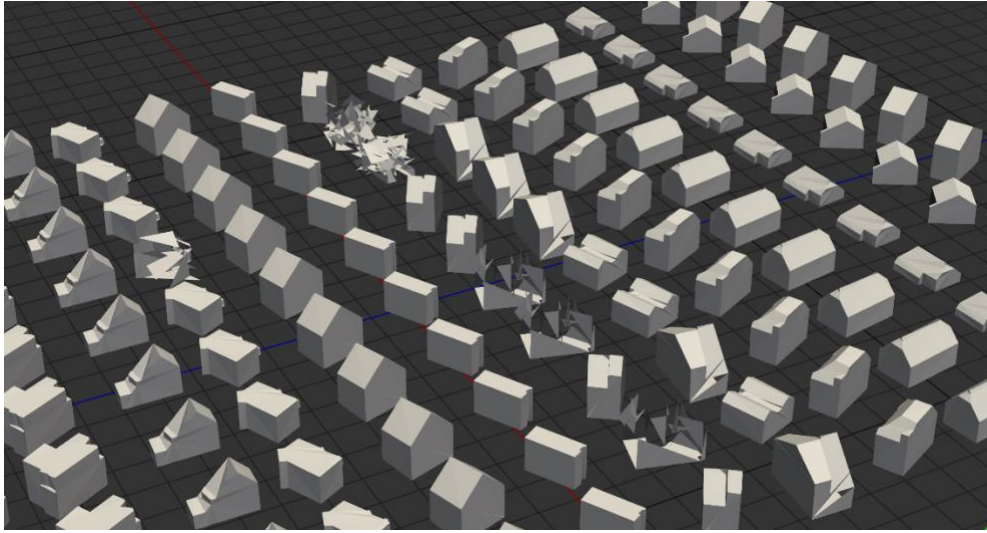


Figure 11 Examples of exploded mesh during inference

Notes on Shape Completion Task

While it may be assumed that shape completion is trivial due to the transformer architecture, this is not the case. MeshGPT differs from common language models in that encoding occurs before tokenization. In other words, each face token is computed after an encoding step that integrates information from neighboring faces via a GNN. This has two important implications:

First, encoding a partial mesh does not yield a corresponding subset of the full mesh encoding — that is, $\text{encoder}(\text{faces}[:\alpha]) \neq \text{encoder}(\text{faces})[:\alpha]$ where α is a number between 0 and 1.

Second, even if we take the first few tokens from the encoding of a full mesh and use them as a partial condition, the actual decoded faces may not contain the original faces. This is because depending on which token sequence is generated; the final decoded faces will change.

As a result, MeshGPT is not naturally suited for shape completion tasks without architectural or training modifications. To achieve shape completion, we may need to train a separate transformer that is cross-conditioned on the encoded partial mesh faces and generated all the face tokens.

LlamaMesh

The model is trained using LlamaFactory [46] using Google Colab A100 for a few hours. Due to memory constraint of an A100, I was only able to conduct lora finetuning and partial finetuning of the last 2 layers of the llama architecture. After 3 epochs of training, we observe significant loss decrease. However, The generated mesh does not make sense. We did not continue training.

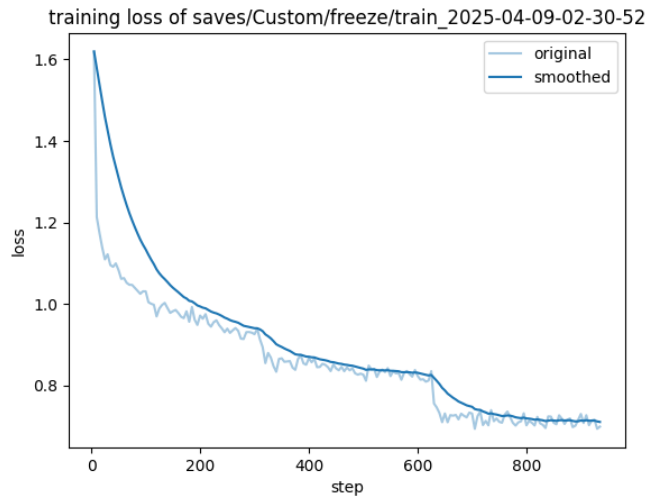


Figure 12 Training loss graph of LlamaMesh model.



Figure 13 Generated mesh after finetuning

Overall, while MeshGPT demonstrated promising results in generating common building forms from text, especially for gabled and flat roof types, its performance dropped with paraphrased prompts or less frequent architectural configurations. The evaluation highlighted both the

potential and current limitations of using language-conditioned generation for 3D architectural data. In contrast, experiments with LlamaMesh—despite initial signs of learning—did not produce coherent outputs, largely due to limited training and model adaptation challenges. Taken together, these findings point to a generative pipeline that works under constrained settings but struggles with generalization. These observations raise important questions about model robustness, dataset diversity, and how designers might meaningfully interact with such tools—issues further explored in the following reflection.

Reflection and Future Work

One of the key achievements in this research is demonstrating that vector-based representations for generative 3D architectural data offer significant advantages over traditional voxel-based methods. Notably, vector representations such as meshes can inherently encode diagonals without requiring post-processing or additional interpretation steps. This contrasts with voxel-based approaches, which often suffer from resolution constraints and blocky artifacts, especially when attempting to represent oblique or non-orthogonal forms. The ability to generate clean, accurate geometries directly from the model output marks a meaningful advancement for architectural applications, where precision and formal expressiveness are essential.

Revisiting Kyle Steinfeld's theory on AI-human relationship

Looking back through the lens of Kyle Steinfeld's theory on AI-human interaction, all three relational modes—**Material**, **Actor**, and **Provocateur**—manifested at various stages of the project, just as hypothesized.

First, **ML as Material** was most apparent during the dataset preparation and caption generation process. Preprocessing the 3DBAG dataset required substantial human effort to clean, decimate, and reformat the models, echoing the craft-like manipulation Steinfeld describes. More strikingly, prompt engineering for GPT-4o to produce consistent architectural captions involved iterative tuning and testing. The model initially struggled to unify eight rendered views into a single caption, prompting several adjustments in instruction phrasing before a reliable format was reached. This tuning process mirrors traditional material practice—shaping the AI through targeted constraints to unlock its generative potential.

Second, **ML as Actor** emerged in the interpretation of model outputs. Especially during evaluation, the transformer occasionally produced outputs that deviated from human expectations but nonetheless revealed coherent spatial logic learned from the data. For example, in the case of ambiguous prompts like “a compact structure with asymmetrical extensions,” the model surfaced variations that demonstrated an emergent grammar of form—suggesting a form of agency. Even when the outputs were imperfect, they sometimes revealed underlying biases in the dataset or latent spatial typologies, offering a new perspective to the designer.

Third, **ML as Provocateur** was evident in the limitations of the generation process. While the model performed well under constrained conditions, its outputs degraded when prompts were paraphrased, or temperature increased. These “failures” were not purely technical deficits—they served as prompts for critical design reflection. They pushed the boundaries of what could be reliably encoded and surfaced deeper questions about architectural generalization and model interpretability. In this sense, the model acted as a creative stimulus, exposing hidden assumptions in both dataset composition and prompt design.

Together, these observations affirm the project's underlying hypothesis: while generative AI can reproduce certain architectural logics, it does so within a human-curated framework. More importantly, these models are not merely tools—they are collaborators, materials, and provocateurs. This triangulated relationship underscores a broader shift in architectural

authorship, where meaning emerges not from the AI alone, but from the interplay between design intent, dataset construction, and computational inference.

This perspective also reframes a common question I get regarding this thesis: Does generative AI threaten the agency of the architect? To address this, it's important to distinguish between different kinds of AI models. In this thesis, I trained a narrow AI—a model specialized on a carefully curated dataset of 3D architectural forms and guided through structured prompts. Unlike large-scale foundation models trained on vast, general-purpose datasets, narrow AI systems exhibit limited generative capacity and operate within a tightly constrained design space. Not to mention that based on the result section, the capacity of deep learning model is no way near human architect. In fact, the tedious and highly manual process of preparing the dataset, tuning captions, and guiding prompt formats is itself a demonstration of human agency. Far from automating the creative process, these efforts reflect an iterative, craft-like collaboration between human and machine. Rather than replacing the role of the designer, narrow AI highlights the active participation and authorship of the architect in shaping how and what the model learns.

Next Steps

Looking back at the dataset and training setup, the MeshGPT transformer was trained based on a subset of approximately 10,000 building meshes, constrained by available GPU budget and time. While this subset offered a rough representation of common typologies—especially gabled and flat-roof houses—it also exhibited clear imbalances. Some roof types, such as multi-gable or cross-gable configurations, were underrepresented, which likely contributed to inconsistent generation results for those categories. Randomly sampling more from the 3DBAG dataset is still likely to leave rare or complex building forms underrepresented. This suggests that simply scaling up may not fully resolve the issue of data imbalance.

To address this, future work need move beyond random sampling and adopt a more strategic data expansion approach—using shape similarity metrics to identify redundant forms and prioritize the inclusion of novel, diverse geometries. Additionally, adopting a workflow that mirrors the original MeshGPT paper in scale (operating with ~30,000 meshes or more) and assigning multiple captions per shape could potentially improve both semantic richness and generalization. Ideally these adjustments would not only help balance category representation but also push the model toward learning deeper architectural variation, enabling more reliable and expressive generative outputs.

Originally, MeshGPT incorporated text conditioning by performing cross-attention using a single, mean-pooled embedding of the entire text—essentially an average token that captures overall semantic content. However, Inspired by the attention mechanism used in image caption task [42], I plan to modify the approach to use the entire sequence of token embeddings to support more complex and nuanced language inputs. By enabling cross-attention over individual tokens rather than just a summary vector, the model can leverage fine-grained linguistic details and contextual relationships between words, thereby enhancing its ability to interpret and respond to more complicated language descriptions.

While this thesis explores the generation of final 3D meshes directly, an alternative approach worth considering is the generation of a sequence of parametric commands, such as those used in Grasshopper. In this design pipeline, the generative model would output a script or set of construction steps rather than a fixed geometry, enabling more control and editability. This approach aligns with how many architects already work—using parametric logic to define spatial

relationships and forms—and could potentially integrate more seamlessly into existing design workflows. Another promising direction involves using Constructive Solid Geometry (CSG) [31], where complex forms are built through Boolean operations on primitive shapes. CSG offers a more interpretable and hierarchical way of encoding design logic, which could be particularly valuable for maintaining semantic meaning throughout the generative process. There is currently less scholarship on it for learning, making it the next interesting and potential research area.

The availability of new 3D indoor datasets presents exciting opportunities for training and evaluating generative architectural models. One such dataset, recently introduced in [24], contains high-quality, labeled 3D indoor scenes that include detailed spatial. Datasets like this one allow for more nuanced learning of architectural conventions, such as room adjacencies and circulation patterns. The inclusion of such rich contextual information supports more sophisticated generation tasks and improves the model’s ability to produce plausible and functionally viable architectural outputs.

Bibliography

- [1] Automated 3D Reconstruction of LoD2 and LoD1 Models for All 10 Million Buildings of the Netherlands: <https://research.tudelft.nl/en/publications/automated-3d-reconstruction-of-lod2-and-lod1-models-for-all-10-mi/datasets/>. Accessed: 2024-11-23.
- [2] Brown, T.B. et al. 2020. Language Models are Few-Shot Learners. arXiv.
- [3] Campagnolo, D. et al. 2023. Fully Automated Scan-to-BIM Via Point Cloud Instance Segmentation. *2023 IEEE International Conference on Image Processing (ICIP)* (Oct. 2023), 291–295.
- [4] Chaillou, S. 2022. *Artificial intelligence and architecture: from research to practice*. Birkhauser Verlag GmbH.
- [5] Chen, J. et al. 2024. BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. arXiv.
- [6] DEEPRISE: https://jinmorhee.net/jinmorhee_3-multimedia/deepriseparti.html. Accessed: 2024-12-11.
- [7] Ferrando, C. 2018. *Towards a Machine Learning Framework in Spatial Analysis*.
- [8] Finch – Optimizing Architecture: <https://www.finch3d.com/>. Accessed: 2024-05-07.
- [9] Guida, G. et al. Neural 3D Synthesis. *43rd Annual Conference for the Association for Computer Aided Design in Architecture* (Denver), 420–428.
- [10] Hasey, M. 2022. *The 3D Form Analysis of Regional Architecture Using Deep Learning: A Case Study of Wood Churches from the Carpathian Mountains*. Carnegie Mellon University.
- [11] Hu, R. et al. 2020. Graph2Plan: Learning Floorplan Generation from Layout Graphs. *ACM Transactions on Graphics*. 39, 4 (2020), 3–4. DOI:<https://doi.org/10.1145/3386569.3392391>.
- [12] Jayaraman, P.K. et al. 2023. SolidGen: An Autoregressive Model for Direct B-rep Synthesis. arXiv.
- [13] Kadaster 2025. Basisregistratie Adressen en Gebouwen (BAG).
- [14] Kerbl, B. et al. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. arXiv.
- [15] Leimer, K. et al. 2022. LayoutEnhancer: Generating Good Indoor Layouts from Imperfect Data. *Proceedings - SIGGRAPH Asia 2022 Conference Papers* (2022).
- [16] Li, Y. 2018. *A Topology-Based Architectural Search Engine*. Carnegie Mellon University.
- [17] Lin, L. et al. 2022. Capturing, Reconstructing, and Simulating: The UrbanScene3D Dataset. *Computer Vision – ECCV 2022* (Cham, 2022), 93–109.
- [18] Liu, Z. et al. 2023. MeshDiffusion: Score-based Generative 3D Mesh Modeling. arXiv.
- [19] Ma, X. et al. 2024. Generating 3D House Wireframes with Semantics. arXiv.
- [20] McInnes, L. et al. 2020. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv.
- [21] Mildenhall, B. et al. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. arXiv.
- [22] Müller, L.-M. 2023. 3D Generative Adversarial Networks to Autonomously Generate Building Geometry. (2023).
- [23] (PDF) 3D Model Generation on Architectural Plan and Section Training through Machine Learning: https://www.researchgate.net/publication/337298529_3D_Model_Generation_on_Architectural_Plan_and_Section_Training_through_Machine_Learning. Accessed: 2024-11-21.
- [24] (PDF) ArchComplete: Autoregressive 3D Architectural Design Generation with Hierarchical Diffusion-Based Upsampling: 2025. https://www.researchgate.net/publication/387382273_ArchComplete_Autoregressive_3D_A

rchitectural_Design_Generation_with_Hierarchical_Diffusion-Based_Upsampling.

Accessed: 2025-04-24.

- [25] PlanFinder: <https://www.planfinder.xyz>. Accessed: 2024-05-07.
- [26] PyTorch3D · A library for deep learning with 3D data: <https://pytorch3d.org/>. Accessed: 2025-05-06.
- [27] Rababaah, A.R. 2024. Building Architectural Styles Classification Using Convolutional Neural Networks Models. *Machine Learning for Real World Applications*. D.K. Sharma et al., eds. Springer Nature. 235–259.
- [28] Rombach, R. et al. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. arXiv.
- [29] Selvaraju, P. et al. 2021. Buildingnet: Learning to label 3d buildings. *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), 10397–10407.
- [30] Shabani, M.A. et al. 2023. HouseDiffusion: Vector Floorplan Generation via a Diffusion Model with Discrete and Continuous Denoising. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2023-June, (2023), 5466–5475. DOI:<https://doi.org/10.1109/CVPR52729.2023.00529>.
- [31] Sharma, G. et al. 2018. CSGNet: Neural Shape Parser for Constructive Solid Geometry. arXiv.
- [32] Siddiqui, Y. et al. 2023. MeshGPT: Generating Triangle Meshes with Decoder-Only Transformers. arXiv.
- [33] Siddiqui, Y. et al. 2023. MeshGPT: Generating Triangle Meshes with Decoder-Only Transformers. *arXiv.org* (Nov. 2023).
- [34] Steinfeld, K. 2021. Significant Others - Machine learning as actor, material, and provocateur in art and design. *The Routledge companion to artificial intelligence in architecture*. I. As and P. Basu, eds. Routledge.
- [35] Sun, J. et al. 2022. WallPlan: Synthesizing Floorplans by Learning to Generate Wall Graphs. *ACM Transactions on Graphics*. 41, 4 (2022). DOI:<https://doi.org/10.1145/3528223.3530135>.
- [36] Wang, K. et al. 2021. Roominoes: Generating Novel 3D Floor Plans From Existing 3D Rooms. *Computer Graphics Forum*. 40, 5 (Aug. 2021), 57–69. DOI:<https://doi.org/10.1111/cgf.14357>.
- [37] Wang, R. et al. 2023. Building3D: An Urban-Scale Dataset and Benchmarks for Learning Roof Structures from Point Clouds. arXiv.
- [38] Wang, Z. et al. 2024. LLaMA-Mesh: Unifying 3D Mesh Generation with Language Models. arXiv.
- [39] Wei, Y. et al. 2023. BuilDiff: 3D Building Shape Generation using Single-Image Conditional Point Cloud Diffusion Models. arXiv.
- [40] Wen, Z. et al. 2023. AnyHome: Open-Vocabulary Generation of Structured and Textured 3D Homes. (2023).
- [41] Wu, W. et al. 2019. Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics*. 38, 6 (Dec. 2019), 1–12. DOI:<https://doi.org/10.1145/3355089.3356556>.
- [42] Xu, K. et al. 2016. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. arXiv.
- [43] Xu, X. et al. 2024. BrepGen: A B-rep Generative Diffusion Model with Structured Latent Geometry. arXiv.

- [44] Zheng, H. and Ren, Y. 2020. Machine Learning Neural Networks Construction and Analysis in Vectorized Design Drawings. *RE: Anthropocene, Design in the Age of Humans - Proceedings of the 25th International Conference on Computer-Aided Architectural Design Research in Asia, CAADRIA 2020*. 2, (2020), 709–718.
- [45] Zheng, J. et al. 2020. Structured3D: A Large Photo-Realistic Dataset for Structured 3D Modeling. *Computer Vision – ECCV 2020*. A. Vedaldi et al., eds. Springer International Publishing. 519–535.
- [46] Zheng, Y. et al. 2024. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. arXiv.
- [47] Zhuang, X. et al. 2023. Synthesis and generation for 3D architecture volume with generative modeling. *International Journal of Architectural Computing*. 21, 2 (Jun. 2023), 297–314. DOI:<https://doi.org/10.1177/14780771231168233>.
- [48] 2018. *Fall 2018 Lecture Series: Jennifer Bonner*.

Appendix

Preliminary Testing of General Model Capabilities

In this stage, I evaluated the general capabilities of MeshGPT, an autoregressive model, in generating architectural meshes. Using the **BuildingNet dataset**, which contains detailed segmented 3D models of buildings, I aim to train MeshGPT without adding any specific conditional inputs. The objective here is to test the model's baseline ability to generate new building meshes and evaluate its performance in representing spatial relationships inherent in architectural forms. This initial phase serves as a foundation for understanding the model's potential and limitations before progressing to more targeted use cases. The performance of the model can be evaluated both qualitatively and quantitatively by retrieving the most similar existing model from the dataset.

I trained a MeshGPT [32] model on 10000 building models from the 3DBAG dataset [1]. The meshes are preprocessed into triangle mesh and I manually filtered out models with 12 – 50 faces so that it requires less training resources. I trained the model with a Nvidia 3080ti GPU.

After ~ 160 epochs of training, the MeshGPT autoencoder achieved a desirable loss of ~ 0.3. Manuals inspecting the model show that the reconstructed mesh from the codes matches the original model greatly. A transformer is then trained to generate the codes generated by the autoencoder and it reaches the loss at around 0.013. The generation is conditioned on the text as all the models are provided with a “house” label, which is generally believed to speed up the model convergence. The model can generate examples from the training dataset, as shown below. However, to generate novel mesh, it may require at least 100 thousand meshes.

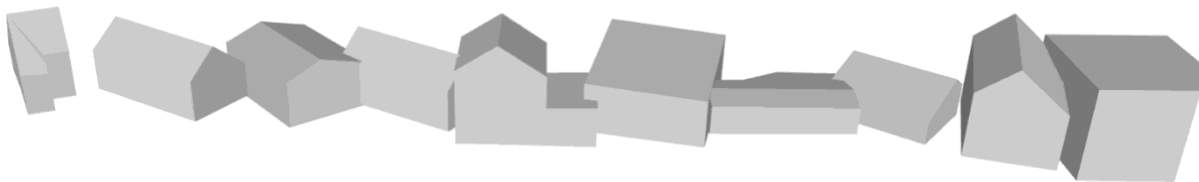


Figure 14 An example of houses that are autoregressively generated MeshGPT

Meanwhile, I faced many challenges before I got the model to work properly. Some of the knowledge I gained in the process include:

- The MeshGPT autoencoder is not very generalizable. I was using the pretrained autoencoder directly and although the transformer became low, the decoded mesh was nothing like that of the model.

At the same time, I was also working on the simplification of the more complicated dataset, BuildingNet. The current mesh simplification method creates holes in the 3D model data. I plan to fix that by patching the holes with simple faces on it.

Plan for Cross Model Benchmarking Evaluation

Since the Goal: Comparing the output of multiple different models, get feedback on use cases with current dataset, give preliminary idea on how this can be furthered on custom dataset. Hence Two type of evaluation gather method will be used: Latin square design survey and Focused group interview with studio instructor and 2 students. The design survey helps us to give quick ranking of the performance of different model. While the open feedback session will give more in-depth discussion.

Questionnaire

The Questionnaire is designed to fulfil two purposes: first, it evaluates the quality of the caption generated by gpt4, which is later used in model training; second, it asks the participants to rank the output of four different models (MeshGPT, Llama-Mesh, Wire3d, Bergen) in terms of aesthetics and text-adherence. The questionnaire is designed such that the rank part comes first while the Caption evaluation comes later.

Focus Group Discussion

Focus Group Discussion is very important to the author because arguably the research question and experiment setup is designed from the perspective of technology and available dataset. In other words, talking with people from architecture domain is helpful for localization of the project refining the approach for future extension.

BrepGen

In BrepGen, the boundary representation (Brep) is generated via a multi-stage diffusion process that sequentially constructs the face bounding box, face shape, edge bounding box, and finally the edge shape. To condition this complex generation pipeline with language, we propose injecting text guidance by applying a linear projection to the text embedding and adding it to the corresponding mesh embeddings.

Originally, the denoising process of the node feature x can be presented as

$$p(x) = p(F, E, V) = p(E_{ZV} \mid E_P, F) \cdot p(E_P \mid F) \cdot p(F_Z \mid F_P) \cdot p(F_P \mid \emptyset)$$

Were

$$- F = F_P + F_Z$$

- F_P : Bounding box representation $[x_1, y_1, z_1, x_2, y_2, z_2]$ encoding the **bottom-left** and **top-right** corners.

- F_Z : Encoded geometric information derived from the parametric surface $F_Z = \text{face_encoder}(F_s)$

where F_s is a 2D array of 3D points sampled on the parametric surface.

$$E = E_P + E_Z$$

- E_P Bounding box representation like F_P :

- E_z : Encoded information from sampled points:

All the denoiser network are implemented as Transformers. Without loss of generality, we will use edge position denoiser (Figure 3 bottom-left) as an example. Input to the denoiser are edge tokens with noisy bounding box parameters. Network is conditioned on the previously denoised face tokens, which are embedded as:

$$F \leftarrow MLP(W_p F_p) + MLP(W_z F_z)$$

Now we include a T which represents encoded text, which gives

$$F \leftarrow MLP(W_p F_p) + MLP(W_z F_z) + MLP(W_t T)$$

Where T is the pooled text embedding derived from T5 encoder and W_t is the projection matrix for text embedding.