

3D Pose Estimation and Topology Reconstruction Using Foundation Models and Render and Compare



Adnaan Ali Sachidanandan

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

Jesus College

August 2023

Dedicated to my family, without whom this journey would not have been possible.

Declaration

I, Adnaan Ali Sachidanandan of Jesus College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed Adnaan Ali Sachidanandan

Date 17 August, 2023

Word Count: 14854

Software Declaration: Standard Python packages for machine learning applications were employed such as PyTorch, scikit-learn, and matplotlib.

Various software libraries and repositories were leveraged in this work:

- **Blender** (<https://www.blender.org/> - Blender Online Community (2018)) is used, without modification. We utilize the built-in libraries to control and render our CAD models.
- **Grounding DINO** (<https://github.com/IDEA-Research/GroundingDINO> - Liu et al. (2023c)) is used, without modification, to load and run the Grounding DINO model for open-set object detection.
- **Segment Anything** (<https://github.com/facebookresearch/segment-anything> - Kirillov et al. (2023)) is used, without modification, to load and run the Segment Anything model for image segmentation.
- **Surface Normal Uncertainty** (https://github.com/baegwangbin/surface_normal_uncertainty - Bae et al. (2021)) is used, without modification, to load and run the surface normal estimation model as part of our orientation tool.

- **Langer et al. (2021)** (https://github.com/florianlanger/leveraging_geometry_for_shape_estimation): We use their evaluation code as a starting point for computing AP^{mesh} scores for evaluation. The evaluation code is adapted to work with our system's outputs.
- **Pix3D** (<https://github.com/xingyuansun/pix3d> - Sun et al. (2018)): We use their Blender rendering demo code as a starting point for our system's code to render Pix3D CAD models at ground-truth positions and orientations.
- **OpenAI Python Library** (<https://github.com/openai/openai-python>) is used, without modification, to use the GPT-4 (OpenAI, 2023) API throughout our project.
- **Transformers Library** (<https://github.com/huggingface/transformers> - Wolf et al. (2020)) is used, without modification, to load and run various transformers models in our experiments.
- **Salesforce LAVIS** (<https://github.com/salesforce/LAVIS> - Li et al. (2023a)) is used, without modification, to load and run the InstructBLIP (Dai et al., 2023) model for our retrieval subsystem.

Coding was done on both the HPC cluster and Google Colab machines.

In our code, we leave comments with references to online resources when they were utilized as part of development.

Adnaan Ali Sachidanandan
August 2023

Acknowledgements

I would like to thank my supervisor Dr. Ignas Budvytis for his support and insights throughout this project. I have truly learned how to be a better researcher through our weekly discussions. I would additionally like to thank Florian Langer for sparing his time, valuable insights, and most importantly his support throughout this project.

I am grateful to have made some special friends in the MLMI program this year. Thank you for the wonderful (and challenging) experiences we have shared together. This year was so much better with your presence. From Darwin salsa to random trips to London, I will never forget these memories. I also want to thank Professor John Dudley for being an incredible course director and Anne Debenham for her endless support of our entire program.

I am lucky to have wonderful friends in Jesus College to spend time with throughout this year. *Roosties*, I am so grateful for all the memories we have had together, from Jack's runs to picnics in Jesus Green to long road trips at 4am to catch a flight on the other side of an island. You are all incredible, bright people who I will dearly miss when I return home. When I first arrived in Cambridge to start this degree, I was concerned if I would blend in, make friends, and struggle living so far from home. Thanks to you all, I instead had some of the greatest moments of my life.

A special thank you to Chris Rump, Jonas Scholz, Emma Prevot, Miguel Neves, Sergio Calvo Ordoñez, Alex Ellul, Lorenzo Bonito, Tony Wu, Ilaria Sartori, John Boom, Lateitia Pilgrim, and Patrick Hajali for all the time we spent together and for your friendship.

Another big thank you Hannah Lin, Isaac Akinduro, Sejal Karmarkar, Froher Yasin, Kieran Smith, James Cummins, and Ryota Nemoto for a legendary Park Street experience.

I want to also thank my childhood friends. Although I was on the other side of the world, we still found ways to spend time and connect, so I never felt too far from home.

Most importantly, I want to thank my mom, dad, and Ridhaa for everything. I would not have come this far without you.

Abstract

Estimation of shape and pose of 3D objects is one of the fundamental challenges in computer vision, with important uses in augmented reality, self-driving vehicles, and robotics. Classical deep-learning-based approaches like Mask2CAD and have tackled this problem by training neural networks on images and 3D data, but these methods are limited by the diversity of the data they train on and notably struggle when they encounter new, unseen objects. The recent development of Large Language Models (LLMs) like GPT-4 trained on massive corpora of data provides a new paradigm for solving these complex problems through their adaptability and robustness, effectively enabling modern systems to avoid the performance gap between training and testing. At the same time, currently accessible LLMs have not been trained on image data, making it difficult to use them for computer vision tasks.

In this work, we tackle topology reconstruction from monocular images with an intelligent system for CAD model retrieval and alignment. Our system combines GPT-4 with foundation models (e.g Segment Anything for segmentation and InstructBLIP for visual question answering) and the Render & Compare approach, where a system iteratively updates an object mesh, renders it, and compares it with a ground-truth image. We use Render and Compare due to its benefits as an extremely general task. Render and Compare has no limitations on how objects are rendered, how images are compared, and how meshes are updated, making our system completely flexible in implementation and robust to flaws in custom-trained methods. We use GPT-4 as an inference engine, using its latent knowledge of objects to extract properties for CAD retrieval and to write executable code orchestrating multiple tools to align CAD models to images. Inspired by Visual ChatGPT, we utilize pre-trained foundation models to gather visual information from images for GPT-4 to understand. We use Blender to update and render meshes.

We demonstrate that with the aforementioned tools, GPT-4 can apply the Render and Compare approach without being explicitly trained on 2D and 3D data. Our work shows that by giving GPT-4 access to foundation model tools that provide visual information like object pixel locations and normal vectors, our system can align objects without any training. We further find that by pairing the LLM with Visual Question Answering (VQA) models, we can automatically create datasets of CAD models with structured annotations for each object

with any human intervention. Using these datasets, our system is able to retrieve visually similar CAD models for an input image and achieve 36% top-1 retrieval accuracy, competing with state-of-the-art trained systems.

End-to-end, our system can also retrieve and align CAD models for images of tables without any training. Our system already has baseline performance on the Pix3D dataset, achieving a 0.4 average AP^{mesh} score, and demonstrates strong potential for competition with the state-of-the-art. Furthermore, unlike trained methods, our approach has no dependence on data and object categories making them more useful for real-world scenarios with numerous unexpected object categories. Therefore, we provide a starting point for object reconstruction using LLMs, and demonstrate that it is feasible to combine them with Render & Compare to perform 3D tasks in a human-like manner.

Table of contents

List of figures	x
List of tables	xvii
1 Introduction	1
1.1 Objective	1
1.2 Motivation	1
1.3 Challenges	3
1.4 Our approach	5
1.5 Contributions	6
1.6 Outline	7
2 Background	8
2.1 Topology Reconstruction	8
2.1.1 Classical Approaches to Reconstruction	8
2.2 LLMs and Foundation Models	12
2.2.1 LLMs	12
2.2.2 Foundation Models	13
3 Method	14
3.1 Overall System	14
3.2 CAD Model Retrieval	16
3.2.1 CAD Database	16
3.2.2 Auto-Annotation of Attributes	16
3.2.3 Retrieval from Auto-Annotated Data	19
3.3 CAD Model Alignment	21
3.3.1 Horizontal and Vertical Shifting	21
3.3.2 Depth Shifting	23
3.3.3 Rotation	24

3.3.4	Orchestrating the the tools and parts to Align	25
4	Experiment Design	28
4.1	Dataset	28
4.2	Pipeline	30
4.2.1	Retrieval Experiment Pipeline	30
4.2.2	Alignment Experiment Pipeline	34
4.2.3	End-to-End Experiment Pipeline	36
4.3	Evaluation Metrics	38
5	Results	40
5.1	CAD Retrieval using Foundation Models and LLMs	40
5.1.1	Classical Trained Methods on Unseen Data	40
5.1.2	LLM- and VQA-based Retrieval with Hand-Annotations	41
5.1.3	Improving Retrieval with Auto-Annotations	44
5.1.4	Extending to New Object Categories	46
5.2	CAD Alignment	46
5.2.1	Classical Methods on Unseen Images	46
5.2.2	Alignment with Foundation Model Tools	47
5.2.3	Improving Performance with Multiple Rotation Initializations	51
5.3	End-to-End Topology Retrieval	52
6	Conclusion and Future Work	55
6.1	Future Work	56
	References	58

List of figures

1.1	Our system takes in an image of an object as input. We output a CAD model, rotation matrix, and translation vector where the CAD model is aligned with the image when placed at the outputted rotation and translation, as visualized in the render on the right.	2
1.2	Real-world use-cases for CAD retrieval and alignment. Augmented reality (left) creates a virtual representation of the physical world (in this case a sofa), by finding a relevant CAD model and fitting it to the sofa’s location. Robots (middle) must estimate the rotation and position of objects like the coffee mug in order to interact with them. Self-driving vehicles (right) must estimate location and orientation of multiple objects in order to safely navigate. Examples like the image above from the Waymo Open Dataset (Sun et al., 2020) are used to train self-driving agents.	3
1.3	High-level diagram of our system. We first take an input image and use the <i>Object Annotator</i> to get object annotations for that image. We generate a CAD database by passing renders of each CAD model through the <i>Object Annotator</i> as well. The <i>CAD Retriever</i> takes the image annotations along with the database of CAD annotations and outputs the CAD models that best match the image. The selected CAD model is loaded and <i>Rendered</i> in Blender. The system then uses various visual foundation model tools to <i>Compare</i> the render to the input image and gather information like pixel shifts and surface normals to help with alignment. The system then plans and executes an <i>Update to the Mesh</i> , after which it is re-rendered and the loop continues until termination.	4

-
- 2.1 Diagram of Approach in Langer et al. (2021) for CAD retrieval and alignment. For retrieval, they train an encoder network to produce vector embeddings for each image. They use a triplet loss to force the embeddings of images and renders of the same CAD model closer together, while pushing away images and renders of other CAD models. For an input image, the system retrieves the CAD models whose renders have the closest (nearest-neighbor) embeddings to the image. Their system then aligns the CAD models to the image by computing keypoint correspondences between the image and CAD model, then jointly optimizing the 3D pose and shape based on those correspondences. While accurate on training data, this method is privy to losses in performance on unseen CAD models and images due to its trained networks. 10
- 3.1 Diagram of our entire system. Each key function is displayed in red, highlighting the flow of information from the input image to the final aligned CAD model. 15
- 3.2 Examples of table properties and annotations. Properties are in the left column and each annotation for the property is below its corresponding table image. Each property provides key information that can help our system filter through a database of different CAD models. 17
- 3.3 Diagram of auto-annotation for CAD models. GPT-4 is prompted to produce a set of properties to annotate. GPT-4 is then prompted to write code that will use InstructBLIP to answer questions about a CAD render. The code produces a set of annotations for the CAD model based on InstructBLIP's responses. 18
- 3.4 Diagram of CAD model retrieval with auto-annotated tables. GPT-4 is given the database of auto-annotated CAD models and is prompted to prepare a series of questions to ask a new input image. The system then asks each question to InstructBLIP with the input image and saves the list of answers. The answers are passed back into GPT-4 with the CAD database and a new prompt asking it to output the top-10 CAD models that correspond to the answers, along with confidence scores for each outputted CAD model. . . . 20
- 3.5 GPT-generated code to take pixel shifts and output Blender code that will shift the CAD model accordingly. The code directly applies Equations 3.1 and 3.2 and outputs a string of Blender code that modifies the object position. Note that depth is the y-position in Blender, hence `current_position[1]`. 23

-
- 3.6 GPT-generated code to take pixel widths and heights and output Blender code that will shift the CAD model’s depth accordingly. The code directly applies Equation 3.4, scales the horizontal and vertical positions based on the new depth, outputs a string of Blender code that modifies the object position. Note that depth is the y-position in Blender, hence `current_position[1]`. 24
- 3.7 GPT-generated code to take median surface normals and output Blender code that will rotate the CAD model to align the surface normals together. The code directly uses Blender’s built-in functions to compute and apply the rotation based on the normal vectors. 25
- 3.8 Descriptions of the functions built around each tool and code snippet. `get_shift()`, `get_width_height()`, and `get_normals()` are the tools we provide for extracting information for horizontal and vertical shifts (see Section 3.3.1), depth shifts (see Section 3.3.2), and rotations (see Section 3.3.3), respectively. `shift_from_pixel_diff()`, `shift_depth_from_width_height()`, and `rotate_from_normals()` are the functions generated by GPT-4 (see Figures 3.5, 3.6, and 3.7, respectively) that produce Blender code to apply modifications to the CAD model. `run_blender()` takes the outputs of these GPT-generated functions to execute the code in the Blender environment and re-render the CAD model. In combination, these methods enable GPT-4 to apply the Render and Compare loop from end-to-end. 26
- 3.9 GPT-generated code to orchestrate the tools and Blender code together for alignment. For each transformation in horizontal and vertical translations, depth translations, and rotations, the code first uses the corresponding information function, followed by the corresponding Blender code generation function, and finally the `run_blender()` function to execute the code. This demonstrates GPT-4’s ability to effectively combine these different functions to tackle the overall task of alignment. 27
- 4.1 Pix3D has a set of images for different categories. Each image has a corresponding CAD model in the dataset. The top row visualizes some Pix3D images and the bottom row visualizes their corresponding CAD models. Each image is annotated with the 3D position and rotation of the object in the image. 29
- 4.2 Hand annotations for a single CAD model. Each key is a property and each value/list of values is the annotations for the CAD model. 30
- 4.3 CAD models are rendered at a slight angle. This helps our system see more of the CAD model’s key features. 31

4.4	Prompt to GPT-4 to generate properties to annotate for each CAD model of <code>object_category</code>	32
4.5	Prompt to GPT-4 to generate executable code to annotate renders of CAD models.	32
4.6	Auto-annotations for a single CAD model. Each key is the question asked and each value/list of values is the annotations for the CAD model, which is the response generated by InstructBLIP.	32
4.7	Prompt to GPT-4 to generate questions to ask a new input image.	33
4.8	Prompt to GPT-4 to select its top-10 CAD models for a new input image, given the InstructBLIP-generated answers for each question planned by GPT-4. The question-answer pairs are stored in <code>qa_pairs</code> variable and the annotations for each CAD model in the database are stored in <code>annotated_cads</code>	33
4.9	For an input image, the CAD model is loaded with the ground-truth alignment. A random translation (top), rotation (middle), or both (bottom) are applied.	35
4.10	Renders for a single CAD model after each rotation initialization. Our system is run on each initialization, and the best result is selected by taking result with the largest mask IOU between the final render and the object in the input image.	37
5.1	Examples of retrieval with hand-annotations. The first column contains the input image. The second column is the question planned by GPT-4 for the attribute we visualize. The third column contains InstructBLIP’s response for the image and question, and the fourth and fifth columns are the hand-annotations for the ground-truth CAD model and system’s output CAD model, respectively. Due to the unreliability of InstructBLIP responses, the system will often collect annotations from input images that match incorrect CAD models, as exemplified by the top example. There are some responses, like in the second example, where ambiguity arises based on misinterpretations of the question. At the same time, the model does get some predictions correct, like in the third example.	42

-
- 5.2 Histogram of final predictions using hand-annotations (blue) in comparison to the distribution of ground-truth CAD models (orange) for \mathbb{S}_1 (top) and \mathbb{S}_2 (bottom). There is little-to-no overlap between the predictions and ground-truth data. This demonstrates that, after aggregating all the questions and answers for an input image, the system struggles to correctly accumulate information from an input image to match human-level annotations. This further demonstrates a bias towards certain CAD models, suggesting the InstructBLIP outputs may better match the hand-annotations of a specific subset of CAD models in the dataset. 43
- 5.3 Qualitative examples of hand-annotations vs auto-annotations. Whereas hand-annotations are perfect, auto-annotations make consistent mistakes for both the image and CAD models, resulting in the auto-annotations better matching annotations for new images. InstructBLIP often hallucinates 2 legs or 1 leg instead of 3 legs, 4 legs, or no legs at all. By auto-annotating, the database’s annotations include these hallucinations, resulting in GPT-4 planning questions around the incorrect data, leading to more consistency in predictions. In the second example, InstructBLIP hallucinates 1 drawer for every table. GPT-4 therefore does not ask a drawer quantity question and instead asks if any drawers are present or not, leading to fewer mismatches compared to the hand-annotations. In the third example, we can see a case where InstructBLIP is inconsistent with its responses on the same image. It correctly outputs "square" for the question from hand-annotated data but incorrectly outputs "rectangle" for the question from the auto-annotated data. This is a rare example where the hallucinations from InstructBLIP benefit the hand-annotated retrieval. 45

-
- 5.4 Examples of alignment after random translations. The first column contains the renders after the CAD models are randomly translated from the ground-truth position. The second column contains the target image. The third and fourth columns visualize the segmentation masks generated by the system to calculation horizontal, vertical, and depth shifts. In the final column we overlay the aligned CAD model with the target image. As demonstrated in the first and second examples, the system is able to translate objects to their target position, even when the target image is occluded by other objects. In the third and fourth examples, however, we see failure cases. The system struggles with objects that are partially out of frame and struggles to identify parts of tables alone. The system also often fails when multiple tables are in the target image, since it is not designed to distinguish between multiple tables for segmentation. 48
- 5.5 Examples of alignment after random rotations. The first column contains the renders after the CAD models are randomly translated from the ground-truth position. The second column contains the target image. The third and fourth columns visualize the surface normals generated by the system. In the final column we overlay the aligned CAD model with the target image. The system is able to align randomly rotated tables when the the object is only slightly rotated from its original position, like in the first and second examples. The system fails when the object is rotated too much, causing the primary visible surface to change. In such cases, the surface normals extracted from the render and target image are on different parts of the table, causing the system to align those parts together instead. The system also struggles with glass since the surface normal estimator and Blender treat them as transparent. Furthermore, as demonstrated in the final example, if the table is rotated around the axis perpendicular to the primary visible surface, the system cannot re-align the table fully. 50
- 5.6 Examples of alignment with and without multiple pre-rotations. By applying multiple possible pre-rotations, it is likely one pre-rotation will be close to the target rotation like in the examples above. This helps the system easily align CAD models to input images without changing its architecture, but comes with the cost of additional compute. 51

List of tables

5.1	Quantitative comparison of results on the \mathbb{S}_1 and \mathbb{S}_2 splits for CAD model retrieval using hand-annotated and auto-annotated data. Our system performs significantly better on auto-annotated data in comparison to hand-annotated data. Our system also notably outperforms competitors in the \mathbb{S}_2 split, a scenario where their networks are not trained on the CAD models, demonstrating our system’s strength on unseen data. <i>*Our results are for 50 random test images of tables, with 30 out of the 63 Pix3D tables as possible CAD models. Langer et al. (2021) is run on the entire dataset for all categories. We are unable to report Top-10 Accuracy for hand-annotations due to query limitations in the GPT-4 API.</i>	41
5.2	Quantitative comparison of retrieval results on table and bed images. Our system maintains strong performance in both categories, demonstrating that our system can easily adapt to new categories without any customization.	46
5.3	Quantitative results on the \mathbb{S}_1 and \mathbb{S}_2 split for re-aligning a table CAD model after an unknown transformation, in comparison to end-to-end results for competitors. Columns indicate the type of random transformation applied (combined refers to both transformation and rotation) or the name of the competitor. The system performs well on translation, but notably loses performance with rotation and the combined transformations. Our system also maintains a much larger portion of performance between \mathbb{S}_1 and \mathbb{S}_2 compared to other methods, even improving for translation. <i>*Note that our AP metrics are computed assuming perfect classification and CAD retrieval. Langer et al. (2021) results are taken from their end-to-end retrieval and alignment results. Mesh-RCNN results are also end-to-end. We pull Mesh-RCNN results from Langer et al. (2021) because the original paper only reports AP50 scores.</i>	47

-
- 5.4 Quantitative results on the S1 and S2 split using multiple pre-rotations vs no pre-rotations. Using the pre-rotations evidently improves performance significantly, with on average 3x improvements for every metric. 52
- 5.5 Quantitative results on the S1 and S2 split for end-to-end retrieval and alignment. After combining both parts of our system together, we achieve performance that demonstrates our system’s potential to compete with state-of-the-art models in future iterations. We highlight this potential by evaluating our model’s performance with ground-truth normals instead of the surface normal estimator. With ground-truth normals, our system approaches the performance of our competitors on the \mathbb{S}_2 split. Our system performs unusually worse on the \mathbb{S}_1 split, which we believe can be attributed to random chance since we sample around 12% of the available images in the dataset. **Again note that competitor results also include object category classification, whereas we assume perfect classification.* 52

Chapter 1

Introduction

1.1 Objective

The work in this thesis introduces a new paradigm for solving tasks in 3D Computer Vision. We specifically focus on the task of **3D topology recovery** as a **CAD model retrieval** and **mesh alignment** problem. These tasks are described in further detail below:

3D Topology Recovery focuses on recovering the high-level three-dimensional structure of an object from a two-dimensional image, including the location and orientation of the object in 3D and how its components relate in position and orientation to each other.

CAD model retrieval is the task of selecting the best CAD models from a database that match the object in an input image. This could include extracting the exact CAD model for an object in an image, or the most visually similar model if an exact match does not exist.

Mesh alignment seeks to shift and rotate (and potentially scale) meshes in 3D space to overlap with a corresponding object in an image when rendered.

With these descriptions in mind, our system's goal is to take an input image of an object, find a CAD model that best matches the object, and align the CAD model in 3D with the object in the image.

1.2 Motivation

Identifying objects in 3D is a fundamental part of how humans navigate and interact with the world. We develop this skill over time, seeing numerous examples and learning the



Fig. 1.1 Our system takes in an image of an object as input. We output a CAD model, rotation matrix, and translation vector where the CAD model is aligned with the image when placed at the outputted rotation and translation, as visualized in the render on the right.

core characteristics that identify objects like chairs and tables. Using our accumulated knowledge of object characteristics, we can easily process unseen objects and identify their corresponding classes. With the ability to identify objects, we position them in 3D to create a mental model of the world around us. This ability is crucial for planning what to do and how to interact with objects along the way. For example, how can you make a cup of coffee without identifying a coffee cup and determining its relative location and rotation to you? As a result, CAD retrieval and alignment is useful for many real world scenarios:

- **Augmented reality** is centered around virtually interacting with and modifying the world around you. It is therefore essential to computationally understand what objects are around the user and where they are located.
- **Robots** must be able to detect objects and determine their relative positions and orientations. Intelligent agents must know the 3D locations of objects to navigate towards or around them and must be able to detect their orientations to interact with them. For example, if an assembly robot is trying to fit a car door onto a chassis, it must know where the chassis is in 3D and how it is rotated in order to plan where to attach the door.
- In order to navigate and move around public roads, **autonomous vehicles** have to be robust to a variety of visual stimuli including pedestrians, other vehicles, signs, roadblocks, etc. Determining the location and velocities of these objects can be assisted with object retrieval and alignment, as position is essential and rotation can provide context for the directions they may move in.



Fig. 1.2 Real-world use-cases for CAD retrieval and alignment. Augmented reality (left) creates a virtual representation of the physical world (in this case a sofa), by finding a relevant CAD model and fitting it to the sofa's location. Robots (middle) must estimate the rotation and position of objects like the coffee mug in order to interact with them. Self-driving vehicles (right) must estimate location and orientation of multiple objects in order to safely navigate. Examples like the image above from the Waymo Open Dataset (Sun et al., 2020) are used to train self-driving agents.

1.3 Challenges

There are a variety of challenges in object reconstruction:

- **Visual variances:** Differences in lighting, pose, color, texture, and overall image content make CAD model retrieval a complex task. This leads to a common sim-to-real gap in performance for approaches that train a network to learn embedding vectors for images and use them for retrieval (Gümeli et al., 2022; Kuo et al., 2020; Langer et al., 2021).
- **Structure consistency and smoothness:** Methods that directly estimate a 3D mesh, like Mesh-RCNN (Gkioxari et al., 2019) struggle to produce smooth surfaces and structurally consistent meshes. These methods reconstruct the part of the object that faces the camera well, but they do not understand the objects well enough to extrapolate the rest of the shape that is unseen. CAD models avoid this issue, since they have a well-defined shape and surface.
- **Unexpected object types:** All trained methods can demonstrate strong performance on the types of objects they see in training data. In the real world, these systems will encounter completely new categories that they were not designed for. This causes mesh creation methods like Mesh-RCNN to struggle to create the object and CAD retrieval systems like Langer et al. (2021) to struggle to find relevant CAD models that match the object.

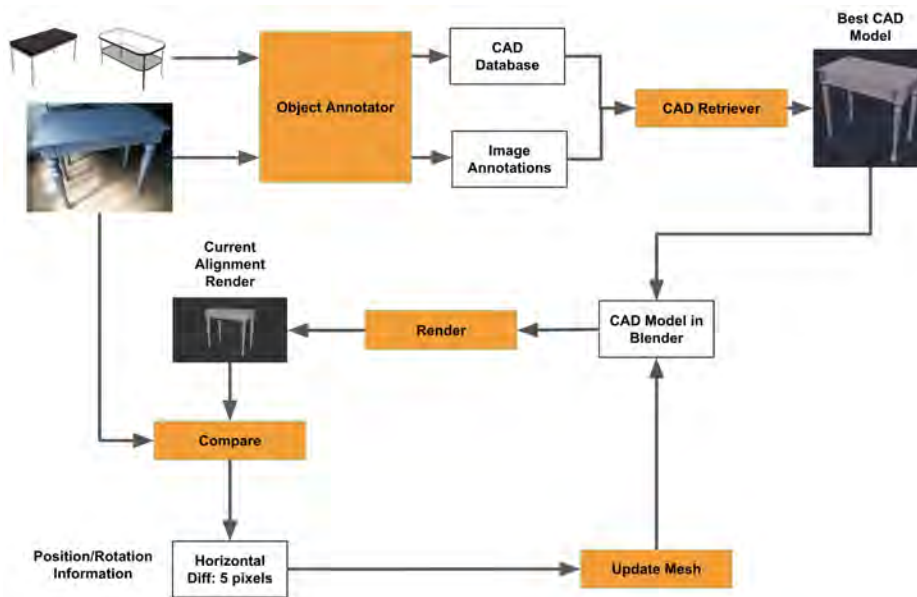


Fig. 1.3 High-level diagram of our system. We first take an input image and use the *Object Annotator* to get object annotations for that image. We generate a CAD database by passing renders of each CAD model through the *Object Annotator* as well. The *CAD Retriever* takes the image annotations along with the database of CAD annotations and outputs the CAD models that best match the image. The selected CAD model is loaded and *Rendered* in Blender. The system then uses various visual foundation model tools to *Compare* the render to the input image and gather information like pixel shifts and surface normals to help with alignment. The system then plans and executes an *Update to the Mesh*, after which it is re-rendered and the loop continues until termination.

1.4 Our approach

To retrieve and align CAD models to an image, our system gathers annotations about an object in an image using our *Object Annotator*. The system takes in an image of an object and uses a VQA model to collect key properties about the object in the image (e.g. number of legs) as annotations. The system then passes the input image’s annotations and the database of CAD model annotations into the *CAD Retriever*, which selects the top CAD models whose annotations match those of the input image. With a CAD model for the image, the system then undergoes the Render and Compare loop. The CAD model is loaded into Blender and rendered. The render and input image are passed into image-processing tools based on foundation models, which *Compare* the images and output information about differences in position and orientation of the objects in the two images. The system then uses GPT-4-written code to process these differences and *Update the Mesh* in Blender. The system then re-renders the object, continuing the loop until termination, after which it outputs the CAD model and its final rotation and translation.

- **Object Annotator:** To avoid the weaknesses of trained networks, we do not take the classical approach of representing an image or CAD model by a learned embedding vector. Instead, we leverage the general-purpose knowledge of GPT-4 (OpenAI, 2023) to suggest key attributes of the type of object we are annotating. We then use a InstructBLIP (Dai et al., 2023), a state-of-the-art VQA (Visual Question Answering) foundation model to answer questions about those attributes based on the input image or a render of the CAD model. GPT-4 plans the questions to ask based on the attributes it chose. The answers to the questions become the annotations for the image or CAD model.
- **CAD Retriever:** Since the annotations are purely text-based, we are unable to use traditional methods for retrieving CAD models like nearest-neighbors search over embedding vectors of images. Therefore, we use GPT-4 to directly filter our data. To do so, we pass in the annotations of the CAD models as well as the annotations of the input image as part of our prompt to GPT-4. We then task the LLM with outputting a list of the CAD models it believes best fit the annotations of the input image.
- **Compare Render and Image:** With the input image and a render of the CAD model in its current position and orientation, we use the *Compare* module to gather information related to the differences in 3D position and rotation between the object in the image and the CAD model. Inspired by Visual ChatGPT (Wu et al., 2023), we use state-of-the-art foundation models and neural networks (Bae et al., 2021; Kirillov et al., 2023;

Liu et al., 2023c) to gather pixel shifts between the two images for horizontal and vertical 3D shifts, differences in width and height for depth shifts, and differences in median surface normals for rotations.

- **Mesh Update:** This module takes the information produced by the compare module and applies transformations to the CAD model in Blender (Blender Online Community, 2018). The module is instantiated by querying GPT-4 to produce executable methods that take in pieces of information from the *Compare* module (e.g. pixel shifts, surface normals) and output Blender code that can be run on the object to update its 3D position and orientation. This module then runs the corresponding method created by GPT-4, inputting the information provided by the *Compare* module. The module then re-renders the CAD model to continue the Render & Compare loop.

Our approach is designed to tackle the challenges discussed in Section 1.3. By using foundation models for VQA and comparing images, we mitigate the issue of visual variances between images, since these networks are trained on a multitude of images that contain these different variations. We avoid issues in structure consistency and smoothness by aligning CAD models, which have predefined shapes that match real-world objects. We finally avoid the issue of unexpected object types by retrieving from a database of CAD models. In doing so, our system has the potential to handle new object types by gathering sets of CAD models from large online databases, as opposed to re-training networks for the new object types.

We evaluate our approach on CAD model retrieval, alignment, and end-to-end retrieval and alignment on the Pix3D dataset (Sun et al., 2018). Our system achieves achieves 36% top-1 and 77% top-10 retrieval accuracies on images of tables. Our system additionally maintains its performance on images of beds, achieving 34% top-1 and 74% top-10 accuracy. Our system achieves an average AP^{mesh} score of 1.2 in realigning CAD tables that have been randomly transformed. Our end-to-end system achieves a 0.4 average AP^{mesh} score which, while not a strong result, demonstrates that our approach has potential for strong performance in topology retrieval since it can be improved in many ways.

1.5 Contributions

In this thesis, we contribute the following:

- We develop a novel use of Render and Compare for topology reconstruction and alignment using LLMs as a knowledge agent by combining them with visual and multimodal foundation models. **We are the first to take this approach and demonstrate that it is a feasible direction for future research.**

- We design an intelligent new way to automatically annotate data for CAD model retrieval. This approach does not require any training and utilizes the latent knowledge of LLMs, enabling us to produce textual annotations that humans can understand. It is also easily adaptable to new object categories out-of-the-box (see Section 5.1.4).
- We introduce a framework for joint retrieval and alignment using zero model training that achieves commendable results on a popular dataset. Note we provide an exciting new baseline for future research into LLMs and foundation models for computer vision tasks. Our work can be expanded and improved upon with analysis on more test images, other datasets, and stronger foundation models and LLMs that improve performance.

1.6 Outline

This thesis is divided into 6 chapters. **Chapter 2** discusses other works with topical relevance to this project. **Chapter 3** discusses our method in depth, including how we annotate images and each CAD model, how we use GPT-4 to retrieve CAD models for an image, what models and tools we use to extract information for translations and rotations, and how we actually apply those translations and rotations using GPT-4-produced code and Blender. **Chapter 4** discusses the details of our experiments and additional implementation details. **Chapter 5** explores the results of our experiments and the insights they provide about our system. We conclude with **Chapter 6**, summarizing our project's contributions and discussing exciting directions for future work.

Chapter 2

Background

This section discusses related works that contextualize our approach in this project. We break this section down into the key topics of this project. Section 2.1 discusses existing works in topology reconstruction, providing background on the different methods researchers have used to tackle this problem. Section 2.2 discusses LLMs and foundation models, providing a brief overview of what they are and why they benefit our system.

2.1 Topology Reconstruction

Topology reconstruction focuses on recovering the spatial structure of objects. In the context of this project, topology reconstruction involves gathering a 3D representation of an object in an image such that its surface is consistent with the overall structure of the image's object. For example, chair armrests may have holes in them that need to be accurately modeled in the object's structure. In this section, we discuss the variety of methods that try to recover an object's topology from an image. We split this discussion into classical computer vision methods and recent diffusion-based methods that attempt to solve this problem.

2.1.1 Classical Approaches to Reconstruction

Recent classical reconstruction methods use network training and loss optimization to reconstruct from images. These approaches include training end-to-end networks that directly regress 3D shapes, applying iterative loops to refine meshes to an image, and applying diffusion methods to multi-view reconstruction methods.

End-to-End Networks

Many approaches use a trained network architecture from start to finish. Some systems directly predict voxel representations, where an object is encoded by a three-dimensional grid of cubes (Liu and Liu, 2021; Shi et al., 2021). These methods suffer from the tradeoff of voxel representations, where low-density voxel grids have cubic surfaces but high-density voxel grids have high computation requirements.

Graph convolution-based methods directly predict a fitted mesh for the object. These methods start with an initial mesh of the object and iteratively refine them through graph convolutions to better fit the input image (Gkioxari et al., 2019; Wang et al., 2018). Pixel2Mesh (Wang et al., 2018) starts with an ellipsoid as its initial mesh and refines from there, but as a result suffers from weak topology due to its inability to break the ellipsoid down into more difficult structures. Mesh-RCNN (Gkioxari et al., 2019) avoids this issue by first predicting a voxel grid with a trained network, converting it to a mesh, and then refining the mesh using graph convolutions. While this method better captures the topology of objects, it still suffers from rough surfaces and inconsistencies for parts of the object that are not directly visible in the image.

Other methods reconstruct an object as a Signed Distance Function, where the object is represented as a function that outputs the distance to its surface from an input 3D point (Jiang et al., 2020; Park et al., 2019). This function simplifies learning complex topologies, but makes in-context reconstruction from images more difficult, hence their focus on reconstructing topology from point clouds or synthetic data. Again, we avoid fully trained end-to-end networks due to their inability to maintain performance on unseen data as well as their inability to output smooth and consistent surfaces.

CAD Model Retrieval and Alignment

A large class of approaches retrieve corresponding CAD models and align them to the input image. More specifically, these methods select a CAD model that best matches the object in an image out of a database of other CAD models. They then estimate the 3D location and rotation of the object in the image. With these outputs, a system can render the CAD model at the location and rotation to spatially overlap with the object in the image. State-of-the-art classical methods retrieve the best CAD model by using learned image embeddings. These approaches pass the input image and renders of the CAD models into a network that outputs an image embedding.

IM2CAD chooses the CAD model whose render has the highest cosine similarity to the input image. The system then aligns the CAD model to the image by numerically optimizing

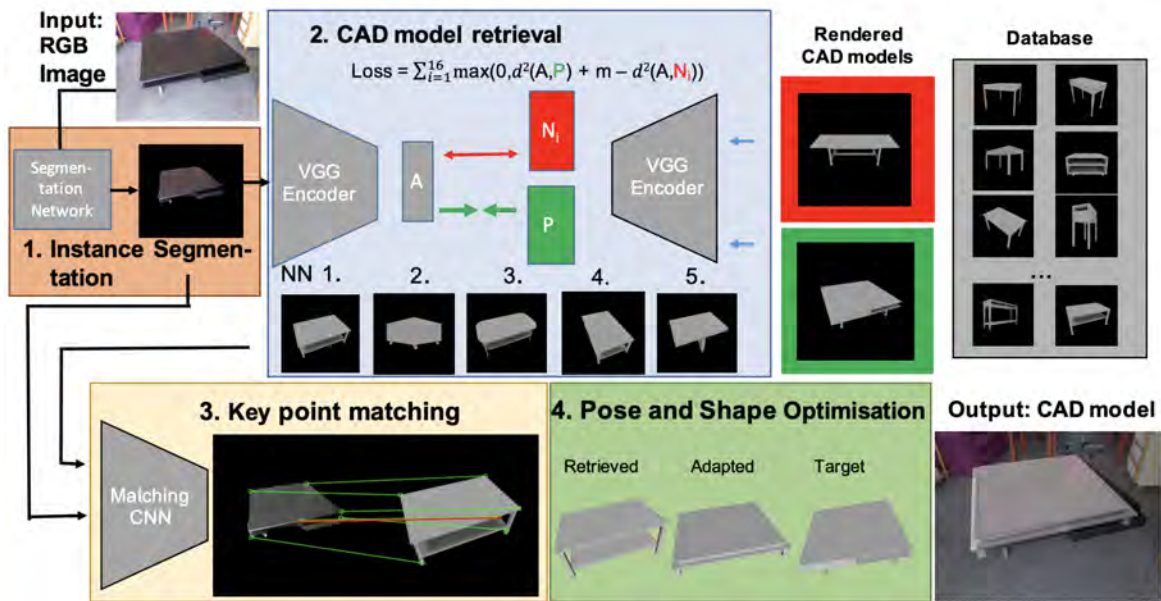


Fig. 2.1 Diagram of Approach in Langer et al. (2021) for CAD retrieval and alignment. For retrieval, they train an encoder network to produce vector embeddings for each image. They use a triplet loss to force the embeddings of images and renders of the same CAD model closer together, while pushing away images and renders of other CAD models. For an input image, the system retrieves the CAD models whose renders have the closest (nearest-neighbor) embeddings to the image. Their system then aligns the CAD models to the image by computing keypoint correspondences between the image and CAD model, then jointly optimizing the 3D pose and shape based on those correspondences. While accurate on training data, this method is privity to losses in performance on unseen CAD models and images due to its trained networks.

the cosine similarity between the rendered scene and input image in terms of the position and rotation of the object (Izadinia et al., 2017). Mask2CAD learns a joint CNN embedding space between renders of CAD models and the regions of images that contain objects. The CAD model’s translation and rotation in the image is estimated by using a combination of Huber loss and extra trained network layers on the image features (Kuo et al., 2020). Patch2CAD takes this further by applying the same approach as Mask2CAD but computes embeddings on patches of the images and renders them instead of the entire images. Langer et al. (2021) learns an embedding space over the entire image and renders. It aligns a CAD model to the image by finding correspondence points between the input image and the 3D mesh of the CAD model and then computing the translation and rotation as a PnP problem. They further improve performance by stretching the CAD models along different 3D planes to better fit input images. Their approach is visualized in Figure 2.1. ROCA (Gümeli et al., 2022) also retrieves CAD models through a joint embedding space, but to align CAD models it uses a combination of image to 3D correspondences, depth estimation, and differentiable Procrustes optimization over the rotation and translation. Each of these methods provide strong performance on datasets they are trained on like Pix3D (Sun et al., 2018) and ShapeNet (Chang et al., 2015), but due to their reliance on classical trained networks, they struggle with the Sim2Real gap, losing performance on unseen test data and real-world images. Like end-to-end trained methods, these approaches still suffer from losses in performance on unseen data, hence we leverage their strength in using CAD models while avoiding manually training networks.

Multi-View Reconstruction with Diffusion Models

A few recent methods have taken advantage of the powerful capabilities of view-conditioned diffusion models, networks that are capable of producing novel views of scenes by conditioning on existing images of the scene and the new viewing angle (Liu et al., 2023b; Rombach et al., 2021; Zhou and Tulsiani, 2023). These systems use view-conditioned diffusion models to generate novel views of an object and apply the generated views to multi-view 3D reconstruction methods. To preface, a key multi-view reconstruction method is Neural Radiance Fields (NeRFs) (Mildenhall et al., 2020). NeRFs represent a scene through a learned function that maps input viewing angles and 3D positions to radiance/color and occupancy at that position. This function therefore encodes a 3D representation of the scene which can be used to easily compute depth maps and therefore other 3D representations like meshes. Multiple methods use variants of NeRFs as well as Instant NGP (Müller et al., 2022), a heavily optimized extension of NeRFs, as their multi-view reconstruction system and learn the 3D shape by generating view-conditioned novel view images using diffusion

models, alongside various other system optimizations (Deng et al., 2022; Liu et al., 2023b; Melas-Kyriazi et al., 2023; Zhou and Tulsiani, 2023). One-2-3-45 (Liu et al., 2023a) is an alternative method that uses a view-conditioned diffusion model with an SDF-based neural reconstruction system, enabling it to quickly produce full meshes of an object from an image. While these diffusion-based approaches do not lose performance on unseen data, they still struggle to produce sharp surfaces so we continue with CAD models due to their strict and accurate surfaces and shapes.

2.2 LLMs and Foundation Models

Our approach relies on the inferential capabilities of LLMs and the zero-shot performance of visual and multimodal foundation models. We provide a brief background on both types of models below, leading to the final models we use in our system.

2.2.1 LLMs

Large Language Models are large Transformer networks (Vaswani et al., 2017) trained on huge corpora of data in order to effectively predict text as a continuation of some input text. BERT (Devlin et al., 2018) introduced an approach to training transformers on purely text data, sparking a turning point where new models could be trained on data scraped from the internet. Radford et al. (2018) proposed the idea of large-scale pre-training of language models on text data followed by fine-tuning on datasets for specific tasks. Wei et al. (2021) demonstrated that by fine-tuning LLMs on a variety of instruction-based datasets, they improve on new, unseen tasks. GPT-3 (Brown et al., 2020) crucially found that LLMs with large enough scale have the emergent property of adapting to new tasks with a single prompt, sparking few-shot instruction fine-tuning and new applications of multi-billion-parameter LLMs.

InstructGPT, also known as GPT-3.5, (Ouyang et al., 2022) demonstrated the conversational capabilities of LLMs as well as their ability for complex reasoning by training them using reinforcement learning from human feedback. LLaMa (Touvron et al., 2023a) and Vicuna (Chiang et al., 2023) are recent LLMs that have been open-sourced to the public and have demonstrated performance competitive with popular closed-source LLMs like GPT-3.5 and Claude (Anthropic AI, 2023). GPT-4 (OpenAI, 2023) is OpenAI’s latest and largest LLM which has demonstrated unprecedented capabilities in complex reasoning, code writing, and general task completion. LLaMa 2, on the other hand, is a recent human-centered model focused around AI safety and red-teaming, sparking a new direction of responsible LLM

development (Touvron et al., 2023b). At the same time, LLaMa 2 lacks the inferential strength of GPT-4, so we use GPT-4 in our work.

2.2.2 Foundation Models

Foundation models are large networks that are trained on a massive amount of unlabeled data (Bommasani et al., 2022). LLMs are one such type of foundation model. In this section we focus on visual and multimodal foundation models relevant to the topic of this project.

Segment Anything (Kirillov et al., 2023) is a foundation model designed to segment any object. Trained on a diverse set of curated images and segmentations, the model is able to take in points or bounding boxes as prompts and output segmentation boxes for almost any object with its zero-shot capabilities. By combining this model with Grounding DINO, an open-set object detector, (Liu et al., 2023c), one can create an open-set object segmentation system, which we heavily use in our work.

Multimodal foundation models are foundation models that can handle multiple forms of data, like images and text. CLIP (Radford et al., 2021) introduced an efficient pre-training task for models to learn visual-text relations, which served as a key step in multimodal foundation models. CLIP uses large datasets of image-text pairs to learn image-text relations by contrastive pre-training a network to select the best caption for an image. This pre-training enables models to learn key properties of image-text relations, so those models can be fine-tuned on new tasks for zero-shot transfer of their emergent knowledge. CLIP relies on a curated dataset to learn, but ALIGN (Jia et al., 2021) expands on CLIP for multimodal contrastive learning on noisy, imperfect datasets.

Flamingo (Alayrac et al., 2022) and BLIP-2 (Li et al., 2023b) are key approaches that combine pretrained vision encoders and language models by inserting and training network architectures between them to bridge the image-text gap. These models achieve strong performance on a variety of different datasets for different multimodal tasks. InstructBLIP (Dai et al., 2023), the successor of BLIP-2, expands on its predecessor by introducing instruction-tuning to the network. It adds an instruction-aware transformer between its vision and text models, enabling task-specific tuning. By training on 13 different vision-language task datasets, InstructBLIP achieves state-of-the-art performance on multiple tasks including VQA. As a result, we use InstructBLIP for VQA in our system.

Chapter 3

Method

This chapter discusses the key details of our approach in this work. We tackle topology reconstruction through two stages: CAD model retrieval and CAD model alignment. In CAD model retrieval, we filter through a database of annotated CAD models to find the models that best match the object in an input image. CAD model alignment then takes the CAD model and rotates and translates it in 3D to have the same position and orientation as the object in the input image. Our approach to both of these tasks uniquely uses pre-trained LLMs and foundation models instead of manually trained networks for specific subtasks.

3.1 Overall System

Our full system is visualized in Figure 3.1. We begin by annotating the CAD models. We use GPT-4 (OpenAI, 2023) to plan questions to ask for each image (input and CAD model renders). These questions are passed into InstructBLIP (Dai et al., 2023) and the answers become the annotations for the image and CAD models, respectively. We pass the annotations back into GPT-4 and task the LLM to select the CAD model(s) that best fit the image’s annotations. This completes the retrieval portion of our system, described further in Section 3.2.

Once a CAD model is selected, it is loaded into Blender (Blender Online Community, 2018) and rendered. GPT-4 is queried to produce executable code that will align the CAD model to the input image using the foundation model tools that are available. The code applies an iterative loop, where the 3D model is rendered, compared to the image using the foundation model tools, and updated in Blender. Once the iterations are complete, the system outputs the aligned CAD model ¹. More details can be found in Section 3.3.

¹The aligned CAD model consists of the CAD model, its final rotation matrix, and its final translation matrix

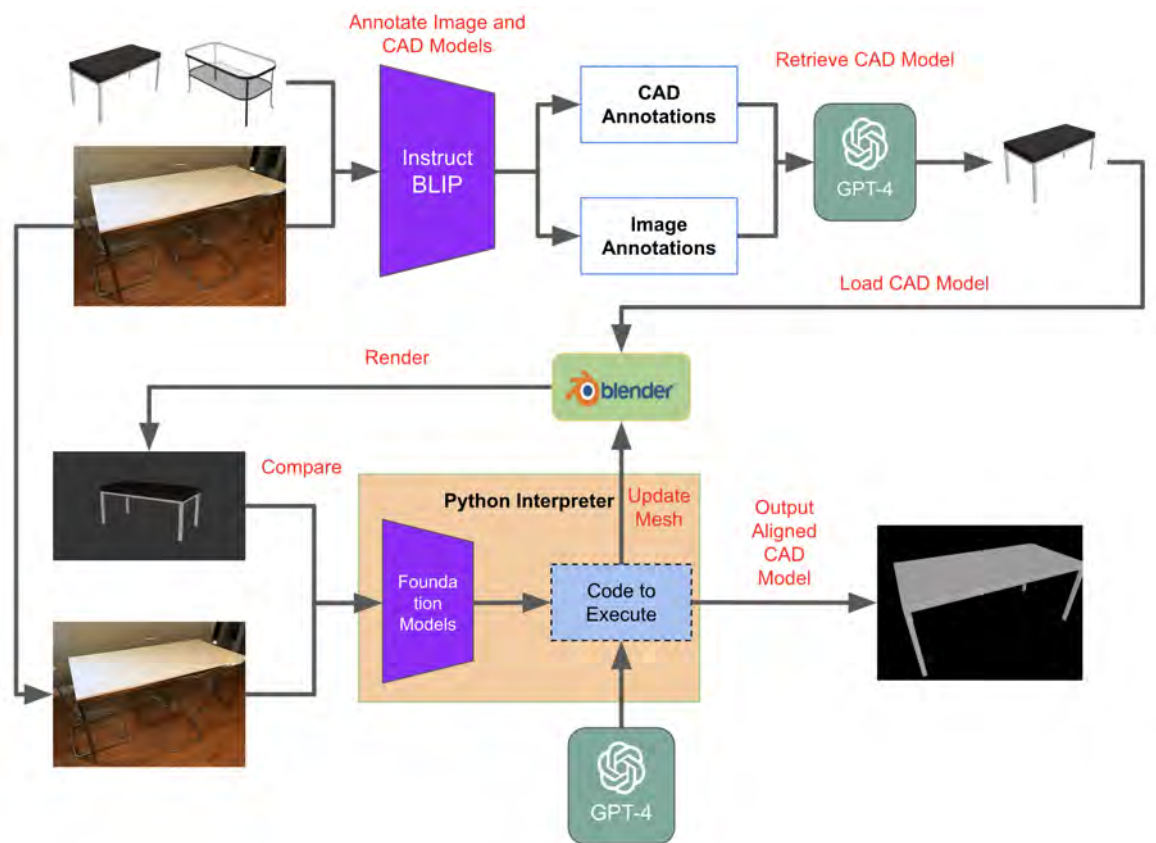


Fig. 3.1 Diagram of our entire system. Each key function is displayed in red, highlighting the flow of information from the input image to the final aligned CAD model.

3.2 CAD Model Retrieval

The first key stage in our system is CAD model retrieval. In this subtask, our system must find the best CAD model that matches an object in an image. In our approach, we construct a database of the available CAD models, gathering textual annotations of different properties of the objects using InstructBLIP. We then pass these annotations as well as the annotations for the input image into GPT-4 to select the best CAD models. The first subsection describes the CAD database and what types of annotations are collected. The second subsection discusses our process for gathering these annotations automatically. The third and final subsection explains how we use the CAD database and annotations of the input image to finally select a corresponding CAD model.

3.2.1 CAD Database

To retrieve CAD models, classical methods (Kuo et al., 2020; Langer et al., 2021) employ learned networks that compute high-dimensional embedding vectors for any image or CAD model render. Our work must crucially avoid relying on a network that is trained for this specific task². Therefore, we encode details of each CAD model in a format that any LLM and system can understand: text.

Each CAD model is annotated for a set of key properties. These properties encode semantic information about the CAD model that can distinguish it from other possible CAD models. One such property is "number of table legs". The number of legs on a table is a unique property that every table has, and furthermore it is an important property to use when finding similar tables and filtering out dissimilar tables. Such properties are beneficial because they are easy to understand and can be directly passed into any language model without any modification, unlike an embedding vector or other complex structure. Examples of additional properties are provided in Figure 3.2.

3.2.2 Auto-Annotation of Attributes

To gather annotations for each CAD model in the database, we use GPT-4 (OpenAI, 2023) paired with InstructBLIP (Dai et al., 2023), a state-of-the-art Visual Question Answering (VQA) foundation model. The data annotation is completed in four steps:

1. Each CAD model is rendered as an image (see Section 4.2.1 for details).

²These networks lose significant performance on unseen CAD models and images. See Section 5.1.1.



Fig. 3.2 Examples of table properties and annotations. Properties are in the left column and each annotation for the property is below its corresponding table image. Each property provides key information that can help our system filter through a database of different CAD models.

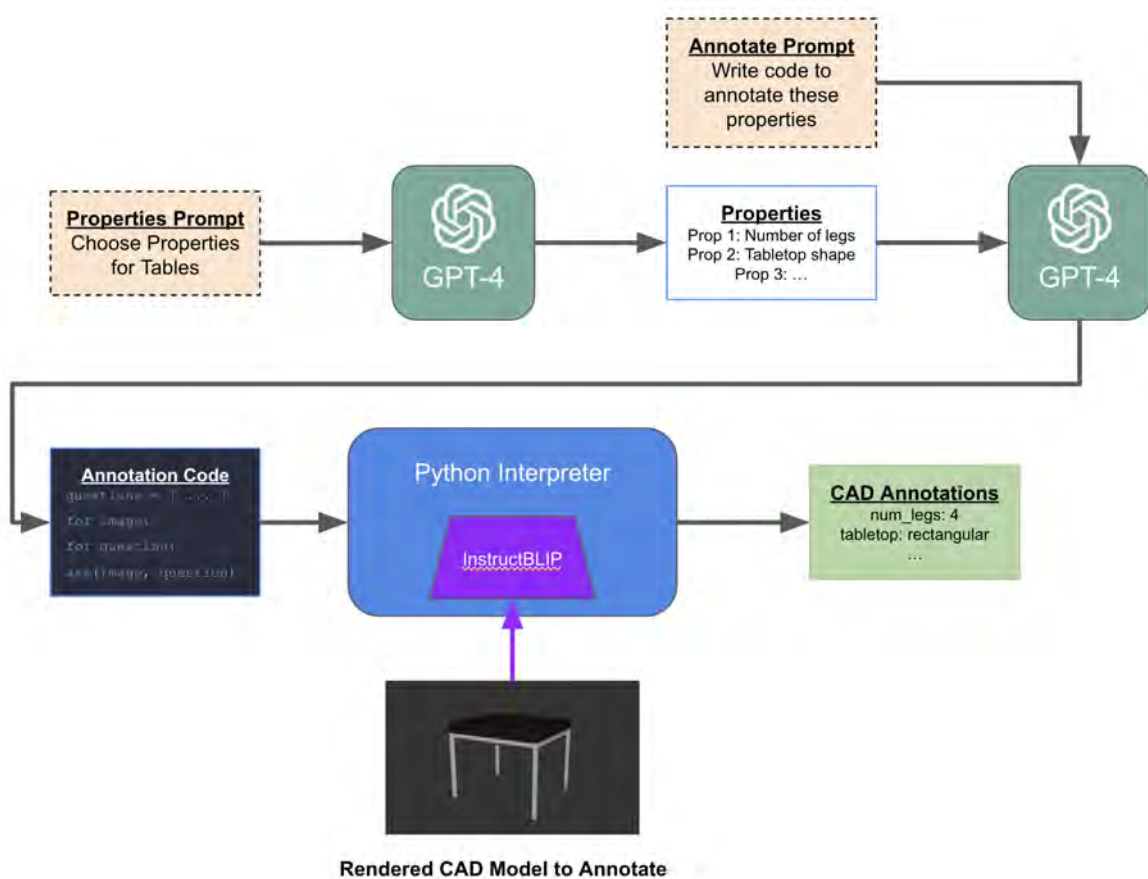


Fig. 3.3 Diagram of auto-annotation for CAD models. GPT-4 is prompted to produce a set of properties to annotate. GPT-4 is then prompted to write code that will use InstructBLIP to answer questions about a CAD render. The code produces a set of annotations for the CAD model based on InstructBLIP's responses.

2. GPT-4 is told it will annotate a set CAD models and chooses a series of properties it will annotate.
3. GPT-4 is then given access to InstructBLIP for VQA through an `ask()` function. The LLM writes a snippet of code which will do the following:
 - Create a data entry for each CAD model.
 - Ask a question about each aforementioned property to the VQA network, with the render of the CAD model as input.
 - Record the answer to each question as an annotation for the corresponding property.
4. The code snippet is executed on the list of CAD model renders.

This process is visualized in Figure 3.3. For full details on prompting for GPT-4, see Section 4.2.1.

Through this auto-annotation system, we automatically annotate each CAD model without any need for training data. This approach uses GPT-4’s latent knowledge and the robustness of the InstructBLIP foundation model such that it can annotate any object of any category without any modification. As a result, our system can easily adapt to an ever-growing database of CAD models and object categories, unlike classical methods that train a network to produce embedding vectors. In addition, this system is modular by design, so any subsequent LLM or multimodal foundation model for VQA can be directly swapped with GPT-4 and InstructBLIP respectively to improve performance. We note that we additionally evaluate our retrieval system with hand-annotated CAD models as an alternative to auto-annotations and find performance to be significantly worse. See Section 5.1.2 for further details.

3.2.3 Retrieval from Auto-Annotated Data

With the database of annotated CAD models, we retrieve the best-match CAD models using a similar approach to auto-annotation. We use GPT-4 (OpenAI, 2023) to process the annotated tables and produce a list of questions to ask InstructBLIP (Dai et al., 2023) for an input image with an unknown object. The outputted questions are then passed one-by-one into InstructBLIP with the input image to get the annotations for the unknown object. These annotations are finally passed back into GPT-4 alongside the original database of auto-annotated CAD models. With the annotations and database, GPT-4 is given a new prompt to output the 10 CAD models in the database that it believes best match the unknown object’s

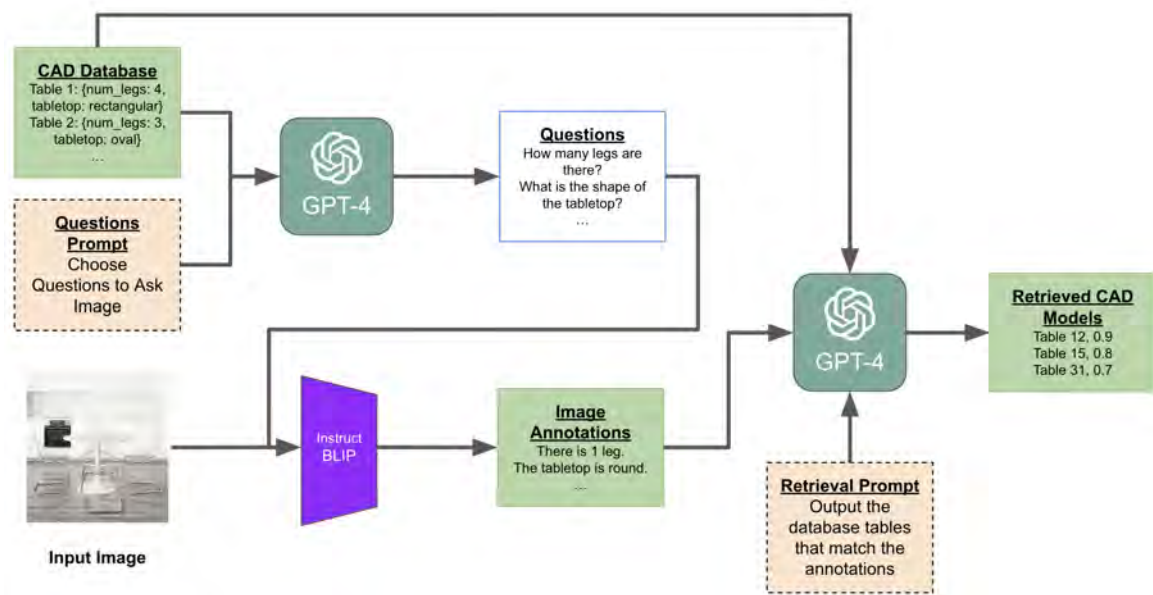


Fig. 3.4 Diagram of CAD model retrieval with auto-annotated tables. GPT-4 is given the database of auto-annotated CAD models and is prompted to prepare a series of questions to ask a new input image. The system then asks each question to InstructBLIP with the input image and saves the list of answers. The answers are passed back into GPT-4 with the CAD database and a new prompt asking it to output the top-10 CAD models that correspond to the answers, along with confidence scores for each outputted CAD model.

annotations. GPT-4 is also asked to output a confidence score for each of the 10 CAD models, enabling a sequential ranking of the models. This process is visualized in Figure 3.4.

Like our auto-annotation system, this retrieval system does not require any training or fine-tuning of models. By design, GPT-4 plans every question we ask to a new image, and we do not apply any processing to the CAD database. This makes the system fully adaptable to new CAD models, object categories, and annotation types. Furthermore, the system can be easily upgraded with more intelligent LLMs and multimodal foundation models in the future by simply replacing GPT-4 and InstructBLIP.

3.3 CAD Model Alignment

To align a CAD model with an object in an image, we take an iterative, two-step approach that follows the Render and Compare paradigm. To initialize, we load the CAD model into Blender (Blender Online Community, 2018) and render it in its current position. We first extract information from the image and render using tools built around visual foundation models and neural networks. We use the tools to **compare** visual information between the image (goal) and render (current state). Second, we use GPT-4 (OpenAI, 2023) to write code that will update the CAD mesh in Blender based on the extracted information and **render** the updated CAD model. We repeat these two steps for a set number of iterations. We group this section into horizontal and vertical shifting, depth shifting, and rotation. For each transformation, we discuss the tools used to gather key visual information and the code that is generated to apply the transformation to the CAD model in Blender.

3.3.1 Horizontal and Vertical Shifting

In this subsection we discuss the tools our system uses and the code it generates to detect and fix horizontal and vertical shifts of objects between two images.

Extracting Information for Horizontal and Vertical Shifts

To detect horizontal and vertical shifts, we estimate the center point of the object in the input image and current render, and then compute the pixel difference between the two center points. This gives a horizontal and vertical delta between the current position in the render and the target position in the input image. Our system detects the center point of the object using Grounding DINO (Liu et al., 2023c) and Segment Anything (Kirillov et al., 2023). By combining these two models together, we have a strong system for producing segmentation

masks from textual inputs³, enabling us to get the 2-dimensional attributes of the object we are trying to align in both the input image and the render of the CAD model in its current state.

For horizontal and vertical shifts specifically, we apply a simple operation on the segmentation mask by gathering the minimum and maximum x and y coordinates and taking the average as the center point of the object. We then compute the shifts by subtracting the object’s center point in the current render from its center in the input image, providing a delta towards the target location.

Applying Horizontal and Vertical Shifts

In order to generate code to apply the horizontal and vertical shifts in 3D, we apply a custom prompt to GPT-4 providing key information. The prompt informs GPT-4 that an object has been shifted an unknown amount horizontally and vertically in 3D and that it must undo that shift. The LLM is informed that it must produce a method that takes in the horizontal pixel shift, vertical pixel shift, and current 3D position of the CAD model and outputs a string of Blender Python code that will transform the object back to its original position. The LLM is additionally told it has access to the camera parameters, including focal length, sensor width and height, and image resolutions, which it must use to convert the pixel shifts to 3D.

GPT-4 extrapolates from this information to produce the following equations in Python code,

$$\Delta x = \frac{s_w}{r_x} * \frac{z}{f} * h \quad (3.1)$$

$$\Delta y = \frac{s_h}{r_y} * \frac{z}{f} * v \quad (3.2)$$

where h and v are the horizontal and vertical pixel shifts, respectively, and x , y , and z are the horizontal, vertical, and depth positions, respectively. s_w and s_h are the sensor width and height, r_x and r_y are the horizontal and vertical image resolutions, and f is the focal length of the camera. These equations are derived from Equation 3.3, the camera-to-world formula that converts pixel position p_x to 3D position x at depth z (and p_y to position y WLOG).

$$x = \frac{s_w}{r_x} * \frac{z}{f} * p_x \quad (3.3)$$

Following this computation, the GPT-produced code applies the shift to the object’s 3D position in Blender. Since we prompt GPT-4 to output a Python method, it can be called with

³See Section 2.2.2.

```

1 def shift_from_pixel_diff(horizontal_pixel_shift, vertical_pixel_shift, current_position):
2     # Convert pixel shifts to 3D shifts
3     horizontal_shift = (horizontal_pixel_shift * current_position[1] * sensor_width) / (focal_length * resolution_x)
4     vertical_shift = (vertical_pixel_shift * current_position[1] * sensor_height) / (focal_length * resolution_y)
5
6     # Subtract shifts from current position to get original position
7     original_position = [current_position[0] - horizontal_shift, current_position[1], current_position[2] - vertical_shift]
8
9     # Return Blender code to shift object back to original position
10    return 'obj.location = ' + str(original_position)

```

Fig. 3.5 GPT-generated code to take pixel shifts and output Blender code that will shift the CAD model accordingly. The code directly applies Equations 3.1 and 3.2 and outputs a string of Blender code that modifies the object position. Note that depth is the y-position in Blender, hence `current_position[1]`.

each pixel shift without prompting GPT-4 again, making the system more time-efficient and less costly. An example of GPT-4’s output is provided in Figure 3.5. This approach further demonstrates the impressive intelligence of LLMs and their potential as an agent that can directly modify objects in 3D.

3.3.2 Depth Shifting

In this subsection we discuss the tools our system uses and the code it generates to detect and fix differences in object depth between two images.

Extracting Information for Depth Shifts

For differences in depth, we estimate the width and height of the object in the input image and current render as surrogate measures of depth. Our system again utilizes the masks generated from Grounding DINO and Segment Anything, as discussed in 3.3.1.

From the segmentation masks, we compute the minimum and maximum horizontal and vertical pixel values. We then compute the width and height of the objects in the images as the difference between the maximum and minimum values.

Applying Depth Shifts

We again generate code by prompting GPT-4. Since the conversion from width and height to depth is more complex, we explicitly provide the following relation derived from the triangulation formulas of an object with fixed dimensions that is shifted in depth:

$$d_o = \frac{w_c}{w_o} d_c \quad (3.4)$$

```

11 def shift_depth_from_width_height(original_width, original_height, current_width, current_height, current_position):
12     # Calculate the depth from the width and height
13     original_depth_width = (current_width * current_position[1]) / original_width
14     original_depth_height = (current_height * current_position[1]) / original_height
15     # Average the depth
16     original_depth = (original_depth_width + original_depth_height) / 2
17     # Calculate the depth ratio
18     depth_ratio = original_depth / current_position[1]
19     # Scale the horizontal and vertical positions
20     original_position_x = current_position[0] * depth_ratio
21     original_position_z = current_position[2] * depth_ratio
22     # Return the Blender code
23     return f'obj.location = ({original_position_x}, {original_depth}, {original_position_z})'

```

Fig. 3.6 GPT-generated code to take pixel widths and heights and output Blender code that will shift the CAD model’s depth accordingly. The code directly applies Equation 3.4, scales the horizontal and vertical positions based on the new depth, outputs a string of Blender code that modifies the object position. Note that depth is the y-position in Blender, hence `current_position[1]`.

where d_o is the original depth, d_c is the current depth, w_o is the object width in the original (pre-transformation) image and w_c is the width in the current render. The same equation applies when w_o and w_c are replaced with h_o and h_c , the object heights in the original image and current render.

Our system’s prompt to GPT-4 provides the relation in Equation 3.4 and instructs it to produce code that shifts the CAD model’s depth in Blender. We further instruct GPT-4 to compute the original depth in terms of width and height separately, and take the average of the two as the updated depth of the object. We additionally tell the LLM to scale the x and y positions of the CAD model based on the ratio of the original depth to the current depth. This is derived from Equation 3.3 when z is changed to a new value. The outputted GPT-4 code is provided in Figure 3.6.

This approach highlights the limitations of GPT-4, as it cannot apply the complex multi-step reasoning to derive each property needed for converting widths and heights to depth shifts, but it still demonstrates the ability of the model to take key properties and convert it into executable code in Blender.

3.3.3 Rotation

In this subsection we discuss the tools our system uses and the code it generates to align objects with different orientations between two images.

Extracting Information for Rotations

We gather rotation information through the surface normals of an object, which encode the overall orientation of the object in a single vector. To extract this information, we apply a

```

24 def rotate_from_normals(original_normal, current_normal):
25     blender_code = """
26     import bpy
27     import mathutils
28
29     # Convert the input normals to Blender Vector objects
30     original_normal = mathutils.Vector({{}})
31     current_normal = mathutils.Vector({{}})
32
33     # Calculate the rotation difference between the original and current normals
34     rotation_difference = current_normal.rotation_difference(original_normal)
35
36     # Convert the rotation difference to Euler rotation
37     euler_rotation = rotation_difference.to_euler('XYZ')
38
39     # Apply the Euler rotation to the object's rotation_euler attribute
40     obj.rotation_euler.rotate(euler_rotation)
41     """.format(original_normal, current_normal)
42
43     return blender_code

```

Fig. 3.7 GPT-generated code to take median surface normals and output Blender code that will rotate the CAD model to align the surface normals together. The code directly uses Blender’s built-in functions to compute and apply the rotation based on the normal vectors.

surface normal estimator (Bae et al., 2022) to get per-pixel surface normals in the input image. For the current render of the 3D object, we use Blender directly to provide ground-truth surface normals of the CAD model. Since these surface normals are per-pixel, we use the combination of Grounding DINO and Segment Anything as described in Section 3.3.1 to get a per-pixel mask of the object. We then take the median of the extracted surface normals within the image mask.

Applying Rotations

Unlike for shifting, where we or GPT-4 had to derive an equation in order to apply the correct transformation, Blender provides a variety of powerful functions for rotations, including a built-in function called `Vector.rotate_difference()` which GPT uses to calculate a quaternion rotation from the median surface normal of the render to the median surface normal of the input image. Blender additionally provides built-in methods to convert quaternions to Euler rotations and apply these rotations to the CAD model. GPT-4 demonstrates some latent knowledge of these functions out of the box, but we mention these functions in the prompt to add more consistency in its code generation. The code output of GPT-4 is provided in Figure 3.7.

3.3.4 Orchestrating the the tools and parts to Align

To make it simple for GPT-4 to orchestrate many moving parts, we specifically design our system to be modular. Each tool for gathering information (e.g. pixel shifts, median surface normal) can be run via a single function. Each snippet of code to update the mesh based on

Function	Description
<code>get_shift()</code>	Returns the pixel shift between the two images of the input object
<code>get_width_height()</code>	Returns the width and height of the input object in the original render and the current render
<code>get_position()</code>	Returns the 3D position of the input object in its current state in Blender
<code>shift_from_pixel_diff()</code>	Takes in a horizontal pixel shift, vertical pixel shift, and 3D position and outputs a string of Blender code that shifts the object in 3D in order to undo the input horizontal and pixel shift
<code>shift_depth_from_width_height()</code>	Takes in the image pixel width and height of the object, render pixel width and height, and current 3D position of the object. It outputs a string of Blender code that shifts the object in 3D in order to undo a shift in depth.
<code>get_normals()</code>	Returns the median surface normals of the object in the input image and current render.
<code>rotate_from_normals()</code>	Takes in a target 3D surface normal and a current 3D surface normal and outputs a string of Blender code to rotate the object in 3D towards the target surface normal.
<code>run_blender()</code>	Takes in a code snippet and runs it in Blender, then updates the render of the current state.

Fig. 3.8 Descriptions of the functions built around each tool and code snippet. `get_shift()`, `get_width_height()`, and `get_normals()` are the tools we provide for extracting information for horizontal and vertical shifts (see Section 3.3.1), depth shifts (see Section 3.3.2), and rotations (see Section 3.3.3), respectively. `shift_from_pixel_diff()`, `shift_depth_from_width_height()`, and `rotate_from_normals()` are the functions generated by GPT-4 (see Figures 3.5, 3.6, and 3.7, respectively) that produce Blender code to apply modifications to the CAD model. `run_blender()` takes the outputs of these GPT-generated functions to execute the code in the Blender environment and re-render the CAD model. In combination, these methods enable GPT-4 to apply the Render and Compare loop from end-to-end.

```
44 def transform_object_back():
45     object_name = 'table'
46     for _ in range(num_iters):
47         # Get shift
48         shift = get_shift(object_name)
49         # Get current position
50         current_position = get_position(object_name)
51         # Generate Blender code to undo shift
52         shift_code = shift_from_pixel_diff(shift[0], shift[1], current_position)
53         # Run Blender code
54         run_blender(shift_code)
55         # Update current position
56         current_position = get_position(object_name)
57         # Get width and height
58         width_height = get_width_height(object_name)
59         # Generate Blender code to undo depth shift
60         depth_shift_code = shift_depth_from_width_height(width_height[0], width_height[1], width_height[2], width_height[3],
61                                                         current_position)
62         # Run Blender code
63         run_blender(depth_shift_code)
64         # Update current position
65         current_position = get_position(object_name)
66         # Get normals
67         normals = get_normals(object_name)
68         # Generate Blender code to undo rotation
69         rotation_code = rotate_from_normals(normals[0], normals[1])
70         # Run Blender code
71         run_blender(rotation_code)
```

Fig. 3.9 GPT-generated code to orchestrate the tools and Blender code together for alignment. For each transformation in horizontal and vertical translations, depth translations, and rotations, the code first uses the corresponding information function, followed by the corresponding Blender code generation function, and finally the `run_blender()` function to execute the code. This demonstrates GPT-4's ability to effectively combine these different functions to tackle the overall task of alignment.

the gathered information can also be generated via a method call. A function to execute a Blender code snippet is also provided to tie everything together. Each function and a brief overview are listed in Figure 3.8.

GPT-4 is told it has images of an object before and after an unknown transformation. It is given the names and descriptions of each function in Figure 3.8 and is told to use the functions to move the object back to its original position and orientation. With this prompt, GPT-4 produces Python code that orchestrates the functions together to transform the object back, which is provided in Figure 3.9.

Chapter 4

Experiment Design

Our experiments in Chapter 5 extensively evaluate our system’s abilities in CAD model retrieval and alignment. This chapter discusses our setup for these experiments. The first section discusses the core dataset we use for all experiments. The second section discusses the pipeline for each experiment, including Blender (Blender Online Community, 2018) configuration and rendering. The third section introduces our evaluation metrics for retrieval and alignment.

4.1 Dataset

In this work we use Pix3D (Sun et al., 2018), a dataset of paired images and CAD models. The dataset is split into different categories of objects including chairs, tables, beds, wardrobes, and more. Each category has a set of CAD models and up to thousands of images, each annotated with one of the CAD models. Some examples are visualized in Figure 4.1. The dataset provides information on camera parameters as well as the rotation and translation matrices of the CAD model in the image, allowing us to render and evaluate our alignments.

Mesh R-CNN (Gkioxari et al., 2019) provides random splits of Pix3D called \mathbb{S}_1 and \mathbb{S}_2 . \mathbb{S}_1 splits the images such that all CAD models are used in train and test. \mathbb{S}_2 splits the images such that the models in train do not overlap with the models in test.

Due to the large scale of the dataset and limited resources, we primarily focus on the ‘tables’ category in Pix3D for almost all experiments.

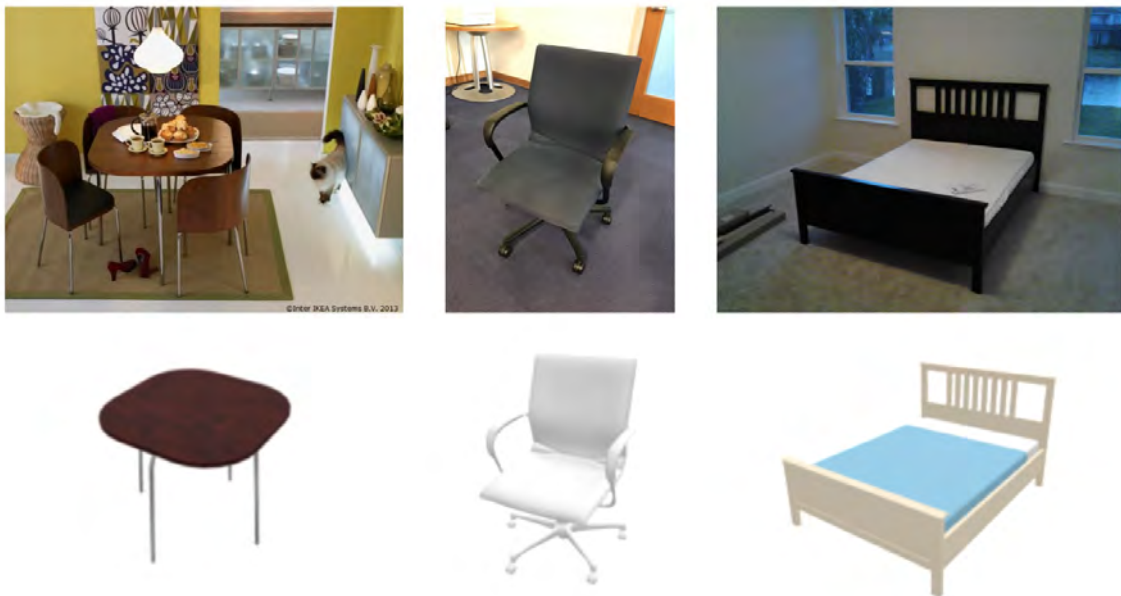


Fig. 4.1 Pix3D has a set of images for different categories. Each image has a corresponding CAD model in the dataset. The top row visualizes some Pix3D images and the bottom row visualizes their corresponding CAD models. Each image is annotated with the 3D position and rotation of the object in the image.

```
71 "IKEA_BJORKUDDEN_1": {  
72   "type": "dining table",  
73   "number of legs": 4,  
74   "tabletop shape": "square",  
75   "size": ["small/medium", "tall"],  
76   "style": "rustic",  
77   "number of drawers": 0,  
78   "number of cabinets": 0,  
79   "number of shelves": 0,  
80   "edge design": ["straight", "sharp"],  
81   "table base design": "none",  
82   "structure complexity": "simple",  
83   "legs style": "rectangular",  
84   "support": "yes",  
85   "apron": "yes",  
86   "tabletop thickness": "medium",  
87   "glass elements": "none",  
88   "wheels": "none",  
89   "leg attachment position": "corners"  
90 }
```

Fig. 4.2 Hand annotations for a single CAD model. Each key is a property and each value/list of values is the annotations for the CAD model.

4.2 Pipeline

We split our system into two sub-components: the retrieval subsystem and the alignment subsystem. This section discusses the implementation details of the experiments we use to evaluate each subsystem, as well as the full system as a whole.

4.2.1 Retrieval Experiment Pipeline

To evaluate our retrieval subsystem, we run experiments on hand-annotated data and auto-annotated data. We discuss implementation details for gathering hand-annotated data, gathering auto-annotated data, using the annotations to retrieve CAD models for new images in our experiments, and notes on why we use truncated datasets for our experiments.

Hand-Annotated Data

We first manually construct a dataset of hand-annotations for each table CAD model. We query GPT-4 (OpenAI, 2023) multiple times for key attributes to annotate for every CAD model. The query we send to GPT-4 is provided in Figure 4.4. Once we have the set of attributes, we determine annotations for each attribute for each CAD model. To do so, we look at the CAD model in Blender (Blender Online Community, 2018), multiple Pix3D images of the model, and resources online related to the CAD model. We then create a JSON file with a dictionary for each CAD model containing each property as a key and the attributes as the value. An example hand-annotation is provided in Figure 4.2.



Fig. 4.3 CAD models are rendered at a slight angle. This helps our system see more of the CAD model’s key features.

Auto-Annotated Data

We auto-annotate the CAD models following our methodology in Section 3.2.2. First, we render each CAD model. We use Blender to render each CAD model, using a focal length of 35mm and placing the object 2m from the camera. We slightly rotate the tables by 15° around the vertical axis and 30° around the depth axis facing out of the camera. This angle lets the render highlight more of the table’s structure and properties. Example renders of CAD models are provided in 4.3.

We then query GPT-4 for properties to annotate using the prompt in Figure 4.4, as we did for hand-annotations. Using the same conversation memory so the LLM can see its previous response with the properties it planned, we follow up with a second prompt, provided in Figure 4.5, asking the model to write code that will auto-annotate the CAD models. We provide key information in the prompt, including that it can ask questions to the VQA model using the `ask()` function and that the CAD model names and images are in a list called `model_imgs`. From this information, GPT-4 outputs a full snippet of code that creates a dictionary named `annotated_cads` containing the annotations. We then execute the snippet of code to get the annotations. An example of the auto-generated annotations for a CAD model is provided in Figure 4.6.

Retrieving CAD models with Annotated Data

Once we have the annotations for each CAD model in our database, whether it be hand-annotations or auto-annotations, we ask GPT-4 to produce a series of questions to ask for any new, unseen image. Our prompt is provided in Figure 4.7. We ask the LLM to output a series of questions that can help it filter through the database of CAD models, such that it asks as few questions as possible for efficiency. In the prompt, we provide the annotations of the CAD models as well as the descriptions of each property that is annotated, which was already generated by GPT-4 earlier. We then get back a list of questions to answer. Since InstructBLIP (Dai et al., 2023) hallucinates often, we prompt GPT-4 to not use open-ended

```

91 You have a variety of different {object_category}s available in a database. You will receive an image of a {object_category} and must
    figure out which {object_category} in the database is in the image. To do so, you must annotate each {object_category} in the
    database with properties of your choice.
92
93 More specifically, you have access to a model that can answer questions about an image. You will render each {object_category} as an
    image and use the model to ask questions about each {object_category} image. This is how you will gather the properties for each {
    object_category}.
94
95 You must decide which properties to gather for the {object_category}s. For a table, for example, you could gather the number of legs of
    the as a property. This means each table will be rendered and you will ask the model how many legs are in the table for each render
    . For each {object_category} in the database, the annotation will be the model's response to this question (recall the input to the
    model is the question and a render of {object_category} table from the database).
96
97 {object_category}s can have the same CAD model but different colors and/or materials, so they may not be a good property to focus on. You
    will later use a VQA model to annotate the CAD models based on the properties you choose. This model is unreliable, so some of its
    annotations may be inaccurate. Plan your attributes to account for this unreliability.
98
99 Decide which properties to allocate for the {object_category}s. You want to choose properties that have high entropy. This means that you
    want to choose properties for which you can identify the correct {object_category} as quickly as possible, so each property should
    help split up the database of {object_category}s as much as possible. Choose 12 to 15 properties.

```

Fig. 4.4 Prompt to GPT-4 to generate properties to annotate for each CAD model of object_category.

```

100 With all these properties in mind, you will now auto-annotate the data.
101
102 You have a function called ask(img, question) which can ask a question about an image. The different CAD models have been rendered and
    put into a list called model_imgs. Each entry of this list has a pair with the first item being the model name and the second item
    being the image.
103
104 Write python code to loop through each CAD model, ask your questions, and output a dictionary called annotated_cads where the key is the
    model name and the value is the corresponding attributes for that model based on the questions you ask and their corresponding
    answers.
105
106 ask() and model_imgs have already been defined and will be loaded before the code is executed. Use tqdm on the loops to help keep track
    of progress.
107
108 Choose a short name for each attribute. DO NOT use the question as the name of the attribute. After planning, run the code.

```

Fig. 4.5 Prompt to GPT-4 to generate executable code to annotate renders of CAD models.

```

110 "IKEA_BJORKUDDEN_1": {
111     "How many legs does this table have?": "2",
112     "What is the shape of the table top?": "square",
113     "What are the dimensions of the table?": "The table in the image has a height of 30 inches and a width of 20 inches.",
114     "Does this table have drawers?": "no",
115     "How many drawers does this table have?": "1",
116     "Does this table have shelves?": "no",
117     "How many shelves does this table have?": "1",
118     "What is the design of the table edge?": "The table edge is square.",
119     "Does this table have extensions?": "no",
120     "What is the material of the table?": "The table is made of wood.",
121     "What is the color of the table?": "The table is brown.",
122     "What is the style of the table?": "square",
123     "Does this table have decorative elements?": "no",
124     "What type of table is this?": "bar",
125     "Does this table have casters?": "no"
126 }

```

Fig. 4.6 Auto-annotations for a single CAD model. Each key is the question asked and each value/list of values is the annotations for the CAD model, which is the response generated by InstructBLIP.

```

127 """You have access to the following dataset of CAD models. the dataset is a dictionary where the key is the name of each CAD model and
    the {object_category}'s attributes are the corresponding dictionary. You are given an image of one of the CAD models and must
    determine which CAD model is in the image. You have a visual question answering model that can answer any question you have about
    the image.
128 A description of the data is as follows: \n""" + data_description + """
129
130 The dataset is as follows:
131 """ + cad_annotations + """ . Choose a series of questions to ask the model in a way that minimizes the total number of questions needed
    to select a CAD model in this dataset. Due to cost issues, you must ask all your questions in one go. Avoid asking open-ended
    questions. Instead give options with each question (for example instead of 'What is the size of the {object_category}?' ask 'Is the
    {object_category} small, medium or large in size?'). Limit the number of options per question to 4-5. Plan all your questions and
    output them with ask_questions"""

```

Fig. 4.7 Prompt to GPT-4 to generate questions to ask a new input image.

```

132 """
133 You have access to a dataset of CAD models. The dataset is a dictionary where the key is the name of each CAD model and the table's
    attributes are the corresponding dictionary. You are given an image of one of the CAD models and must determine which CAD model is
    in the image.
134
135 The dataset is as follows:
136 """ + {annotated_cads} + """ . A series of questions have been asked to the model. The questions and their answers are as follows:
137 """ + qa_pairs + """
138 Note that properties like color and material may be less important, as the image could have a CAD model in a different color.
139
140 The VQA model that answered the questions is unreliable, so some responses may be inaccurate. If no clear CAD model exists, choose the
    next-closest CAD model that fits the provided answers. Based on this information, use the output_answer to output a list of
    possible CAD models from the database that correspond to the image as well as a confidence score between 0 and 1 that you believe
    the image and CAD model correspond. Choose the 10 best CAD models.
141
142 When you are ready, save the CAD models with the available function. DO NOT output code and instead choose the top CAD models yourself"

```

Fig. 4.8 Prompt to GPT-4 to select its top-10 CAD models for a new input image, given the InstructBLIP-generated answers for each question planned by GPT-4. The question-answer pairs are stored in qa_pairs variable and the annotations for each CAD model in the database are stored in annotated_cads.

questions and instead ask questions about quantities or multiple-choice questions. Once GPT-4 returns its questions, we ask each question to InstructBLIP for each image in our experiment.

Once we have the questions and answers for a new image, we query GPT-4 a final time, providing the CAD model annotations and the question and answer pairs, and asking the LLM to output the top-10 CAD models it believes best fit the annotations of the new image. We also ask GPT-4 to output confidence scores for each model to enable top-1, top-5, and top-10 rankings. Our prompt is provided in Figure 4.8. The output of GPT-4 is our final retrieval results. Since the confidence scores are each between 0 and 1, we additionally normalize them.

Truncated Datasets

Two key limitations with GPT-4 is its context length and cost. GPT-4 has a limited context length of 8,000 tokens. In each conversation with a GPT model, the current prompt, any message history, functions to call, and the LLM's response are all included in this context limit. In our retrieval experiments, we found that using annotations for at most 30 out of

the 63 available table CAD models would fit within this context limit across hand- and auto-annotations. We additionally note that the cost for GPT-4 is \$0.03 for every 1,000 tokens, making individual experiments costly. As a result, we only evaluate our retrieval subsystem on 50 images in our experiments. For \mathbb{S}_1 and \mathbb{S}_2 each, we randomly select 50 images out of the 419 images in \mathbb{S}_1 and 393 images in \mathbb{S}_2 .

4.2.2 Alignment Experiment Pipeline

We evaluate alignment by applying random shifts and rotations to table CAD models and analyzing the alignment subsystem’s ability to move the CAD models back to their original positions. We discuss implementation details for generating these random transformations and details on how we apply multiple rotation initializations as discussed in Section 5.2.3.

Generating Random Transformations

To generate random transformations for our alignment experiments, we start with an example in Pix3D. As mentioned in Section 4.1, each example has a real-world image, a corresponding CAD model, the rotation and translation matrices to align the CAD model with the image, and the camera parameters. We use this information to create a render of the CAD model in its ground-truth alignment using Blender. Once we have the CAD model in its ground-truth position in Blender, we apply a random transformation.

For random translations, we compute 3D bounds for where the CAD model can be shifted. We first compute a maximum and minimum size the CAD model can be in the render. The maximum is 10 times the object’s ground-truth width (or the width of the render, whichever is smaller) and the minimum is 1/10th the object’s ground-truth width in the image (or 0.08 times the width of the render, whichever is larger). We then use Equations 3.3 and 3.4 to compute the depth at which the CAD model has these minimum and maximum widths in the renders. These depths become the minimum and maximum bounds from which we sample the the new depth. Once we uniformly sample a depth from these bounds, we compute bounds for horizontal and vertical shifts. We again use Equation 3.3 to compute the furthest horizontally and vertically we can shift the object such that it is still fully visible in the image. With these bounds, we randomly sample a horizontal and vertical position uniformly. After sampling this full 3D position, we render the CAD model at this shifted location and run alignment between this render and the ground-truth image.

For random rotations, we uniformly sample rotations around the horizontal, vertical, and depth axes. For the horizontal and depth axes, we sample a rotation angle between -9° and 9° , ensuring the object is not rotated to an extreme, non-practical orientation. For the vertical

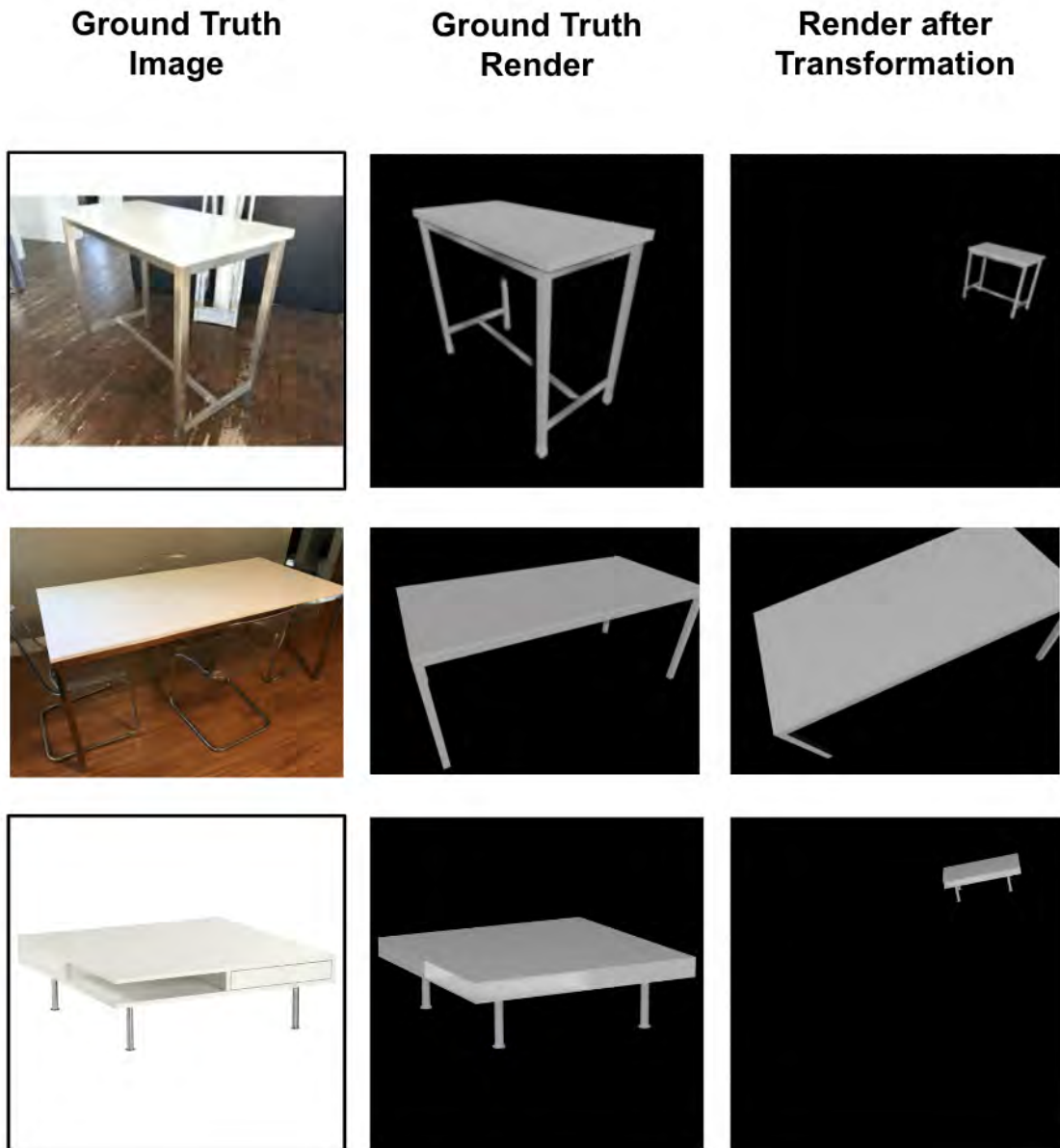


Fig. 4.9 For an input image, the CAD model is loaded with the ground-truth alignment. A random translation (top), rotation (middle), or both (bottom) are applied.

axis, we sample a rotation angle between -90° and 90° since large rotations around this axis are more common in the real world.

We additionally run trials on combinations of rotations and translations. For these experiments, we simply apply a random transformation followed by a random rotation as described above. Examples of a random translation, random rotation, and a combination of the two are provided in Figure 4.9.

Generating Multiple Rotation Initializations

We explore improving our system with multiple rotation initializations. Since these are initializations and we run our entire existing system on each initialization, we report implementation details in this section instead of Chapter 3.

To create multiple rotations, we start with the randomly transformed CAD model (post-rotation and/or translation). We loop through three rotations around the vertical axis: -60° , 0 , and 60° . For each vertical axis rotation, we loop through three rotations around the horizontal axis: -22.5° , 0 , and 22.5° . This results in 9 different rotations (which includes no rotation when the angles are both 0). For each rotation, we render the CAD model and save that as an initialization to run alignment on. An example of the initializations is visualized in Figure 4.10.

Once alignment is run on each initialization, we must select which initialization to treat as the system’s full prediction. To do so, we use Segment Anything (Kirillov et al., 2023) to generate masks on the ground-truth image and the final render and compute the IoU, the ratio of the number of pixels where the masks overlap divided the number of pixels in either mask:

$$IOU = \frac{|\mathbb{P}_{render} \cap \mathbb{P}_{image}|}{|\mathbb{P}_{render} \cup \mathbb{P}_{image}|} \quad (4.1)$$

where \mathbb{P}_{render} and \mathbb{P}_{image} are the set of pixels within the mask of the final render and input image, respectively. We select the output which has the highest final IOU as our final prediction.

4.2.3 End-to-End Experiment Pipeline

For our end-to-end experiment, we combine key implementation details from the experiments for each subsystem. We start by following the system pipeline for gathering retrievals, starting with auto-annotations and outputting the top-10 CAD models and their confidence scores. We then select the CAD model with the highest confidence score and render it in Blender. Since each image has different focal lengths, we place the CAD model $45 * focal_length$

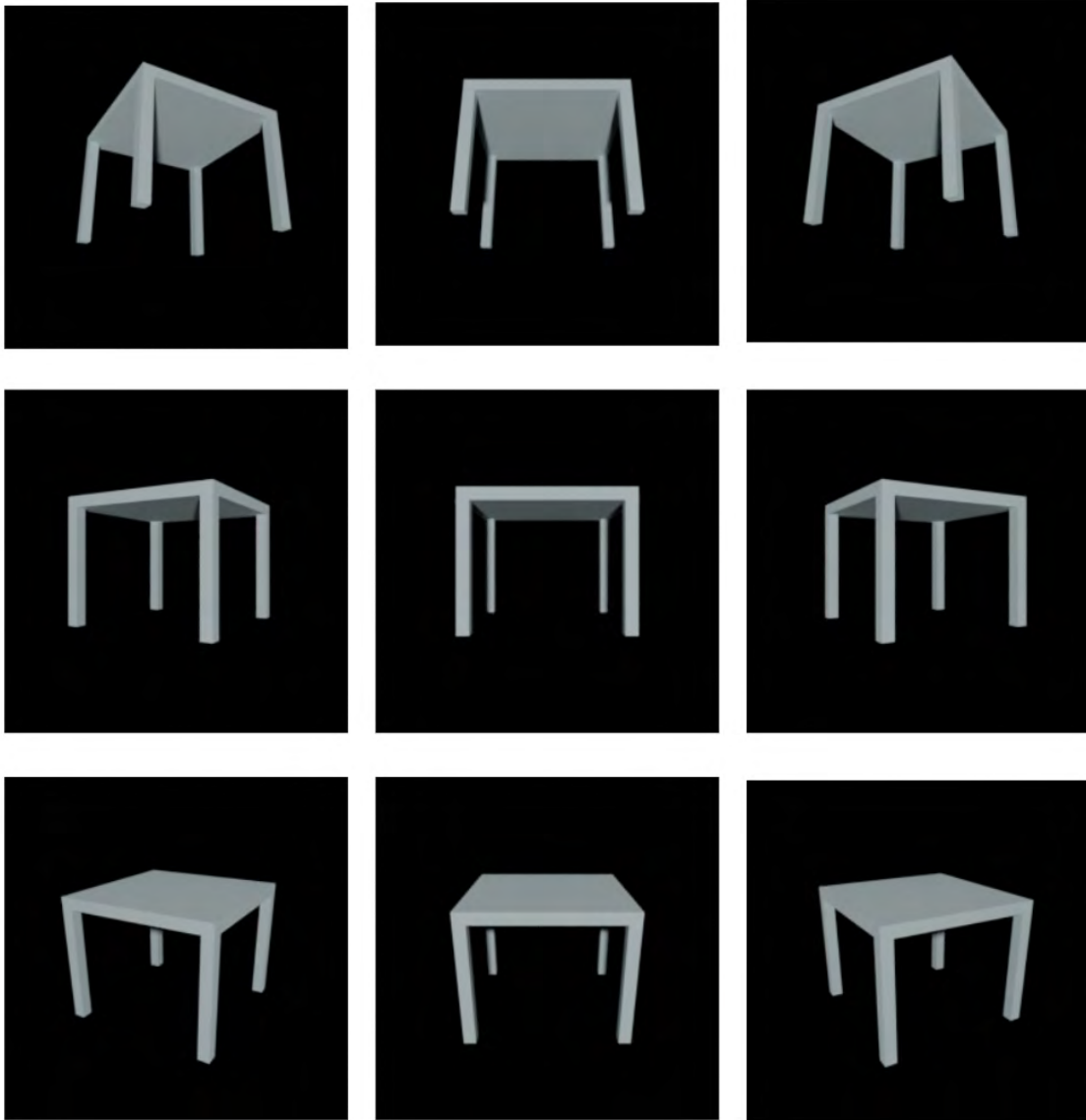


Fig. 4.10 Renders for a single CAD model after each rotation initialization. Our system is run on each initialization, and the best result is selected by taking result with the largest mask IOU between the final render and the object in the input image.

meters¹ from the camera. We then apply the multiple rotation initializations discussed in Section 4.2.2 and run our alignment system. Again, we select the final prediction out of each rotation initialization by taking the final prediction with highest mask IOU between its render and the input image.

4.3 Evaluation Metrics

To evaluate our retrieval system on its own, we compute Top-1, Top-5, and Top-10 retrieval accuracies. We evaluate alignment and our end-to-end system using the AP^{mesh} metric.

- **Retrieval Accuracies** compute the percentage of evaluation examples where the predicted CAD model is the same as the actual CAD model in the input image. We compute Top-1, Top-5, and Top-10 accuracies. Top-1 is the accuracy when the predicted CAD model with the highest confidence matches the ground-truth. Top-5 accuracy is computed as the percentage of examples where the ground-truth CAD model is one of the predicted CAD models with a top-5 confidence score. Since our system outputs 10 CAD models, Top-10 accuracy is the percentage of examples where the ground-truth CAD model is one of the outputted CAD models. Therefore, top-k accuracy is

$$Acc_k = \frac{1}{N} \sum_{i=1}^N \mathbb{1}(conf_i(gtcad_i) \geq c_{k,i}) \quad (4.2)$$

where N is the number of examples, $conf_i$ is a function that maps from a CAD model to its confidence score for the i 'th example², $gtcad_i$ is the ground-truth CAD model for the i 'th example, and $c_{k,i}$ is the k 'th highest confidence score for the i 'th example.

- AP^{mesh} is a complex metric designed to evaluate the performance of systems that identify and reconstruct objects in 3D. This metric is defined as the mean area under the per-category precision-recall curve for predictions. Each prediction is a true positive if its predicted category label is correct, it is not a duplicate prediction (in the case of multiple objects in an image), and its $F1^{0.3}$ is greater than an IOU threshold. Therefore, the AP^{mesh} is associated with its threshold³. $F1^{0.3}$ is the harmonic mean of 1) the fraction of points in the ground-truth mesh that are within 0.3 of a point in the predicted

¹Note the focal length is in millimeters so we convert to meters first, hence the large multiplicative constant.

²CAD models that are not in the Top-10 predictions for the i 'th example have $conf_i = 0$.

³e.g. an AP^{mesh}_{50} score uses an IOU threshold of 0.5, or 50%

mesh and 2) the fraction of points in the predicted mesh that are within 0.3 of a point in the ground-truth mesh.

We follow Langer et al. (2021) and compute an aggregate AP^{mesh} score, computed as the average of the AP^{mesh} scores at IOU thresholds of 0.5 to 0.95 with 0.05 increments. This is also referred to as the AP50-95 score in the COCO object detection protocol (Lin et al., 2014).

We additionally note that, since we focus on tables alone, we assume perfect category classification of tables whereas competing methods apply an object detector before reconstruction or retrieval and alignment.

Chapter 5

Results

This section covers our key experiments for evaluating the different components of our system as well as their future potential. The first set of experiments focuses on CAD model retrieval and the system’s ability to find the best table mesh corresponding to an input image. The second set of experiments focuses on CAD model alignment, specifically aligning a CAD mesh in 3D with an input image via translation and rotation. The final set of experiments evaluates the entire system from input image to output aligned CAD model.

5.1 CAD Retrieval using Foundation Models and LLMs

We evaluate our system’s ability to retrieve a CAD model that best matches an object in an input image, primarily focusing on tables. We apply experiments on hand-annotated CAD models, auto-annotated CAD models, and we evaluate our system’s ability to extend to a new object category.

5.1.1 Classical Trained Methods on Unseen Data

Traditional methods (Kuo et al., 2020; Langer et al., 2021) tackle CAD retrieval by training a network to predict an embedding vector from an image. This approach is limited by the quality and variety of data the networks are trained on. An inevitable result of this weakness is low test performance when these systems encounter an image with a CAD model the networks have not been trained on. For the \mathbb{S}_1 split, Langer et al. (2021) trains its embedding network on renders of every CAD model available in Pix3D (Sun et al., 2018), resulting in the network explicitly learning to separate the embeddings of different CAD models through its triplet loss and having strong performance on the test data. In \mathbb{S}_2 , the network is trained on a subset of CAD models but is evaluated on images of different CAD models. As a

Evaluation Metric	\mathbb{S}_1 Split			\mathbb{S}_2 Split			\mathbb{S}_1 to \mathbb{S}_2 Difference in Accuracy	
	Hand-Annotated*	Auto-Annotated*	Langer et al. (2021)	Hand-Annotated*	Auto-Annotated*	Langer et al. (2021)	Auto-Annotated*	Langer et al. (2021)
Top-1 Accuracy (%)	0	30	65	0	42	24	+12	-41
Top-5 Accuracy (%)	8	58	-	16	78	-	+20	-
Top-10 Accuracy (%)	-	70	85	-	84	52	+14	-33

Table 5.1 Quantitative comparison of results on the \mathbb{S}_1 and \mathbb{S}_2 splits for CAD model retrieval using hand-annotated and auto-annotated data. Our system performs significantly better on auto-annotated data in comparison to hand-annotated data. Our system also notably outperforms competitors in the \mathbb{S}_2 split, a scenario where their networks are not trained on the CAD models, demonstrating our system’s strength on unseen data. **Our results are for 50 random test images of tables, with 30 out of the 63 Pix3D tables as possible CAD models. Langer et al. (2021) is run on the entire dataset for all categories. We are unable to report Top-10 Accuracy for hand-annotations due to query limitations in the GPT-4 API.*

result, their approach loses notable performance on the new images, as demonstrated in Table 5.1 where there is a large loss in accuracy between \mathbb{S}_1 and \mathbb{S}_2 for both Top-1 and Top-10 accuracy.

5.1.2 LLM- and VQA-based Retrieval with Hand-Annotations

We first run our retrieval system on random samples of Pix3D table images, retrieving CAD models from a hand-annotated dataset. Details on the construction of this dataset are discussed in 4.2.1. We pass the hand-annotations into GPT-4 (OpenAI, 2023) and tell it to plan a series of questions to ask InstructBLIP (Dai et al., 2023) for a new image. With the planned questions, we apply the rest of our retrieval method, asking each question about the image to InstructBLIP, passing the answers and CAD dataset into GPT-4, and collecting the top CAD models outputted by the LLM. The results of this approach are provided in Table 5.1.

The results show that our system struggles with hand-annotations. The high granularity is not well-suited for the inconsistent predictions of InstructBLIP. As demonstrated in Figure 5.1, InstructBLIP provides responses that do not match the hand-annotations of the true CAD model. In addition, InstructBLIP can misunderstand parts of the question, like interpreting "glass elements" of a table as any glass items on or around the table. At the same time, InstructBLIP demonstrates commendable intuition of complex characteristics like the type of table, detecting that the last example image is a nightstand.

Diving deeper into the behavior of our system using hand-annotated data, in Figure 5.2 we plot the distribution of predictions of each CAD model in comparison to the distribution of actual ground-truth CAD models in the images. We can see that the CAD models predicted by the system differ immensely from the actual CAD models in the image, to the point that




Image	GPT-Generated Question	InstructBLIP Response	Correct Model Hand Annotation	Selected Model Hand Annotation
	How many legs does the table have? Is it 0, 3, or 4?	3	IKEA_FUSION 4	IKEA_LINDVED 3
	Does the table have any glass elements? If yes, where are they located?	Yes, there is a glass vase on the table.	IKEA_TORSBY_1 can sometimes have glass tabletop	IKEA_LINDVED full glass tabletop
	What is the type of the table? Is it a dining table, nightstand, desk, console table, end table, side table, coffee table, or a wall mounted table?	The table in the image is a nightstand.	IKEA_RAST general-purpose table, nightstand	IKEA_DOCKSTA dining table

Fig. 5.1 Examples of retrieval with hand-annotations. The first column contains the input image. The second column is the question planned by GPT-4 for the attribute we visualize. The third column contains InstructBLIP's response for the image and question, and the fourth and fifth columns are the hand-annotations for the ground-truth CAD model and system's output CAD model, respectively. Due to the unreliability of InstructBLIP responses, the system will often collect annotations from input images that match incorrect CAD models, as exemplified by the top example. There are some responses, like in the second example, where ambiguity arises based on misinterpretations of the question. At the same time, the model does get some predictions correct, like in the third example.

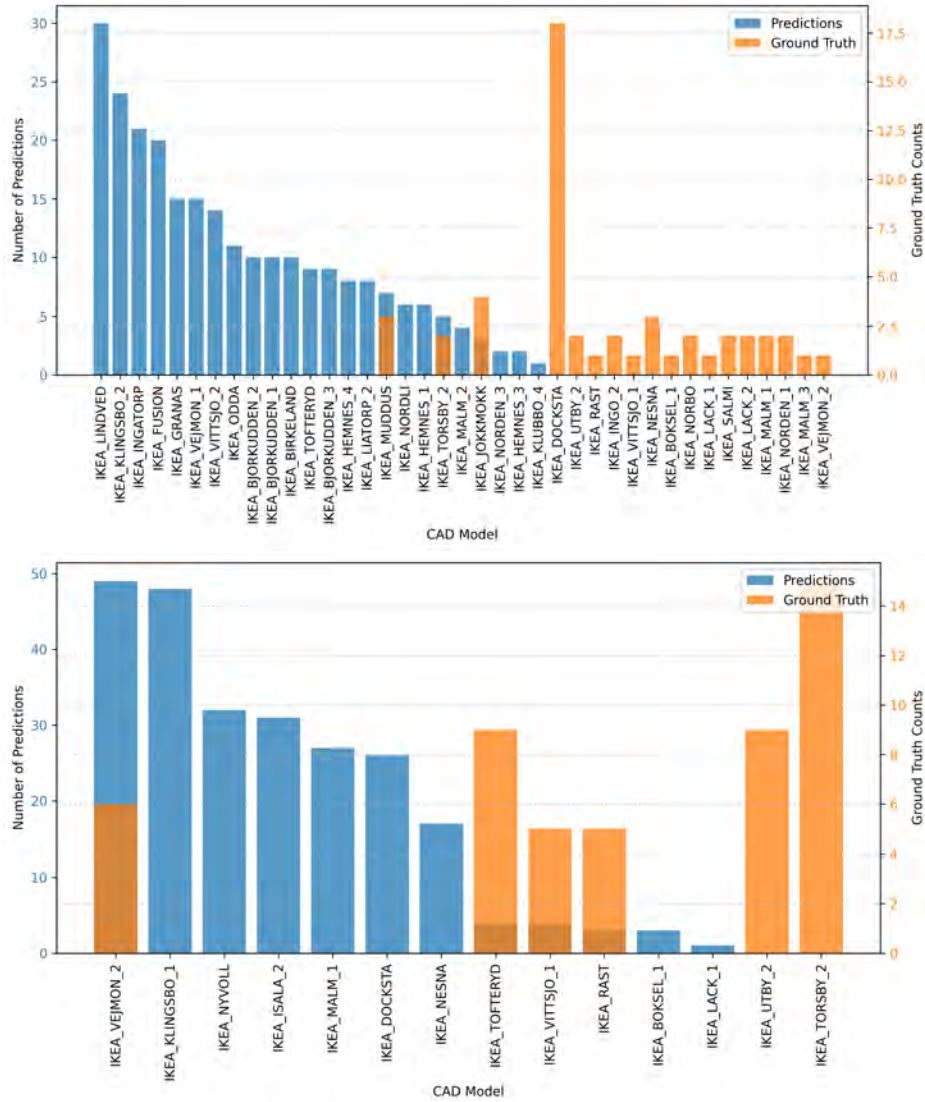


Fig. 5.2 Histogram of final predictions using hand-annotations (blue) in comparison to the distribution of ground-truth CAD models (orange) for S_1 (top) and S_2 (bottom). There is little-to-no overlap between the predictions and ground-truth data. This demonstrates that, after aggregating all the questions and answers for an input image, the system struggles to correctly accumulate information from an input image to match human-level annotations. This further demonstrates a bias towards certain CAD models, suggesting the InstructBLIP outputs may better match the hand-annotations of a specific subset of CAD models in the dataset.

the system tends to predict CAD models that never show up in any of the images tested. This difference exemplifies the mismatch between the outputs of InstructBLIP and the granular, ground-truth detail of the hand-annotations. Furthermore, the bias towards a specific few incorrect CAD models suggests that InstructBLIP consistently outputs incorrect answers which coincidentally align with these CAD models' annotations.

5.1.3 Improving Retrieval with Auto-Annotations

We now run our full retrieval system including auto-annotation of CAD models using GPT-4 (OpenAI, 2023) and InstructBLIP (Dai et al., 2023). As described in Section 3.2.2, we render each CAD model, use GPT-4 to write code that will ask questions about each render, and execute that code to automatically generate a database with the CAD model annotations. We then retrieve CAD models using the same approach as described in Section 5.1.2: we pass the CAD model annotations into GPT-4, get back a list of questions to ask for the input image, ask those questions for the input image using InstructBLIP, and pass the responses to GPT-4 again to select the best-matching CAD models from the data. The results using our auto-annotations are provided in 5.1.

Using auto-annotations significantly improves retrieval accuracies. Our system becomes more competitive with Langer et al. (2021) on the \mathbb{S}_1 split and even outperforms them on the \mathbb{S}_2 split. We note that since our system retrieves from 30 candidate table CAD models whereas Langer et al. (2021) retrieves from 63 candidate table CAD models (alongside the CAD models of other object categories), it is more accurate to compare our system's Top-5 accuracy with Langer et al. (2021)'s Top-10 accuracy, which we still outperform for \mathbb{S}_2 . Furthermore, our system surprisingly has a positive change in performance between \mathbb{S}_1 and \mathbb{S}_2 performance in comparison to Langer et al. (2021)'s large performance loss, though this can likely be attributed to random chance between our sub-samples of \mathbb{S}_1 and \mathbb{S}_2 images.

Looking at specific examples of annotations in Figure 5.3, we can evaluate how auto annotations improve performance for retrieval. As demonstrated by the first example, in many cases InstructBLIP hallucinates incorrect outputs for certain CAD models. When using the hand-annotated data, these hallucinations for input images do not match the ground-truth annotations we created, resulting in the system retrieving different CAD models for the input image. For some questions, InstructBLIP provides the same incorrect response for almost every CAD model. For example, in the auto-annotations, InstructBLIP incorrectly believes that every table has 1 drawer. As displayed in the second example of Figure 5.3, GPT-4 intelligently avoids asking questions about drawer quantity, but still asks if *any* drawers are present on the table. As a result, for the auto-annotated retrieval, InstructBLIP provides the same output for the correct and incorrect CAD model, preventing incorrect information that




Image	GPT-Generated Question	InstructBLIP Response	Correct Model Annotation	Incorrect Model Annotation	
	How many legs does the table have? Is it 0, 3, or 4?	3	4	3	Hand Annotations
	How many legs does the table have? Is it 1, 2, or 4?	2	2	1	Auto-Annotations
	How many drawers does the table have? Is it 0, 1, or 2?	1	0	1	Hand Annotations
	Does the table have drawers? Yes or no?	no	no	no	Auto-Annotations
	What is the shape of the tabletop? Is it square, rectangular, round, oval, or a rounded square?	square	square	round	Hand Annotations
	What is the shape of the table top? Is it rectangle, square, round, or oval?	rectangle	square	rectangle	Auto-Annotations

Fig. 5.3 Qualitative examples of hand-annotations vs auto-annotations. Whereas hand-annotations are perfect, auto-annotations make consistent mistakes for both the image and CAD models, resulting in the auto-annotations better matching annotations for new images. InstructBLIP often hallucinates 2 legs or 1 leg instead of 3 legs, 4 legs, or no legs at all. By auto-annotating, the database's annotations include these hallucinations, resulting in GPT-4 planning questions around the incorrect data, leading to more consistency in predictions. In the second example, InstructBLIP hallucinates 1 drawer for every table. GPT-4 therefore does not ask a drawer quantity question and instead asks if any drawers are present or not, leading to fewer mismatches compared to the hand-annotations. In the third example, we can see a case where InstructBLIP is inconsistent with its responses on the same image. It correctly outputs "square" for the question from hand-annotated data but incorrectly outputs "rectangle" for the question from the auto-annotated data. This is a rare example where the hallucinations from InstructBLIP benefit the hand-annotated retrieval.

Evaluation Metric	S_1 Split		S_2 Split		S_1 to S_2 Difference in Accuracy	
	Tables	Beds	Tables	Beds	Tables	Beds
Top-1 Accuracy (%)	30	30	42	38	+12	+8
Top-5 Accuracy (%)	58	56	78	54	+20	-2
Top-10 Accuracy (%)	70	72	84	76	+14	+4

Table 5.2 Quantitative comparison of retrieval results on table and bed images. Our system maintains strong performance in both categories, demonstrating that our system can easily adapt to new categories without any customization.

can actually affect the retrieval. For the hand-annotated retrieval, however, GPT-4 asks how many drawers are present, and InstructBLIP makes the same hallucination which does not match the hand-annotations for the correct CAD model.

5.1.4 Extending to New Object Categories

Although our retrieval subsystem performs well on tables, it is uncertain if our approach is simply well-suited for table retrieval but does not expand to other categories. We therefore attempt retrieval on beds in Pix3D and provide our results in Table 5.2. Our results demonstrate that our system still performs strongly on new categories like beds without any change in its architecture or training. This performance demonstrates the adaptive power of our system, suggesting that it can adapt to any type of object for which GPT-4 has sufficient knowledge of the properties of the object category.

5.2 CAD Alignment

We evaluate our alignment sub-system based on its ability to re-align objects after random, unknown transformations including random translations, rotations, and both. We additionally evaluate how applying multiple rotation initializations improves the performance of our system.

5.2.1 Classical Methods on Unseen Images

While classical methods tackle the task of fitting an object to an image in a variety of different ways, each of them rely on trained networks that inevitably struggle with unseen data. Langer et al. (2021) relies on a Swin Transformer (Liu et al., 2021) trained on Pix3D data to produce segmentation masks as part of their alignment. Mesh-RCNN (Gkioxari et al., 2019) uses a

Evaluation Metric	Langer et al. (2021)*	Mesh-RCNN*	Translation	Rotation	Combined
\mathbb{S}_1 AP ^{mesh}	24.7	11.0	25.2	12.8	1.3
\mathbb{S}_2 AP ^{mesh}	3.5	7.0	36.4	10.0	1.1
Change in Performance from \mathbb{S}_1 to \mathbb{S}_2 (%)	-85.8	-36.4	+44.4	-21.9	-15

Table 5.3 Quantitative results on the \mathbb{S}_1 and \mathbb{S}_2 split for re-aligning a table CAD model after an unknown transformation, in comparison to end-to-end results for competitors. Columns indicate the type of random transformation applied (combined refers to both transformation and rotation) or the name of the competitor. The system performs well on translation, but notably loses performance with rotation and the combined transformations. Our system also maintains a much larger portion of performance between \mathbb{S}_1 and \mathbb{S}_2 compared to other methods, even improving for translation. **Note that our AP metrics are computed assuming perfect classification and CAD retrieval. Langer et al. (2021) results are taken from their end-to-end retrieval and alignment results. Mesh-RCNN results are also end-to-end. We pull Mesh-RCNN results from Langer et al. (2021) because the original paper only reports AP50 scores.*

trained network to produce its initial voxel structure of the object. As demonstrated in Table 5.3, these methods lose notable performance between the \mathbb{S}_1 and \mathbb{S}_2 splits. We consider the caveat that these systems’ results are end-to-end and therefore incorporate their retrieval or object categorization accuracy. Langer et al. (2021) still loses significantly more performance between the two datasets considering its difference in retrieval performance in Table 5.1. Mesh-RCNN does not retrieve CAD models and instead directly predicts meshes from images, therefore its loss in performance is a noteworthy point of comparison.

5.2.2 Alignment with Foundation Model Tools

Since our alignment system consists of separate tools for translation and rotation, we evaluate their performance separately and in combination. We evaluate these components by randomly perturbing the 3D position and rotation of examples in Pix3D and rendering the transformed objects for our system to re-align, as described in Section 4.2.2. The quantitative results of these experiments are provided in Table 5.3. Our system performs strongly on translation, but notably loses performance on rotation and further loses performance on the combination of the two.

As demonstrated in Figure 5.4, our system can successfully align a variety of different tables that have been translated unknown amounts. Our system is especially effective at aligning tables that are fully in frame and only partially occluded. We can see in the second example that the segmentation model is sometimes able to separate the occluding objects from the table, strengthening the performance of the alignment. At the same time, there

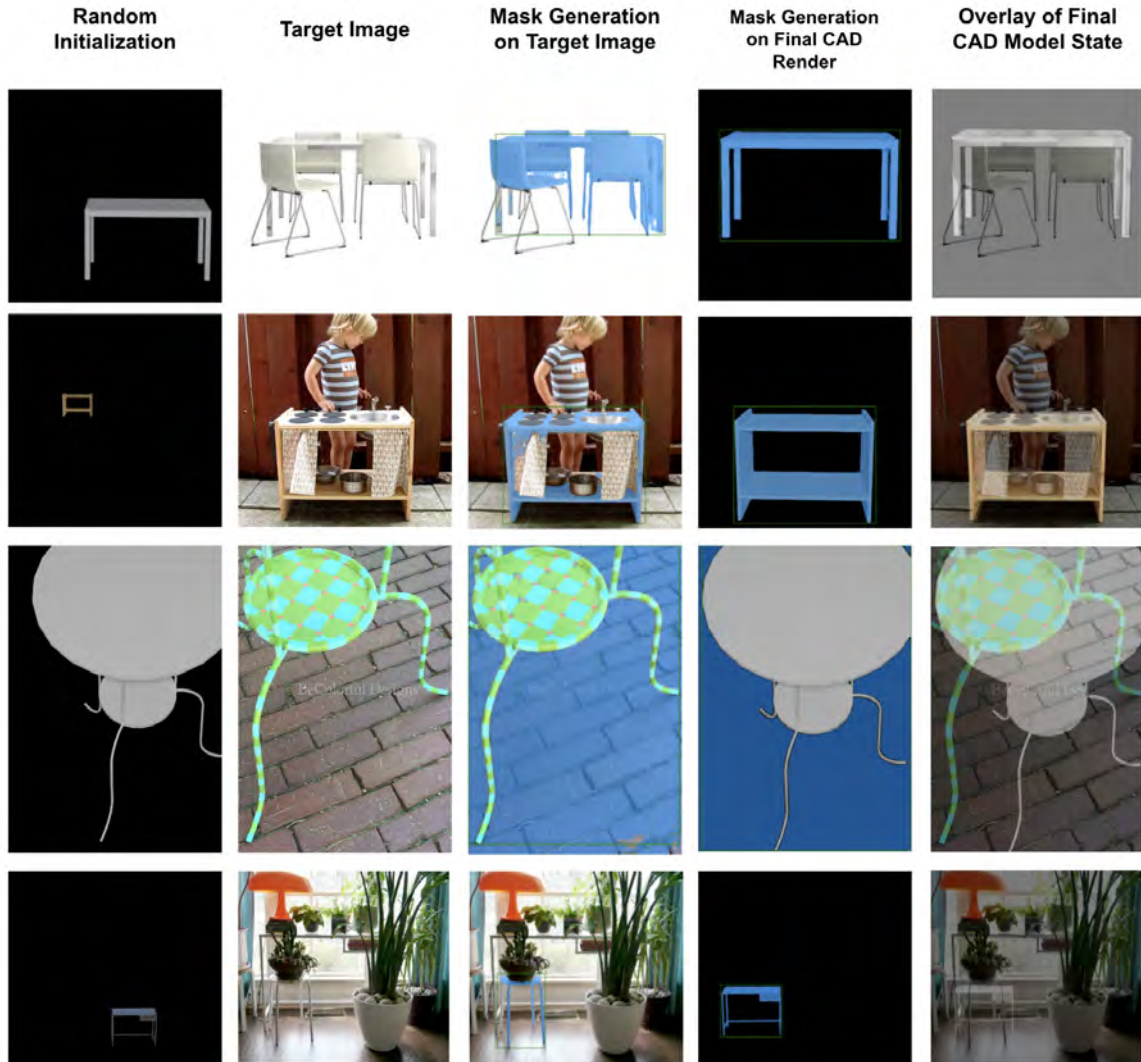


Fig. 5.4 Examples of alignment after random translations. The first column contains the renders after the CAD models are randomly translated from the ground-truth position. The second column contains the target image. The third and fourth columns visualize the segmentation masks generated by the system to calculation horizontal, vertical, and depth shifts. In the final column we overlay the aligned CAD model with the target image. As demonstrated in the first and second examples, the system is able to translate objects to their target position, even when the target image is occluded by other objects. In the third and fourth examples, however, we see failure cases. The system struggles with objects that are partially out of frame and struggles to identify parts of tables alone. The system also often fails when multiple tables are in the target image, since it is not designed to distinguish between multiple tables for segmentation.

are cases where the segmentation is inadequate for alignment. When the object is partially out of frame, the segmentation mask cannot extend out of frame, resulting in the width and height estimations being inaccurate. As a result, you can have cases like in the third example where both segmentation masks take up the full image, resulting in width and height being the same and the horizontal and vertical pixel differences being zero. In other cases, the target position may be partially out of frame, resulting in the target width or height being smaller than expected. As a result, the system interprets that as the object being smaller in the target image and therefore pushes the CAD model further in depth, resulting in incorrect alignment. In the fourth example, we see another failure case where there are multiple tables in the image, so the segmentation network segments the wrong table. As a result, the CAD model is properly aligned to the wrong table.

Our system is able to rotate tables to align surface normals, but certain properties must hold true in order to have an accurate alignment. As demonstrated in the first two examples of Figure 5.5, our system can fix random rotations almost perfectly when the random initializations are well-suited for computing median surface normals to represent orientation. When the primary surface of the table (like the tabletop in the first example of Figure 5.5) is heavily in-view of both images, it takes up the majority of pixels corresponding to the table. As a result, the median surface normals in the CAD render and in the target image will be on that same surface, resulting in consistency in surface normals and ideal alignment. There are additional scenarios where rotation works well when the primary surface is not fully in view, like in the second example of the figure. The majority of pixels for the table correspond to the longer horizontal side of the table in both the render and target image, leading to the same behavior of the median surface normal lying on the same part of the table in both images.

As exemplified in the third example, however, when a different part of the table is mostly in view between the render and target image, the system struggles. The median surface normal in the CAD render is on the bottom of the tabletop, whereas in the target image it is on the top of the tabletop. As a result, the system aligns the normals, resulting in the CAD table being flipped over. An edge case visualized in the fourth example shows how the surface normal estimator struggles with glass and Blender (Blender Online Community, 2018) outputs zero surface normals for the glass surface in the CAD model. At the same time, the segmentation mask still includes the glass tabletop, so the median surface normal becomes invalid and therefore the system cannot align the table.

Even when the same primary surface is in view, the system can still fail to align the object, as visualized in the fifth example. When the object is initialized such that it is rotated around the axis of the median normal relative to the object in the target image, the system is unable to fully fix the rotation. This is because the system will align the surface normals,

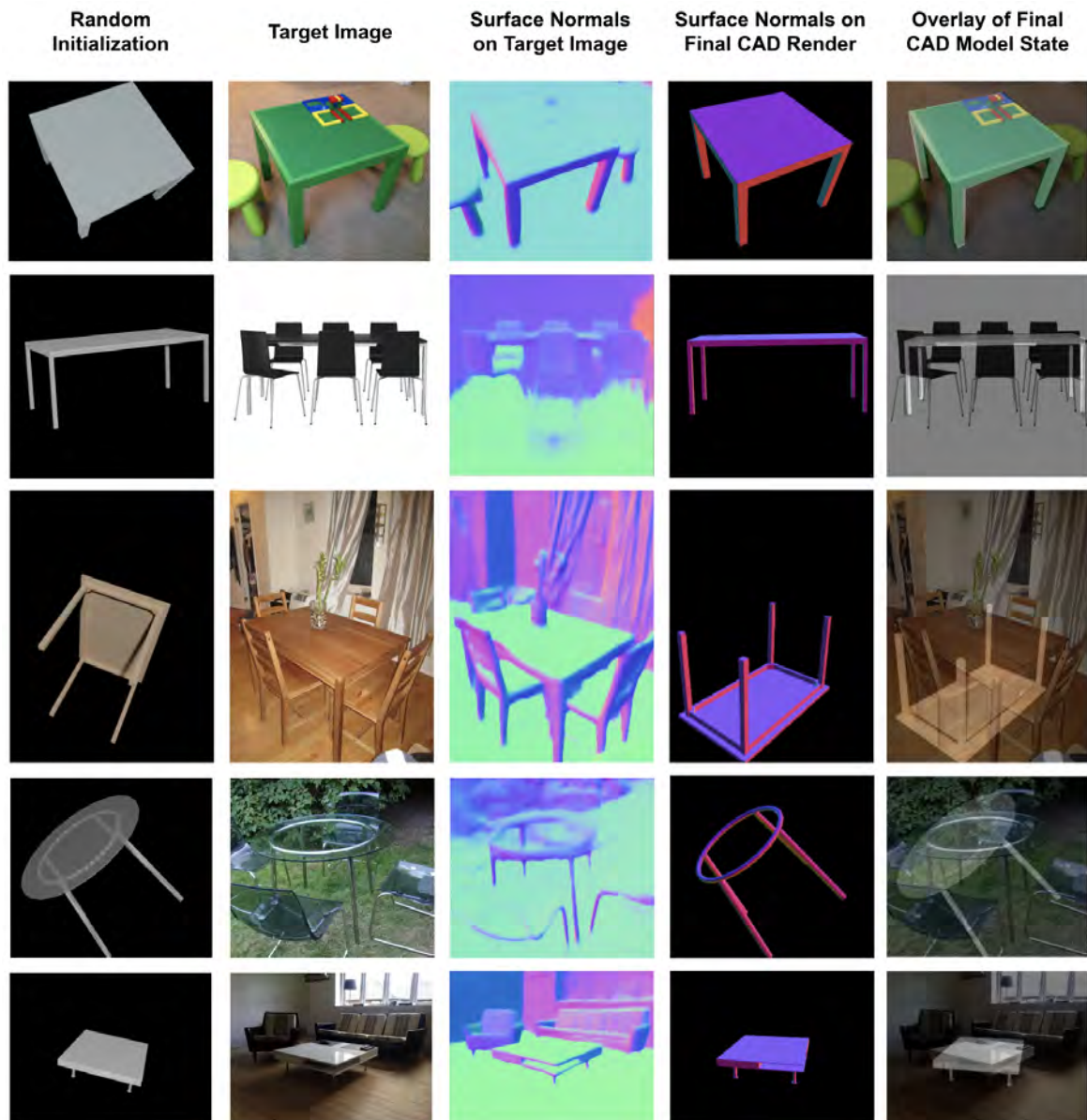


Fig. 5.5 Examples of alignment after random rotations. The first column contains the renders after the CAD models are randomly translated from the ground-truth position. The second column contains the target image. The third and fourth columns visualize the surface normals generated by the system. In the final column we overlay the aligned CAD model with the target image. The system is able to align randomly rotated tables when the object is only slightly rotated from its original position, like in the first and second examples. The system fails when the object is rotated too much, causing the primary visible surface to change. In such cases, the surface normals extracted from the render and target image are on different parts of the table, causing the system to align those parts together instead. The system also struggles with glass since the surface normal estimator and Blender treat them as transparent. Furthermore, as demonstrated in the final example, if the table is rotated around the axis perpendicular to the primary visible surface, the system cannot re-align the table fully.

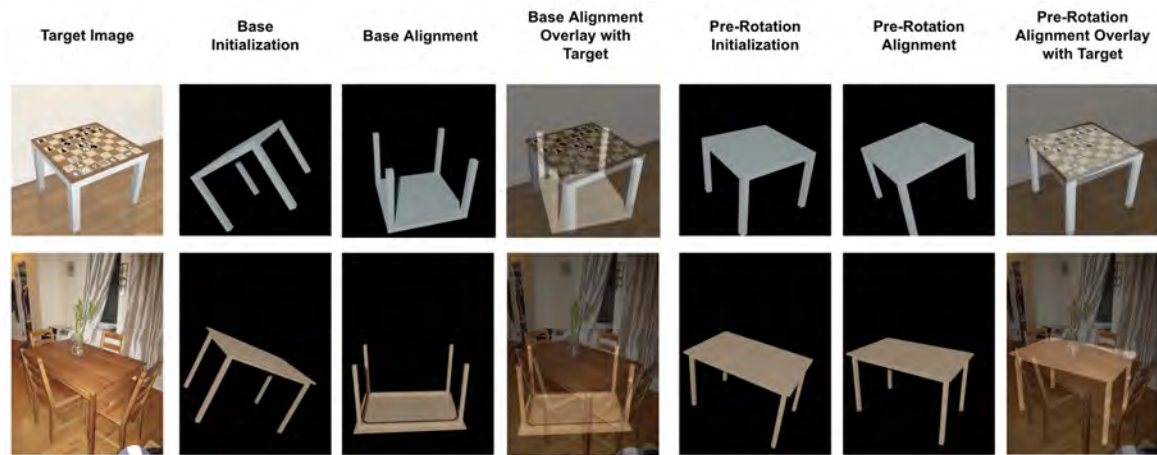


Fig. 5.6 Examples of alignment with and without multiple pre-rotations. By applying multiple possible pre-rotations, it is likely one pre-rotation will be close to the target rotation like in the examples above. This helps the system easily align CAD models to input images without changing its architecture, but comes with the cost of additional compute.

but it cannot fix the remaining rotation *around* the axis of the surface normal. We explore a solution to this problem in 5.2.3.

For combinations of random translations and rotations, we inevitably see even weaker performance since the challenges from each alignment subtask compound. We additionally find that poor rotations weaken the performance of translation since different rotations produce different bounding boxes for tables, which the translation tools rely on. This further weakens the system’s performance, explaining the low AP^{mesh} scores.

5.2.3 Improving Performance with Multiple Rotation Initializations

Since our system is incapable of rotating a table around its primary surface normal, we pre-rotate the table at 9 pre-determined angles and then align for each pre-rotation, resulting in 9 possible alignments for a single target image. We select the best alignment by taking the mask IOU between the render of each alignment and the target image and selecting the highest one. Further details are provided in Section 4.2.2. We evaluate the benefits of this approach by comparing the performance before and after this modification.

As demonstrated in Figure 5.6, applying pre-rotations increases the chance that an initialized rotation is close to the target rotation. When such an initialization occurs, the system is able to more-accurately align the CAD model to the image, like in the first example. At the same time, because we attempt only a few possible pre-rotations there are still scenarios where the initialization is slightly rotated around the axis of the surface normal, preventing an ideal alignment but still getting close like in the second example.

Evaluation Metric	Single Rotation	Multiple Pre-Rotations	Single Rotation (w/ Translation)	Multiple Pre-Rotations (w/ Translation)
\mathbb{S}_1 AP ^{mesh}	4.0	12.8	0.5	1.4
\mathbb{S}_2 AP ^{mesh}	3.8	10.0	0.2	1.6

Table 5.4 Quantitative results on the \mathbb{S}_1 and \mathbb{S}_2 split using multiple pre-rotations vs no pre-rotations. Using the pre-rotations evidently improves performance significantly, with on average 3x improvements for every metric.

Evaluation Metric	Langer et al. (2021) *	Mask2CAD*	Mesh-RCNN*	Ours	Ours (GT Norms)
\mathbb{S}_1 AP ^{mesh}	24.7	29.2	11.0	0.018	0.81
\mathbb{S}_2 AP ^{mesh}	3.5	3.6	7.0	0.72	2.3

Table 5.5 Quantitative results on the \mathbb{S}_1 and \mathbb{S}_2 split for end-to-end retrieval and alignment. After combining both parts of our system together, we achieve performance that demonstrates our system’s potential to compete with state-of-the-art models in future iterations. We highlight this potential by evaluating our model’s performance with ground-truth normals instead of the surface normal estimator. With ground-truth normals, our system approaches the performance of our competitors on the \mathbb{S}_2 split. Our system performs unusually worse on the \mathbb{S}_1 split, which we believe can be attributed to random chance since we sample around 12% of the available images in the dataset. **Again note that competitor results also include object category classification, whereas we assume perfect classification.*

Quantitative results with and without pre-rotations are provided in Table 5.4. The pre-rotations significantly increase performance in both \mathbb{S}_1 and \mathbb{S}_2 splits. These improvements demonstrate that there are clear, achievable increases in performance for this alignment system. For example, with more pre-rotations, we can likely further increase performance to compete with state-of-the-art competitors.

5.3 End-to-End Topology Retrieval

After combining our retrieval and alignment systems together, we evaluate our approach on table images in the \mathbb{S}_1 and \mathbb{S}_2 splits and report our quantitative results in Table 5.5. We find that our system is able to retrieve and align CAD models for some input images, but struggles in comparison to other methods. We attribute this to a combined loss of performance between the retrieval and alignment subsystems. As seen with our system’s performance using ground-truth surface normals, however, we can quickly achieve better performance, demonstrating this approach has exciting potential to quickly compete with and even outperform state-of-the-art competitors in future iterations.

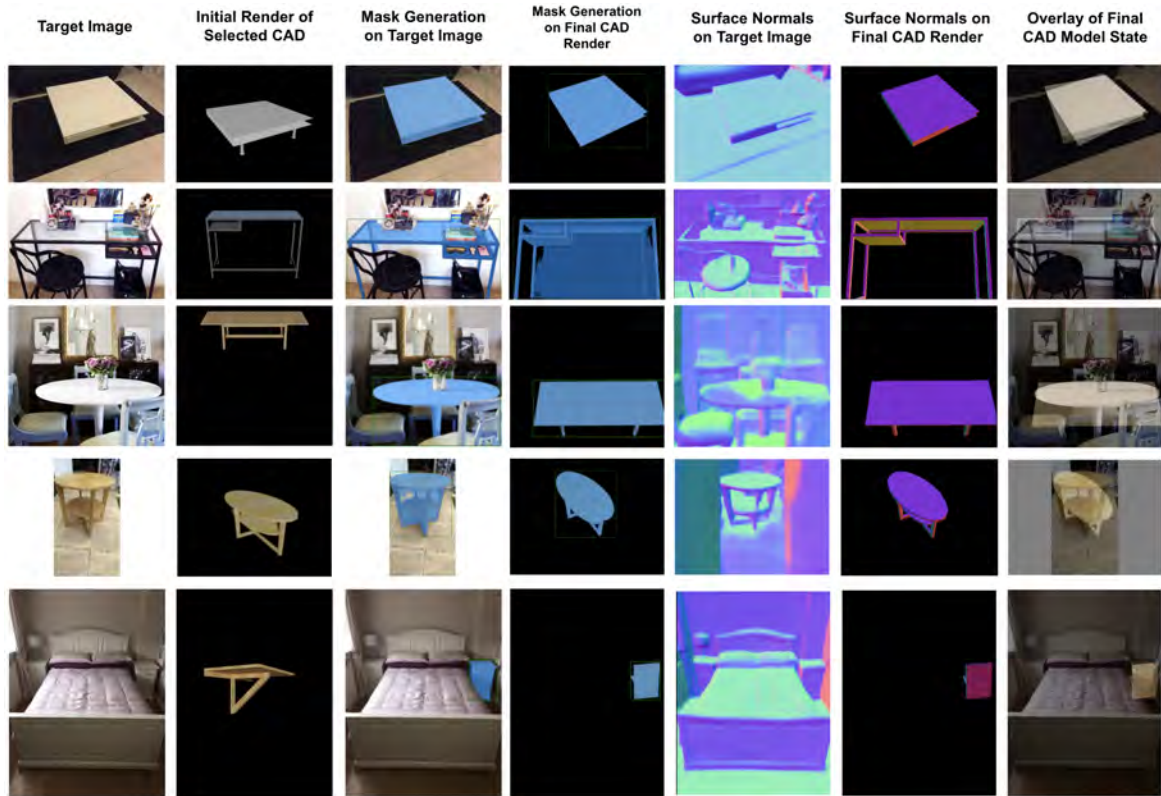


Fig. 5.7 Selected examples of end-to-end retrieval and alignment attempts by our system. We highlight some successes and key weaknesses in our system that cause low performance. Our system aligns the first example very well, but unfortunately has low AP^{mesh} because the table is rotated 90 degrees and the shelves do not overlap. Our system is able to translate the tables based on the segmentation masks in the second and third examples, but the table is partially cut out in the target image and our system does not shift the tables down further. In addition, the second example does not rotate the table to the proper alignment because the segmentation mask of the render includes part of the background causing the median surface normal to point towards the camera. Furthermore in the third example, the retrieved table is incorrect. In the fourth example, the surface normal estimator for the target image is slightly inaccurate, causing the aligned table to point too much towards the camera. In the final example, the retrieved CAD model is incorrect, but the translation and rotation is quite successful, leading to a good overlap between the CAD model and the table in the target image.

We analyze some key examples of our system’s attempts at end-to-end topology retrieval in Figure 5.7. Due to the variety of moving parts in our system, failure in even one part can ruin the alignment. Alongside successes, we find some failure cases in our segmentation model, surface normal estimator, approach for translation alignment, and our retrieval sub-system. We find in those qualitative examples, however, that in most scenarios the rest of the system works as expected. This demonstrates that iterative improvements to the sub-components of our system can drastically improve performance, as we saw quantitatively with ground-truth surface normals.

As discussed in Section 5.2.3, if we apply more pre-rotations before alignment, our system would likely succeed on the first example in 5.7 since one of the pre-rotations will properly align the shelves in the tables. Extensions to Segment Anything like HQ-SAM (Ke et al., 2023; Kirillov et al., 2023) can improve the quality of segmentation masks. Better surface normal estimators will continue to appear over time, further augmenting the performance of our system. We can additionally try different attempts for aligning rotations, including depth estimation, point correspondences for surface normals, and other approaches. Overall, by introducing more and better tools, we can increase the robustness of our system and better compete with the state-of-the-art.

Chapter 6

Conclusion and Future Work

In this project, we developed a novel system for CAD retrieval and alignment using LLMs and Foundation Models with the Render and Compare approach.

We first designed **a system to retrieve CAD models that utilizes the latent knowledge of LLMs like GPT-4**. Our system used GPT-4 to plan properties to annotate for each CAD model and design questions to ask about the CAD models that correspond to those properties. We then used the LLM to write code that automatically annotates each CAD model by passing each question alongside a render of each CAD model into InstructBLIP, a visual-question-answering model, and storing the answers as a dataset. We demonstrated that GPT-4 is capable of processing the annotations of the CAD models alongside annotations of a new input image and can produce CAD models which it believes best correspond to the object in the input image. Furthermore, we demonstrated that **our retrieval system does not lose performance between datasets**, unlike trained methods that lose performance on unseen data. We additionally showed that **our system has strong potential to outperform state-of-the-art retrieval methods in future iterations**.

Our work also introduced **a new subsystem for aligning CAD models to an input image using foundation models and Render and Compare**. Our system used a series of foundation model tools that gather key information between an input image and a render of the current state of the CAD model, including pixel shifts, width and height information, and orientation through surface normals. We then used GPT-4 to generate code that orchestrates these tools together and updates the CAD model in Blender to iteratively align the CAD model with the input image. Our work showed that with the right prompting, **GPT-4 can generate complex 3D camera projection properties and can crucially write code using the Blender library that directly modifies CAD model meshes** based on the information it collects from the foundation-model-based tools. We again demonstrated that our system does not lose performance between datasets unlike state-of-the-art trained methods that lose

significant performance between seen and unseen data examples.

We finally evaluated our system end-to-end and show that our full system can successfully retrieve and align CAD models to some images. We further show that **with stronger image processing tools, our system can start to compete with state-of-the-art methods.**

Our project therefore contributes an exciting new approach to 3D data annotation and topology reconstruction using LLMs, foundation models, and the Render and Compare approach. We hope our work will provide a baseline for additional research into applying the knowledge of LLMs and the visual capabilities of foundation models to improve our approach and tackle more challenging 3D tasks.

6.1 Future Work

Alongside the potential improvements we have suggested in this paper, some initial extensions to our work include:

- **Robustness:** A key finding in this work was that, due to the variety of models and moving parts in our system, a single incorrect prediction by one part of our system can cause the entire prediction to be inaccurate. This fragility can be solved by giving the model a redundancy of tools for each step in its processing, such that the system can be robust to the failure of one tool by relying on the other redundant tools. Some examples of redundant tools to explore can include multiple segmentation models, generating additional surface normals from depth estimators, and multiple VQA models.
- **Better tools for orientation:** A large weakness in our alignment subsystem was its use of the median surface normal as a measure of orientation. The median surface normal has inaccuracies due to the differences in orientation of visible surfaces when you rotate objects. Furthermore, we found that even after aligning surface normals, objects can be rotated *around* the axis of the surface normal, which our system cannot handle. Approaches like aligning multiple surface normal correspondences can resolve this flaw.
- **Multimodal LLMs:** Our approach relies on multimodal foundation models to bridge the gap between images and the text that GPT-4 can process. With multimodal LLMs like GPT-4 with images, our system can be augmented to achieve better performance by avoiding the need for separate models and unifying information processing. We look forward to the release of these models to explore this direction further.

- **Multiple objects:** We currently designed our approach around retrieving and aligning CAD models for a single object in an image. At the same time, segmentation models allow for multi-object detection, making multi-object retrieval and alignment a straightforward extension of this work.

References

- Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., Ring, R., Rutherford, E., Cabi, S., Han, T., Gong, Z., Samangooei, S., Monteiro, M., Menick, J. L., Borgeaud, S., Brock, A., Nematzadeh, A., Sharifzadeh, S., Bińkowski, M. a., Barreira, R., Vinyals, O., Zisserman, A., and Simonyan, K. (2022). Flamingo: a visual language model for few-shot learning. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 23716–23736. Curran Associates, Inc.
- Anthropic AI (2023). Introducing claude.
- Bae, G., Budvytis, I., and Cipolla, R. (2021). Estimating and exploiting the aleatoric uncertainty in surface normal estimation. In *International Conference on Computer Vision (ICCV)*.
- Bae, G., Budvytis, I., and Cipolla, R. (2022). Irondepth: Iterative refinement of single-view depth using surface normal and its uncertainty. In *British Machine Vision Conference (BMVC)*.
- Blender Online Community (2018). *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam.
- Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J. Q., Demszky, D., Donahue, C., Doumbouya, M., Durmus, E., Ermon, S., Etchemendy, J., Ethayarajh, K., Fei-Fei, L., Finn, C., Gale, T., Gillespie, L., Goel, K., Goodman, N., Grossman, S., Guha, N., Hashimoto, T., Henderson, P., Hewitt, J., Ho, D. E., Hong, J., Hsu, K., Huang, J., Icard, T., Jain, S., Jurafsky, D., Kalluri, P., Karamcheti, S., Keeling, G., Khani, F., Khattab, O., Koh, P. W., Krass, M., Krishna, R., Kuditipudi, R., Kumar, A., Ladhak, F., Lee, M., Lee, T., Leskovec, J., Levent, I., Li, X. L., Li, X., Ma, T., Malik, A., Manning, C. D., Mirchandani, S., Mitchell, E., Munyikwa, Z., Nair, S., Narayan, A., Narayanan, D., Newman, B., Nie, A., Niebles, J. C., Nilforoshan, H., Nyarko, J., Ogut, G., Orr, L., Papadimitriou, I., Park, J. S., Piech, C., Portelance, E., Potts, C., Raghunathan, A., Reich, R., Ren, H., Rong, F., Roohani, Y., Ruiz, C., Ryan, J., Ré, C., Sadigh, D., Sagawa, S., Santhanam, K., Shih, A., Srinivasan, K., Tamkin, A., Taori, R., Thomas, A. W., Tramèr, F., Wang, R. E., Wang, W., Wu, B., Wu, J., Wu, Y., Xie, S. M., Yasunaga, M., You, J., Zaharia, M., Zhang, M., Zhang, T., Zhang, X., Zhang, Y., Zheng, L., Zhou, K., and Liang, P. (2022). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.
- Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago.
- Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P. (2023). Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.
- Dai, W., Li, J., Li, D., Tiong, A. M. H., Zhao, J., Wang, W., Li, B., Fung, P., and Hoi, S. (2023). Instructblip: Towards general-purpose vision-language models with instruction tuning. *arXiv preprint arXiv:2305.06500*.
- Deng, C., Jiang, C. M., Qi, C. R., Yan, X., Zhou, Y., Guibas, L., and Anguelov, D. (2022). Nerdi: Single-view nerf synthesis with language-guided diffusion as general image priors. *arXiv preprint arXiv:2212.03267*.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Gkioxari, G., Malik, J., and Johnson, J. (2019). Mesh r-cnn. *ICCV 2019*.
- Gümeli, C., Dai, A., and Nießner, M. (2022). Roca: Robust cad model retrieval and alignment from a single image. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*.
- Izadinia, H., Shan, Q., and Seitz, S. M. (2017). Im2cad. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2422–2431, Los Alamitos, CA, USA. IEEE Computer Society.
- Jia, C., Yang, Y., Xia, Y., Chen, Y.-T., Parekh, Z., Pham, H., Le, Q., Sung, Y.-H., Li, Z., and Duerig, T. (2021). Scaling up visual and vision-language representation learning with noisy text supervision. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4904–4916. PMLR.
- Jiang, Y., Ji, D., Han, Z., and Zwicker, M. (2020). Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Ke, L., Ye, M., Danelljan, M., Liu, Y., Tai, Y.-W., Tang, C.-K., and Yu, F. (2023). Segment anything in high quality. *arXiv preprint arXiv:2306.01567*.
- Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P., and Girshick, R. (2023). Segment anything. *arXiv:2304.02643*.

- Kuo, W., Angelova, A., Lin, T.-Y., and Dai, A. (2020). Mask2cad: 3d shape prediction by learning to segment and retrieve. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J.-M., editors, *Computer Vision – ECCV 2020*, pages 260–277, Cham. Springer International Publishing.
- Langer, F., Budvytis, I., and Cipolla, R. (2021). Leveraging geometry for shape estimation from a single rgb image. In *Proc. British Machine Vision Conference*, (Virtual).
- Li, D., Li, J., Le, H., Wang, G., Savarese, S., and Hoi, S. C. (2023a). LAVIS: A one-stop library for language-vision intelligence. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 31–41, Toronto, Canada. Association for Computational Linguistics.
- Li, J., Li, D., Savarese, S., and Hoi, S. (2023b). Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*.
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham. Springer International Publishing.
- Liu, F. and Liu, X. (2021). Voxel-based 3d detection and reconstruction of multiple objects from a single image. In *In Proceeding of Thirty-fifth Conference on Neural Information Processing Systems*, Virtual.
- Liu, M., Xu, C., Jin, H., Chen, L., T. M. V., Xu, Z., and Su, H. (2023a). One-2-3-45: Any single image to 3d mesh in 45 seconds without per-shape optimization. *arXiv preprint arXiv:2306.16928*.
- Liu, R., Wu, R., Hoorick, B. V., Tokmakov, P., Zakharov, S., and Vondrick, C. (2023b). Zero-1-to-3: Zero-shot one image to 3d object. *arXiv preprint arXiv:2303.11328*.
- Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Li, C., Yang, J., Su, H., Zhu, J., et al. (2023c). Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*.
- Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.
- Melas-Kyriazi, L., Ruppert, C., Laina, I., and Vedaldi, A. (2023). Realfusion: 360 reconstruction of any object from a single image. In *CVPR*.
- Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*.
- Müller, T., Evans, A., Schied, C., and Keller, A. (2022). Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15.
- OpenAI (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., and Lowe, R. (2022). Training language models to follow instructions with human feedback. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.
- Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019). DeepSDF: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*.
- Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.
- Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2021). High-resolution image synthesis with latent diffusion models. *arXiv preprint arXiv:2112.10752*.
- Shi, Z., Meng, Z., Xing, Y., Ma, Y., and Wattenhofer, R. (2021). 3d-retr: End-to-end single and multi-view 3d reconstruction with transformers. In *BMVC*.
- Sun, P., Kretschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi, A., Zhang, Y., Shlens, J., Chen, Z., and Anguelov, D. (2020). Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Sun, X., Wu, J., Zhang, X., Zhang, Z., Zhang, C., Xue, T., Tenenbaum, J. B., and Freeman, W. T. (2018). Pix3d: Dataset and methods for single-image 3d shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023a). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. (2023b). Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Wang, N., Zhang, Y., Li, Z., Fu, Y., Liu, W., and Jiang, Y.-G. (2018). Pixel2mesh: Generating 3d mesh models from single rgb images. In *ECCV*.
- Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. (2021). Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.
- Wu, C., Yin, S., Qi, W., Wang, X., Tang, Z., and Duan, N. (2023). Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.
- Zhou, Z. and Tulsiani, S. (2023). Sparsefusion: Distilling view-conditioned diffusion for 3d reconstruction. In *CVPR*.