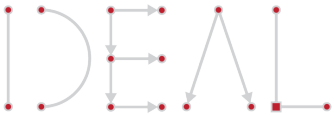
Generative AI for Woodworking Designs

Ryan Slocum
Matriculation Number: 24-936-510

Thesis Supervisors:
Dr. Milli Schlafly
Prof. Dr. Mark Fuge

In Partial Fulfillment of the Requirements
for the Degree of
Master of Science ETH in Robotics, Systems and Control



Swiss Federal Institute of Technology Zurich
Department of Mechanical and Process Engineering (D-MAVT)
IDEAL – Chair of Artificial Intelligence in Engineering Design

Fall Semester 2025

Abstract

In recent years, the potential impact of generative AI on human creativity has been increasingly realized. However, there remains a significant gap in the application of these technologies to physical design and manufacturing. This project aims to bridge that gap by developing a system for creating woodworking designs using generative AI. Four different methods for generating designs from text prompts are proposed and implemented, including supervised fine-tuning, reinforcement learning, and agent-based approaches. The generated designs are then evaluated based on their performance on a series of prompts. The results show that in the complex task of creating assembleable woodworking designs, agent-based approaches outperformed fine-tuned models, although there exists room for improvement with both approaches.

Acknowledgments

I would like to thank my supervisor, Dr. Milli Schlafly for her support and guidance throughout this project.

Contents

List of Figures

List of Tables

Abbreviations

GAI	Generative Artificial Intelligence
GRPO	Group Relative Policy Optimization
LLM	Large Language Model
VLM	Vision Language Model
RLHF	Reinforcement Learning with Human Feedback
SFT	Supervised Fine-Tuning

1 Introduction

In recent years generative AI has proved useful for augmenting human creative output, especially in the fields of writing and software development. Tools like ChatGPT have increasingly found their way into the workflows of many professionals (Sidoti et al. 2025), allowing for rapid ideation and iteration.

However, in the realm of physical manufacturing, fewer tools exist to offer end-to-end solutions for rapid prototyping and ideation, especially when it comes to creating complex, multi-component assemblies. Section 2 discusses existing methods in 3D generative AI, generative AI for manufacture, and generative AI for automated assembly.

Section 3 presents a few methods that allow users to generate and quickly iterate on woodworking plans, designed for autonomous construction using a two-armed robotic platform. In Section 4, the introduced methods are evaluated by generating many designs and evaluating each on the basis of prompt satisfaction, assembleability, and design quality. The results and their implications are then discussed in Section 5.

The contributions of this work are as follows:

- A dataset of woodworking plans paired with text descriptions.
- An exploration of SFT and reinforcement learning techniques for the task of generative design for woodworking.
- A web-based interface utilizing LLMs as agents to generate woodworking plans.
- A quantitative review assessing the quality of generated designs from all four methods.

2 Related Works

There are, to the author's knowledge at the time of writing, no existing solutions for generating woodworking projects from text prompts. However, there are several works related to this goal. This section will examine three main areas: generative AI for 3D design, generative AI for assemblies, and design for automated assembly.

2.1 Generative AI for 3D Design

There are many works that focus on the generation of 3D models from text or image prompts, with or without a 3D shape prior. Some methods are diffusion-based, such as Ntavelis et al. 2023, or more recently, based on Gaussian splatting as in Tang et al. 2024 and Zhang et al. 2024. These methods are notable and useful for the generation of 3D models, but they focus largely on the generation of free-form, complex meshes. For the task of creating woodworking designs, low-poly meshes that could be constructed out of rectangular or planar wooden pieces are more relevant.

Works that focus on less complex meshes include MeshGPT (Siddiqui et al. 2023) and LLaMa-Mesh (Z. Wang et al. 2024), which both use a transformer-based architecture to generate 3D meshes with a lower triangle count. Both works take advantage of the autoregressive nature of transformer-based models in order to construct meshes that have a progressive structure.

LLaMa-Mesh is particularly relevant, as it utilizes pre-trained foundation models to generate 3D meshes in the OBJ format. This approach shares many similarities with what is attempted in this project: they attempt to take a natural language prompt and turn it into a 3D design, represented by a series of construction steps. They do this by creating a large dataset of prompts and their corresponding 3D models from the Objaverse dataset (Deitke et al. 2023), which they then use to train a 1B parameter foundation model using SFT. This paper yields robust results in both object diversity and prompt satisfaction. The meshes also maintain a level of complexity comparable to those generated with diffusion or Gaussian splatting methods. However, this work is limited in that it does not consider the physical manufacturability of its meshes, nor does it allow for the creation of multi-component assemblies.

2.2 Generative AI for Assemblies

In another step towards the goals of this project, there exist several works that focus on generating 3D assemblies based on a variety of inputs. Image2Lego (Lennon et al. 2021) uses an autoencoder to convert an image input into a voxelized structure, then uses a deterministic algorithm to generate a toy brick structure, complete with assembly directions, from the voxels.

PartPacker (Tang et al. 2025) also takes a single image as input, and outputs a 3D model with an arbitrary number of semantically meaningful parts. They achieve this by utilizing a VAE architecture with generation conditioned on the latent space representation of the input image. Their results are impressive, with parts being segmented in a semantically meaningful way, suggesting potential applicability for creating an assembly plan for woodworking plans from an image. However, due to the limitation of robotic assembly requiring a set library of parts, it would be challenging to create part generations that are suitable for manufacture.

For more technical parts and assemblies, a recent development in the field is SGS-1 (Spectral Labs 2025), which is a product that takes multimodal input (3D models, text, and images), and is capable of creating parametric CAD models and assemblies. SGS-1 could be helpful for this project, but unfortunately it is a proprietary product, and the creators have released their results without implementation details.

2.3 Design for Automated Assembly

There also exist many works focused on generative design for robotic assembly, specifically in the form of building simple rectangular building blocks. Speech to Reality (Kyaw et al. 2025) is one such paper, focusing on an end-to-end pipeline to go from speech input to robotic assembly of a generated design. They achieve this by converting the input to a 3D voxel grid, after which they process the grid into assembly instructions and have a robotic arm assemble blocks to

meet the user's request. This system is exactly what this project aims to achieve. However, the designs generated are very simple, and the building medium is also less versatile than the wooden blocks slated for use in this project.

With slightly more complex designs for robotic assembly, BrickGPT (Pun et al. 2025) focuses on the task of creating toy brick structures from text prompts. BrickGPT uses a similar strategy to LLaMa-Mesh Z. Wang et al. 2024, in that they construct their own dataset of text-structure pairs from ShapeNet Chang et al. 2015 models, and fine-tune a 1B parameter model using SFT. They are more limited in the models they can produce, with only 21 object categories represented in their dataset, but they were successful in generating structures that are physically realizable and assembleable by robots. The primary limitation here is that the designs are also voxel-based, and thus not sufficiently complex for woodworking.

With a markedly different approach, BloxNet Goldberg et al. 2025 attempts to create robotically assembleable structures from blocks. They query a VLM for a design made of blocks from their library, provide the VLM with simulated results of that design, and allow it to iterate on its design to make improvements as deemed necessary. They perform this many times across different VLM instances, and then perform an iterative selection process to narrow the options down to one final design, where yet another VLM instance picks the best of pairs of designs. The results of this paper are promising, with an impressive diversity of objects constructed in their experiments, and reasonably good prompt satisfaction despite the limited library of parts available. The designs are also assembleable by a robot with a gripper.

3 Method

When designing a system that creates woodworking designs, there are three main required attributes:

1. An understanding of the natural language input prompt
2. The ability to reason about complex designs in 3D space
3. Generation of output as a series of parts in assembly order

With these requirements, Large Language Foundation Models are a natural fit. These models have an inherent understanding of natural language as a part of their training on vast text corpora, and they also have the autoregressive property that is suitable for assembly, where the placement of the next part is dependent on the placement of previous parts. The reasoning ability of the foundation models varies, but it has been shown that these models can have enough capacity to take on challenging tasks where reasoning is required.

The initial plan for this project was to replicate previous works by fine-tuning an LLM for the woodworking design task. This approach will be discussed in

Subsection 3.1 details a fine-tuning approach adapting the methods of previous works

to the task of generating woodworking designs. Subsequently, Subsection 3.2 presents an agent-based approach taking advantage of both the physical reasoning and code generation capabilities of the most recent foundation models.

3.1 Fine-tuned Models

The goal of this subsection was to replicate the results of works such as BrickGPT (Pun et al. 2025) and LLama-Mesh (Z. Wang et al. 2024), but with the added complexity of being able to place parts in a more continuous assembly space. Whereas placed bricks in BrickGPT have five possible parameters (pose of X, Y, and Z, as well as dimensions of length and width), woodworking designs require that parts can be defined with more parameters (at the very least, a 6-dimensional pose and a part identifier/dimension).

To solve this problem, a dataset must be generated representing the kinds of designs that one might want to create. Fine-tuning may then proceed using SFT and the newly created dataset. Finally, to encourage the generation of physically realizable designs, GRPO (Shao et al. 2024) is used to provide the model with rewards for good part placements during training.

3.1.1 Dataset Generation

There are some existing datasets for the assembly of parts in 3D space that could be used for training, but none suffice for the specific task tackled by this project. BrickGPT (Pun et al. 2025) presents StableBrick, a dataset of text prompts and brick structures from a variety of item categories, but these structures are not well suited to construction with wood. Ben-Shabat et al. 2020 presents a dataset of humans assembling furniture, but the designs themselves are not well represented, and there are only a few dozen furniture items represented. IKEA-Manual (R. Wang et al. n.d.) offers a dataset that would be perfect for this project, with part 3D models, an assembly plan, and text labels, but there are only 102 objects in the dataset, making the scale of the data insufficient for the planned training with SFT.

The most promising dataset of 3D models decomposed into parts is called PartNet (Mo et al. 2019), a dataset containing over 26k 3D models from ShapeNet, along with high-quality part-level annotations from human reviewers. This means that with the right preprocessing, it would be possible to translate these models into wooden designs.

However, this is only half of the problem–the text labels describing what each item is are still missing. Fortunately, the StableBrick dataset does include design-text pairs, and these designs are also derived from ShapeNet models. This means that it is possible to take models from PartNet, labels from StableBrick, and to join them together such that the results meets the project's use case.

In the first stage of the data generation, 3D models from the PartNet dataset are filtered by category. Only models matching the categories *Bed*, *Chair*, *Table*, and *StorageFurniture* are kept, as these are the types of objects users of this project might want to create.

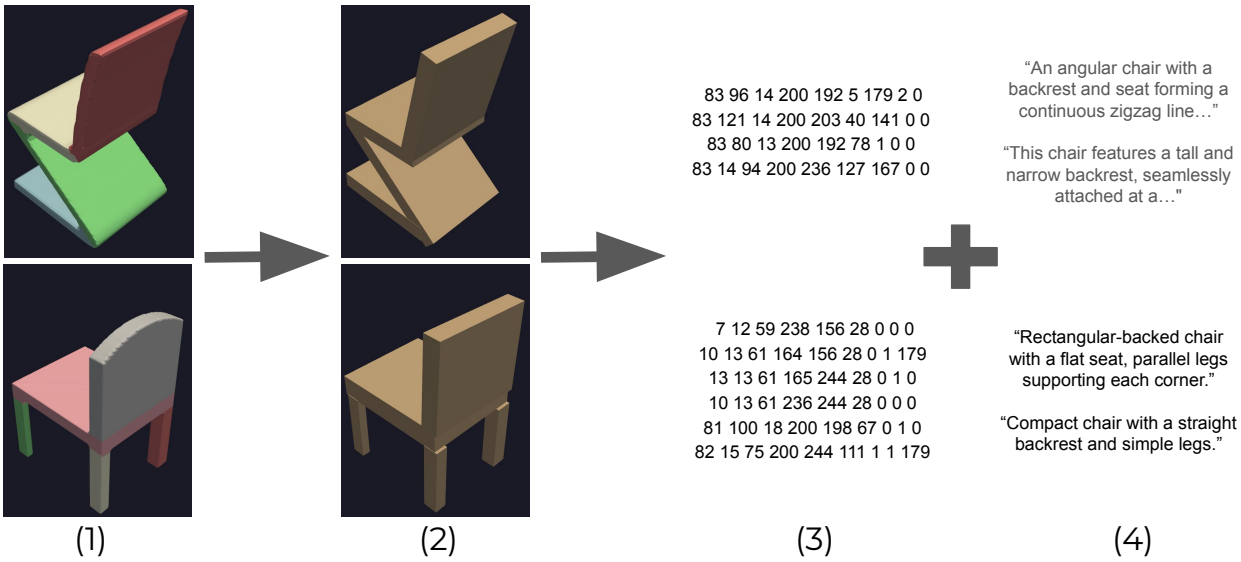


Figure 1: Generation of dataset in four stages: (1) Parts from PartNet are (2) fit to rectangular prisms, (3) encoded as text, and (4) paired with text labels from StableBrick

The models are then transformed such that they are a standardized size and centered in the middle of the build space. As there are multiple part hierarchies available within each PartNet model, the lowest level is taken, meaning that all annotated parts are fitted to a wooden part.

At this stage, each part is fitted to a wooden part. Each wooden part is represented as a rectangular prism, meaning that it has a 6 dimensional pose and 3 different dimensions: length, width, and height. The fitting of each part in the model is achieved by rotating the point cloud iteratively until it has the smallest possible axis-aligned bounding box. The dimensions of the bounding box and the inverse of the rotation are then used along with the centroid of the part to define the 6D pose and the dimensions of the wooden part.

There are limitations to this approach, mainly in that it does not allow for the construction of assemblies from a part library. However, due to the complex nature of the assemblies, it is challenging to retrospectively fit designs that use the same pieces, still resemble the original object, and fit within the build volume. There is also some loss of information involved in fitting a rectangular prism to a potentially non-rectangular piece–in many furniture items, curved or rounded pieces will not be well represented. However, an advantage of this method is that the resulting model should be able to represent a variety of objects due to its free choice in the dimensions of the parts it places.

After each design has a series of associated wooden parts, the parts can be encoded into a format that is understandable for the LLM. To do this, the parts are ordered from bottom to top by their Z-coordinate (a good approximation for assembly order), and encode each part as

a line of text. Each line has 9 numbers: the first three represent the length, width, and height of the part, the next three represent the x, y, and z coordinate of the centroid, and the last three numbers represent the Euler angles of rotation around the x, y, and z axes. The part is rotated such that none of the Euler angles are negative (this is possible due to the multi-axis symmetry of the wooden pieces). Combined with the fact that all numbers in the encoding are limited to three digits, this means that each line can be represented with a fixed number of tokens after tokenization, making parsing at inference time simpler and less error-prone.

Once all models have been processed into this text format, they are paired with text descriptions from the StableBrick dataset. The StableBrick dataset contains up to 5 text labels of varying lengths per 3D model. Because both the StableBrick and PartNet are derived from the ShapeNet dataset, it is relatively simple to join the generated text and the descriptions on the object ID. At the end of this process, the dataset consists of 68.4k text-design pairs.

Finally, the data pairs must be formatted in a way that is understandable for the LLM. This is done as below, with the model being provided context on how it is expected to respond, as is best practice for the Llama Instruct models that will be used for fine-tuning (Grattafiori et al. 2024).

SFT training format

System: You are a helpful assistant
User: Create a wooden model of the input. Format your response as a list of wooden pieces: <dimensions> <position> <rotation>, where piece position is x y z and piece rotation is rx ry rz. All values are space separated, and pieces are separated with a newline. No negative values are allowed. Rotations are in degrees from 0 to 179.
Input: `Compact chair with a straight backrest and simple legs.`
Assistant:

```
7 12 59 238 156 28 0 0 0
10 13 61 164 156 28 0 1 179
...
```

3.1.2 SFT Stage

After the dataset is created, everything necessary for fine-tuning a model is available. The foundation model used is the Llama 3.2 1B Instruct model, which was chosen due to its relatively small size (making training locally or on the Euler cluster tractable), the fact that it is pre-trained to follow instructions given by a user, and the results that were achieved by using the same model in the BrickGPT and LLaMa-Mesh papers. SFT is performed using the Hugging Face Transformers library (Wolf et al. 2020) for 3 epochs over the training data, which takes approximately seven hours on one NVIDIA RTX4090. Other training parameters are available in the project repository, see Appendix A.1. The model is saved at this point for later evaluation and comparison against other models.

3.1.3 GRPO Stage

At this stage, the SFT model is able to construct a design that resembles the input prompt given. However, there are many issues with the designs produced by this initial model: they feature overlapping parts, parts that are not connected to the main design, and other mistakes that make the final design physically unrealizable.

In an attempt to further fine-tune the model in the direction of assembleability, a reward function for part placements is developed. The idea is that if the model places a part in a way that satisfies a series of rules, that part placement can be rewarded. If it fails at this task, it will receive a smaller reward proportional to how close it was to achieving the goal. Algorithm 1 details the pseudocode for this approach, and Figure 2 provides a visualization of the cases that are tested.

Algorithm 1 Part Placement Reward Function

procedure RewardNewPart(newPart, design)
  if newPart is too small **then**
    return 0
  end if
  if newPart is out of bounds **then**
    return 0
  end if
  if newPart is below floor **then**
    return $1 - \tanh(\text{floor\_distance})$
  end if
  intersection_volume $\leftarrow$ IntersectionVolume(newPart, design)
  if intersection_volume > 0 **then**
    return $1 - \tanh(\epsilon \cdot \text{intersection\_volume})$
  end if
  distances $\leftarrow$ Distance(newPart, design)
  min_distance $\leftarrow$ Min(distances)
  return $1 - \lambda \cdot \tanh(\text{min\_distance})$
end procedure

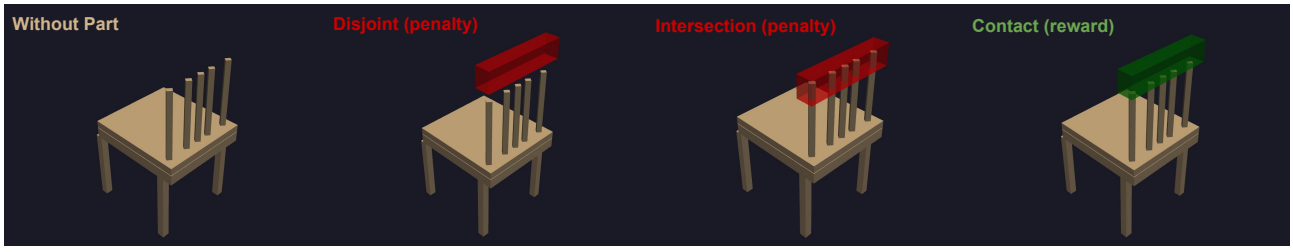


Figure 2: Visualization of some checks the reward function performs

To create opportunities for the model to generate part placement on a variety of models at a variety of construction stages, each entry in the SFT dataset is expanded to have one entry per part addition. This larger dataset is then randomly shuffled, and the first 100k entries are selected, as learning typically plateaus beyond this point.

For training, GRPO is used as the reinforcement learning algorithm (Shao et al. 2024), which allows the model to generate multiple parts per prompt, and calculate the "advantage" of each part, learning over time which part placements maximize the reward. For each part placement, 8 different completions are generated, and this process is performed over 100k part placements, which takes approximately 24 hours on an NVIDIA RTX4090. Other training parameters are available in the project repository, see Appendix A.1.

3.2 Agent-Based Method

Unspecialized LLMs can also excel at the task of creating 3D designs using either the text format discussed in Subsection 3.1.1, or by using code to generate their designs. This section presents another approach for generating woodworking designs taking advantage of these reasoning capabilities, with the goal of compare it against the method presented in Subsection 3.1.

This approach involves the interaction of two different agents, both of which are built with Google's Gemini 2.5 model family (Comanici et al. 2025). The first agent is the *Designer*, which is instructs it to construct a design that fulfills the user's requirements and is assembleable given the specific constraints of the robotic system. The second agent is the *Quality Assurance (QA)*, which is instructed to review artifacts of the design and provide feedback to the Designer. In the complete system, these two agents have a back-and-forth exchange with the objective of producing a better design.

The Designer agent is instructed to output CadQuery code, which allows it to define 3D designs in a text-based format. This decision was made for two reasons: because LLMs have been optimized to write code, and because code output allows the agent to write functions and loops that allow it to do repeated or more precise part placements as needed.

After the Designer agent has generated a design, the code is executed and artifacts are produced, including an image of the design from four viewing angles and the output of a series of tests. These tests are written to ensure assembleability of the design, and currently check the following about the design:

1. Code execution status (syntax errors).
2. Adherence to the available part library.
3. Absence of part intersections.
4. Connectivity of all parts.
5. Static stability of the design.

These tests are flexible and could be modified and expanded as more requirements for assembly change.

Once these artifacts are generated, the QA agent is asked to review the design image

and the test suite results and to give necessary feedback. It is instructed to conduct its review based on three main categories: user requirements, assembleability, and design quality. It may then choose to give feedback to the design agent, or to deem the design as having passed.

This back-and-forth between the two agents is automated: after the user provides their prompt, the Designer agent creates a design, the design code executes and produces the artifacts, the QA agent provides its feedback, and the Designer agent revises its code accordingly. This cycle repeats continuously up to a configurable number of iterations.

The code for the web interface facilitating the interaction between the user and these two agents is available in the link in Appendix A.1.

4 Experiments

To evaluate the performance of the different approaches proposed in Section 3, designs from a variety of text prompts are generated using each model. The models evaluated are listed in Table 1. The SFT and GRPO models are the two different stages listed in Subsection 3.1, whereas the Designer-QA and Designer models represent the agent-based approach (Subsection 3.2) with and without the feedback loop, respectively.

Property	**SFT Model**	**GRPO Model**	**Designer-QA**	**Designer**
Parameter Count	1B	1B	1T	1T
Part Library	No	No	Yes	Yes
Fine-tuned	Yes (SFT)	Yes (SFT + RL)	No	No
Locally-hosted	Yes	Yes	No	No

Table 1: Comparison of the four evaluated models and their key properties.

Each model is evaluated on 50 different text prompts of varying lengths, comprising 25 furniture items and 25 from other categories. Figure 3 illustrates a sample of the generated designs.

In the interest of having quantitative metrics, each design is evaluated on three different evaluation metrics:

- Prompt Satisfaction: Does the design satisfy the user's request?
- Assembleability: Does the assembly appear assembleable by a human or robot?
- Design Quality: Does the design appear useable and reflect the expected characteristics of the prompt?

The designs were scored on a scale from 1 to 3 in each of these evaluation metrics by the author and a VLM. The scores overall are summarized in Table 2, and the scores by object category in Table 3.

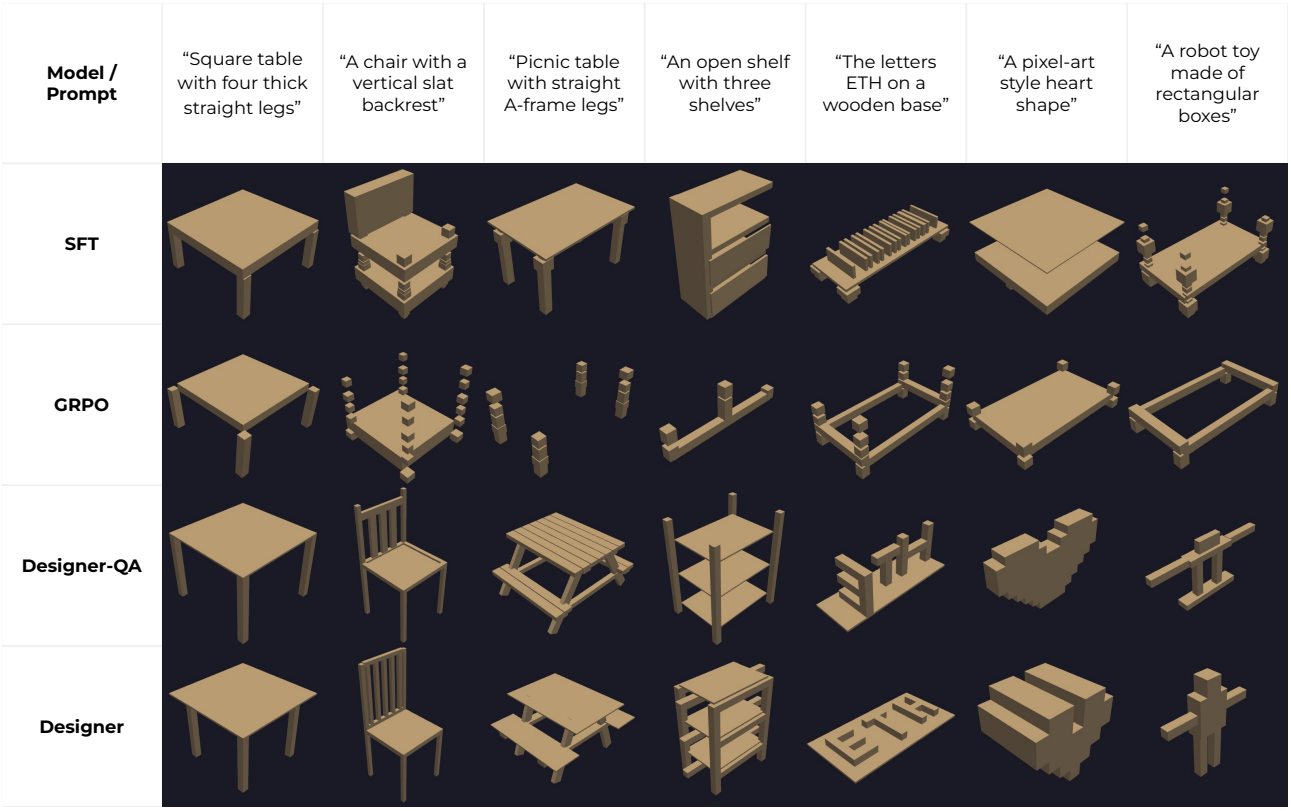


Figure 3: Example designs from each of the four models across seven different prompts

Metric	Reviewer	SFT	GRPO	Designer-QA	Designer
Prompt Satisfaction	Author	1.54	1.12	**2.78**	2.64
	VLM	1.62	1.26	**2.70**	2.56
	Average	1.58	1.19	**2.74**	2.60
Assembleability	Author	1.64	1.44	2.60	**2.66**
	VLM	2.22	2.08	2.46	**2.48**
	Average	1.93	1.76	2.53	**2.57**
Design Quality	Author	1.32	1.02	**2.44**	2.20
	VLM	1.56	1.20	**2.44**	2.12
	Average	1.44	1.11	**2.44**	2.16

Table 2: Breakdown of evaluation scores for each model, by metric and reviewer

Metric	Reviewer	SFT	GRPO	Designer-QA	Designer
Prompt Satisfaction	Furniture	1.92	1.32	**2.72**	2.62
	Other	1.24	1.06	**2.76**	2.58
Assembleability	Furniture	2.16	1.6	**2.62**	2.56
	Other	1.7	1.92	2.44	**2.58**
Design Quality	Furniture	1.76	1.18	**2.58**	2.20
	Other	1.12	1.04	**2.30**	2.12

Table 3: Breakdown of averaged evaluation scores by category

5 Discussion

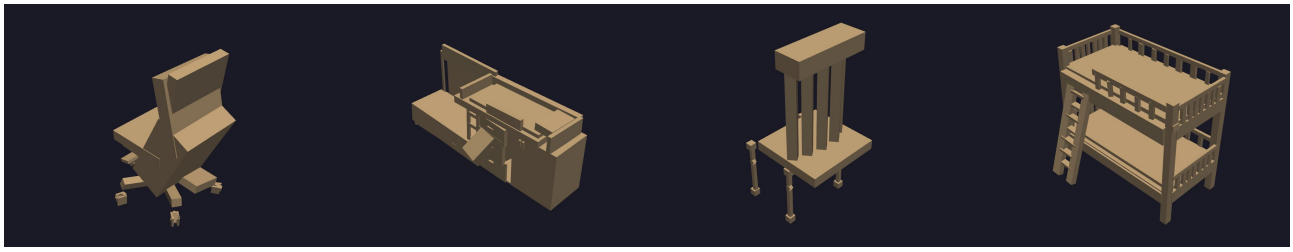
Overall, the SFT and the GRPO models underperform compared to the other two approaches. There is a major trade-off for the quality of the generated designs with the generation time.

Model	Mean	Min	Max
SFT Model	9.06	**1.80**	15.99
GRPO Model	**8.64**	1.81	**15.85**
Designer-QA	199.05	17.85	998.90
Designer	47.51	18.11	134.11

Table 4: Average, minimum, and maximum generation time per model (in seconds).

While the two fine-tuned models took a maximum of about 16 seconds to generate a design, the Designer and Designer-QA methods took significantly longer, due to the size of the models involved, the need to make API calls to the VLM host, and the time taken to execute the code and create the artifacts.

Due to the very limited and relatively poor-quality data that the SFT model was fine-tuned with, as well as its fairly small model size, it was not able to generalize well on what constitutes a good design, despite being trained on a dataset of only a few object categories. Figure 4 illustrates examples from the generated dataset that may have impeded training due to low data quality. The main way to improve results with the SFT model would likely be to provide it with more, higher-quality, more diverse data.



(1) (2) (3) (4)

Figure 4: Examples of data samples that: (1) don't meet assembleability constraints (2) don't represent well the prompt (3) contain anomalies in part fitting (4) are more complex than desired

The GRPO training stage seems to have worsened results when compared with the SFT model. There are a few possible explanations for this. First, it is likely that the reward function was susceptible to reward hacking. In the experiment results, (Figure 3), it is clear that the model has learned that the safest way to collect rewards is to make the smallest possible part, and to stack them on each other in such a way that they do not intersect. This could be solved by having a more robust reward function that avoids such hacking, or by using a neural network trained on assembleable part placements to provide the reward.

Another explanation for the degraded performance is that there was no reward for fulfilling the requirements of the prompt. One possible way to avoid this is to utilize the KL-divergence β term of the GRPO training algorithm to encourage the model to generate parts similar to the original SFT model. However, when testing a variety of values for KL-divergence weight, the model failed to improve in cases when $\beta > 0$. Another possible way to improve the semantic performance of the model would be to have a mixed reward function, which also

considers if a part placement brings the design closer to fulfillment of the prompt.

When comparing the Designer-QA and the Designer methods, the Designer-QA performs better in Prompt Satisfaction and Design Quality, whereas the Designer marginally outperforms in assembleability. This is unexpected due to the fact that the QA agent is largely responsible for assembleability, and should improve the score in this area by providing feedback based on the test results it has access to. However, the difference can be explained partially by the fact that the number of design iterations in the Designer-QA system was limited to 3, and thus it is possible that some designs were only refined to meet the user's requirements, and then cut off due to reaching the max number of iterations. Given more opportunities to refine the design (increasing max iterations to 5 or to 10), it is possible that the assembleability scores would have improved as well. A more comprehensive test suite would likely also be helpful.

6 Conclusion

This work proposed four different approaches to generating woodworking designs for robotic assembly from text. The quality of the designs generated by these four approaches was then evaluated over 50 different text prompts, and the results show that the larger VLM models without specific fine-tuning on the task consistently perform better in generating high-quality, assembleable designs.

Future works have several avenues for improvement, whether that be using a similar fine-tuning approach with a higher-quality dataset of wooden assemblies, investigating a different reward function for GRPO stage of fine-tuning, or introducing more robust test-cases for the agent-based method to iterate upon.

References

Ben-Shabat, Yizhak et al. (2020). "The IKEA ASM Dataset: Understanding People Assembling Furniture through Actions, Objects and Pose". In.

Chang, Angel X. et al. (2015). *ShapeNet: An Information-Rich 3D Model Repository*. arXiv: 1512.03012 [cs.GR]. URL: https://arxiv.org/abs/1512.03012.

Comanici, Gheorghe et al. (2025). *Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities*. arXiv: 2507.06261 [cs.CL]. URL: https://arxiv.org/abs/2507.06261.

Deitke, Matt et al. (2023). "Objaverse-XL: A Universe of 10M+ 3D Objects". In: *arXiv preprint arXiv:2307.05663*.

Goldberg, Andrew et al. (2025). "Blox-Net: Generative Design-for-Robot-Assembly using VLM supervision, Physics, Simulation, and a Robot with Reset". In: *2025 International Conference on Robotics and Automation (ICRA)*. IEEE.

Grattafiori, Aaron et al. (2024). *The Llama 3 Herd of Models*. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

Kyaw, Alexander Htet et al. (Nov. 2025). "Speech to Reality: On-Demand Production using Natural Language, 3D Generative AI, and Discrete Robotic Assembly". In: *Proceedings of the ACM Symposium on Computational Fabrication*. SCF '25. ACM, pp. 1–12. DOI: 10.1145/3745778.3766670. URL: http://dx.doi.org/10.1145/3745778.3766670.

Lennon, Kyle et al. (2021). *Image2Lego: Customized LEGO Set Generation from Images*. arXiv: 2108.08477 [cs.CV].

Mo, Kaichun et al. (June 2019). "PartNet: A Large-Scale Benchmark for Fine-Grained and Hierarchical Part-Level 3D Object Understanding". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ntavelis, Evangelos et al. (2023). "Autodecoding Latent 3D Diffusion Models". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., pp. 67021–67047. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/d3b93537b521f15613524415dfe43f37-Paper-Conference.pdf.

Pun, Ava et al. (2025). "Generating Physically Stable and Buildable Brick Structures from Text". In: *ICCV*.

Shao, Zhihong et al. (2024). *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*. arXiv: 2402.03300 [cs.CL]. URL: https://arxiv.org/abs/2402.03300.

Siddiqui, Yawar et al. (2023). "MeshGPT: Generating Triangle Meshes with Decoder-Only Transformers". In: *arXiv preprint arXiv:2311.15475*.

Sidoti, Olivia and Colleen McClain (June 2025). *34% of U.S. adults have used ChatGPT, about double the share in 2023*. Pew Research Center. URL: https://www.pewresearch.org/short-reads/2025/06/25/34-of-us-adults-have-used-chatgpt-about-double-the-share-in-2023/.

Spectral Labs (Sept. 2025). *Introducing SGS-1*. Research announcement. URL: https://www.spectrallabs.ai/research/SGS-1.

Tang, Jiaxiang et al. (2024). *DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation*. arXiv: 2309.16653 [cs.CV]. URL: https://arxiv.org/abs/2309.16653.

Tang, Jiaxiang et al. (2025). *Efficient Part-level 3D Object Generation via Dual Volume Packing*. arXiv: 2506.09980 [cs.CV]. URL: https://arxiv.org/abs/2506.09980.

Wang, Ruocheng et al. (n.d.). "IKEA-Manual: Seeing Shape Assembly Step by Step". In: *NeurIPS 2022 Datasets and Benchmarks Track*.

Wang, Zhengyi et al. (2024). *LLaMA-Mesh: Unifying 3D Mesh Generation with Language Models*. arXiv: 2411.09595 [cs.LG]. URL: https://arxiv.org/abs/2411.09595.

Wolf, Thomas et al. (2020). *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. arXiv: 1910.03771 [cs.CL]. URL: https://arxiv.org/abs/1910.03771.

Zhang, Bowen et al. (2024). *GaussianCube: Structuring Gaussian Splatting using Optimal Transport for 3D Generative Modeling*. arXiv: 2403.19655 [cs.CV].

A Appendix

A.1 GitHub repositories

The code repositories for this project can be found at the following links. Each repository contains a README file detailing setup and operation of the code.

Dataset creation and model training: https://github.com/RyanS161/cutlist

Web-based interface: https://github.com/RyanS161/cutlist-web

A.2 AI Use Declaration

To ensure transparency and uphold academic integrity, the following table outlines the artificial intelligence (AI) tools utilized during the preparation of this report. This declaration aligns with ETH Zurich's guidelines on the responsible use of AI in scientific writing.

Table 5: AI Use Declaration

AI-Based Tool	Use Case	Scope	Remarks
Gemini 3.0 Pro	Code generation	Designer-QA Web application	Responsible for all front-end and much backend generation
GPT 5.0	Code generation	Dataset generation and training code	Some help used in debugging issues
Claude Opus 4.5	Code generation	Data labeling and figures code	Responsible for creating labeling platform and figures code