Using Segmented Novel Views, Depth, and BERT Embeddings for Training in Robot Learning

Stanford CS224N Custom Project

Matt Strong
Department of Computer Science
Stanford University
mastro1@stanford.edu

Abstract

In this work, we develop a simple, end-to-end framework for generating 3D inputs (RGB+depth) via 3D Gaussian Splatting (3DGS), **conditioned on language** for **training** robotic imitation learning policies. In traditional robotics tasks, manipulators are only provided with camera (RGB) data and their own kinematics. They often lack the modality of both depth, which gives 3D structure to a scene, and language, which should condition robots on where to go and what to do. We present a simple and unique framework that adds these extra inputs to robotic imitation models. Our contribution is three-fold: firstly, we present a teleoperation data collection tool in the OmniGibson simulator for constructing Gaussian Splats; second, we train a 3DGS on a captured scene and condition novel views on CLIP embeddings, and finally, we train a simple robotic imitation learning model conditioned on language with pretrained BERT embeddings that improve learning. Our findings indicate that language should now be a staple input for training robot learning.

1 Key Information to include

- Mentor: Kaylee Burns
- External Collaborators (if you have any): None
- Sharing project: None

2 Introduction

As large language models become omnipresent, it is clear that the field of Natural Language Processing is at a stage where the idea of "foundational models" can be reasonably claimed. In robotics, can this be claimed? Recent works, such as Brohan et al. (2023) and Brohan et al. (2022) have trained large Vision-Language Models (VLMs) from internet data, with impressive results on out-of-distribution tasks, where a human can query a robot to perform an action, and it does a task (for example, "move the red bull to the H object") with high success rate. A recent work called Open-X Embodiment, by Collaboration et al. (2023), which collected robotic datasets from many labs across the world, demonstrates high success rate on novel tasks from different labs, outperforming most other models. The generalization abilities of these models, along with semantic understanding of objects and even the difference in prepositions, such as "above" and "below", is remarkable. However, these robotic models have billions of parameters, and as such require access to large-scale compute that requires a robotic manipulator running in real-time to have access to a good internet connection. In addition, the interpretability of these models is rather unexplored, possibly due to the lack of other researchers being able to ablate on these models. A final concern of these works is the importance of vision and language in these models and how exactly they work together. What representations are useful for robot learning?

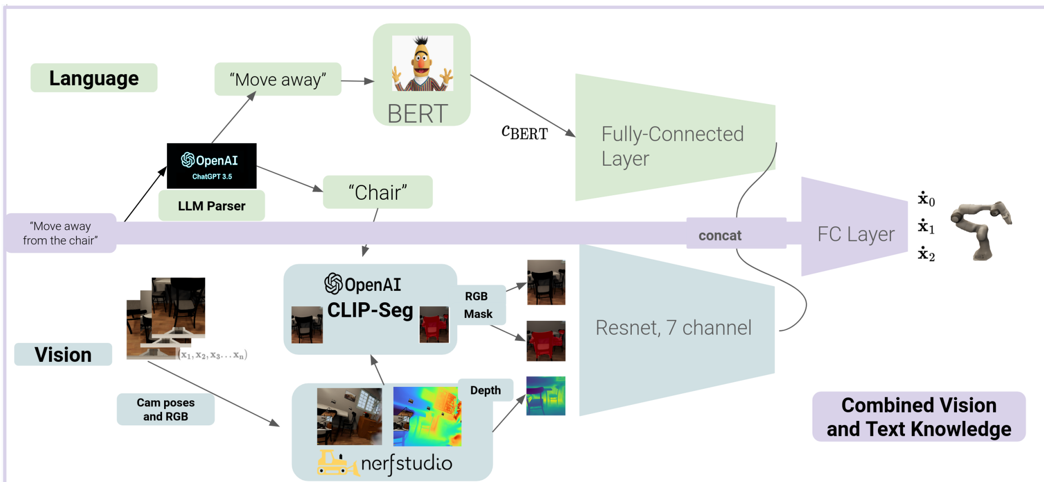


Figure 1: The proposed framework. Given a simple action-object phrase, a **LLM**, such as GPT, parses the phrase into the action and object. The action is sent to pretrained **BERT**, in which an embedding is computed, and sent to a fully connected layer. On the vision side, a user can teleoperate a robot in simulation and collect a series of images and camera poses, and create a 3D scene with **Nerfstudio**. Together with the object word, this scene can be queried for any pose x to produce an RGB and depth rendering. We segment out the object with **CLIP-Seg**, and provide the depth, object mask, and RGB to a **Resnet** encoder. The vision and text encoder are then concatenated and sent to a fully connected layer, which predicts an action, which we can optimize via trajectories in imitation learning.

The approach of this project is to take an opposite direction of these large foundation models. It is start from ground zero and explore not only the role of imitation learning for robotics, but exploring using novel view and depth synthesis from a technique called 3D Gaussian Splatting (3D-GS), which is becoming a emerging 3D visual representation for robotic manipulation. While this work does not advance baselines in the field, it will highlight the potential usefulness of simple language embeddings for robot learning, and using novel view synthesis.

3 Related Work

Robotic Simulators for Learning. It is still of interest in the robotics community to utilize photo-realistic and physically-accurate simulators for learning robotic policies. Simulators are easy to use and can serve as a foundation for researchers to prototype before extending to the real world. Simulators such as I-Gibson (Li et al. (2021)) and Habitat 2.0 (Szot et al. (2022)) lack the visual realism to transfer to real-world tasks. OmniGibson in Li et al. (2024) has emerged as a solution that supplants this realism with task diversity for human-centric tasks. OmniGibson uses Nvidia's Omniverse and PhysX to provide realistic physics simulation on rigid and deformable objects, and is our simulator of choice.

Visual Representations for Robotic Manipulation. Robots manipulators traditionally rely on RGB input to guide a task. Imitation learning, in which an expert demonstrates a task multiple times in a diverse manner and trains a robot on those trajectories, often leverages visual input and knowledge of robot state Osa et al. (2018). The work of Levine et al. (2016) and Chi et al. (2023) both learn policies from RGB data and robot pose, which are often sufficient for successful task completion.

NeRFs and Gaussian Splatting Emerging from the computer vision community, Neural Radiance Fields (NeRFs) in Mildenhall et al. (2020) and 3D Gaussian Splatting Kerbl et al. (2023) are two techniques that use only RGB images and corresponding camera poses to reconstruct 3D photorealistic models. Many robot manipulators give accurate camera poses and RGB input at the end effector, lending themselves for GS.

Language Embeddings for Robot Manipulation. Language Conditioned Robot Manipulation has now been thrust into the limelight of robot learning. It is now essential to use it in tasks that require any language. A first work addressing this was in Stepputtis et al. (2020), which used Glove word embeddings and a GRU to train a robot imitation learning model. However, this work does not use a 3D scene, relies on Faster-RCNN to find the probable object region from a sentence embedding, and uses a GRU cell for the sentence embeddings. Our work uses the intrinsic common sense of LLMs to get parse the action and object, to which we use the abilities of CLIP (Lüddecke and Ecker (2022), Shridhar et al. (2021)) to construct object masks. Lynch and Sermanet (2021) and Mees et al. (2022) integrates robotic learning for learning over unstructured data. A key insight is that pretrained language models can handle out of distribution synonyms, which we also present in this work. Other works are using LLM to generate commands for robots in Zhou et al. (2024).

3DGS and Language Embeddings for Robot Manipulation. 3DGS and language embeddings being used in robotic manipulation is a new method. Li and Pathak (2024) works on object manipulation with 3DGS without language. The most similar work to ours is ManiGaussian in Lu et al. (2024), which takes as input an RGB and depth image, and uses a dynamic Gaussian Splatting framework to learn behavior cloning policies. However, this work does not mention *embedding languag*e and understanding semantically similar actions.

4 Approach

The end goal of this project is query a robot to do something like "move away from a chair" and generate 3-D Cartesian velocity commands that are conditioned on the language, RGB, and depth commands. The overall framework is depicted in Fig 1. The steps consist of:

1. LLM Action Phrase Object Parsing.
2. OmniGibson Data Collection and 3DGS Training.
3. 3DGS Segmentation.
4. Imitation Learning Data Collection.
5. Imitation Learning.

LLM Action Phrase Object Parsing. Given a phrase "move to the chair", we query GPT-3.5 to give us the action ("move to") and object ("chair"). As humans parse sentences like this to understand the action and object, we let an LLM perform this task.

OmniGibson Data Collection and GS Training. We use OmniGibson to collect data for representing a scene and training our imitation learning model. We desire for a scene to be captured with sufficient RGB images (and corresponding camera poses) to represent a scene. We built on top of a keyboard teleoperation tool implemented in OmniGibson for Cartesian Control of a Franka Panda robotic arm. OmniGibson is a high-fidelity robotics simulator for learning. We then manually captured a scene by moving the robot around the scene. For each image we take to train 3DGS, we store the RGB image from the end-effector camera, the mask of the robot gripper, and the camera pose in the fixed world frame. Each image is saved and the paths and poses, as well as the camera instrinsics, and all are saved in a transforms.json file for Nerfstudio. During training of the Gaussian Splat, we mask out the pixels that are of the gripper in the loss. Once the GS is trained, we can use the model and query it given a pose $\mathbf{x}$ in the world frame.

3DGS Segmentation with Object. With a trained splat, we can query it and output a novel RGB and depth image. To segment the object of interest, we take the parsed object from GPT-3.5 and compute CLIP embeddings z_{RGB} and z_{object}. We then use the CLIP-SEG model (Lüddecke and Ecker (2022)) to output logits of the detected object, and run a Sigmoid function on the logits to have the output in the $(0, 1)$ range, and take the logits with a higher value than 0.4 as the object mask. This gives us an object mask, scene view, and scene depth for *any* view.

Imitation Learning Data Collection. We collect data with a P controller to train an imitation learning policy. Specifically, we want a simple example where a robot can move in different directions based on what it sees and a language command. We construct multiple trajectories in OmniGibson, run them, and store the command, RGB, robot end-effector pose, and P controller velocity command.

Imitation Learning Training. We train an imitation learning policy π.

$$\dot{x} = \pi(c, GS_{\text{color}}(x), GS_{\text{depth}}(x), m) \tag{1}$$

where the policy is conditioned on the novel view and depth $(GS_{\text{color}}(x), GS_{\text{depth}}(x))$ given pose x, with object mask m and command c. We stack the mask, depth, and RGB image and feed into a

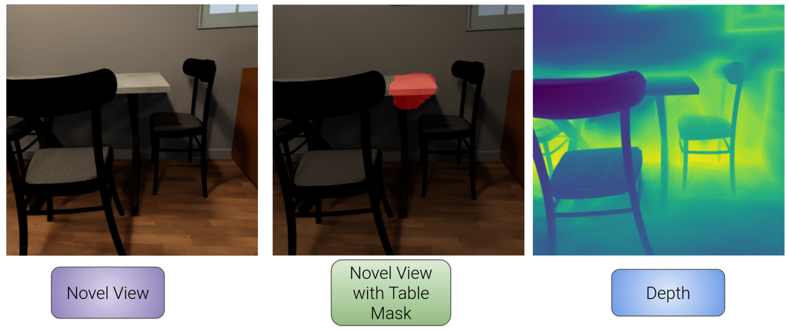


Figure 2: An example of the 3DGS pipeline – the novel view, the novel view with the table mask from CLIP-SEG, and the depth output from 3DGS.

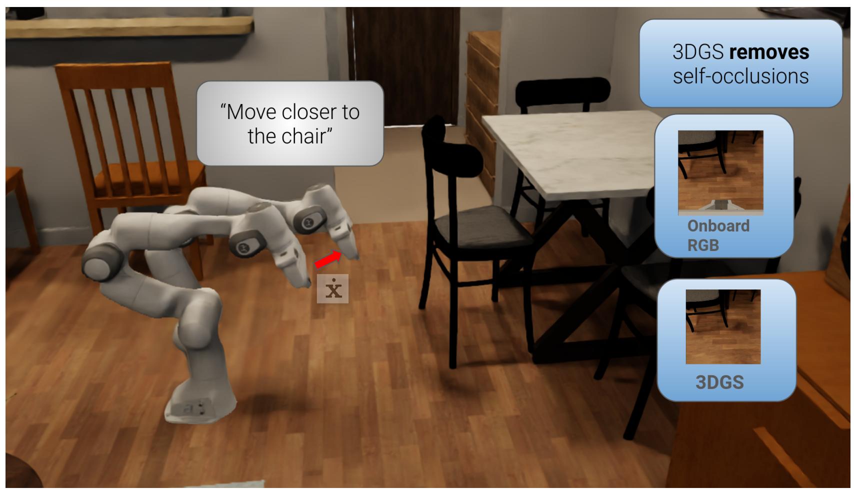


Figure 3: Setup in OmniGibson. 3DGS removes the self-occlusion of the robot gripper.

pre-trained Resnet. The action command c is fed into a BERT tokenizer to get text tokens, and then into the BASE base model (pre-trained), and we take the mean of the last hidden state of the input embedding. Finally, to downsize the embedding from dimension 768 to 64, we add a couple of linear layers and ReLU. The output of Resnet and the text layer is then concatenated, with a final regression head predicting end effector velocity. More details can be found in the Appendix.

5 Experiments

Experiments are conducted end-to-end, from a user-provided query to training an imitation learning model. A visual of the setup can be found in Fig 3.

5.1 Data

Action-Object Parsing: We use GPT-3.5 with no finetuning. However, the data for a simple evaluation is that we query GPT-4o to give a list of everyday objects and actions, and run through each combination of action and object to construct a phrase. We manually move confusing actions such as "push to" that do not make sense in the context of one object. The list of actions and objects is shown in Fig 4, where we have $22 \cdot 25$ actions and objects for a total of 500 unique prompts. The goal of this experiment is to assess the basic abilities of the parser. The input is the combined action and object, and GPT-3.5 should be able to parse the two. We perform sanitization on the output, including removing capitalization, punctuation, and spaces.

OmniGibson Data Collection for 3DGS and 3DGS Training: Our transforms.json on the main dataset consists of 81 manually taken images, a camera pose represented as a transformation matrix,

```
actions = [
    "move", "move to", "move close to", "push",
    "pull", "set up", "carry", "pick up", "place", "throw",
    "roll", "slide", "move away from", "push away",
    "grab", "carry to", "lift", "drop", "set down", "go to",
    "travel to", "go away from"
]

objects = [
    "chair", "table", "desk", "sofa", "lamp", "book", "cup",
    "bottle", "box", "pen", "phone", "laptop", "notebook",
    "pillow", "blanket", "backpack", "shoe", "shirt", "hat",
    "key", "watch", "remote", "mouse", "keyboard", "bag"
]
```

Figure 4: GPT 4o generated actions and objects with small human changes.

and robotic gripper mask image. We then train a Gaussian Splat on the scene on a RTX 4080 GPU, taking under 10 minutes. The end model is less than 150MB in size. Average inference time of GS on a novel camera pose takes 2ms on average, making it suitable for real-time robotic applications.

3DGS Segmentation with Object: We use the pretrained data from CLIP and CLIP-Seg without any fine-tuning. We find that the realism of OmniGibson and massive datasets CLIP and CLIP-Seg were trained on work well in our scenario.

Imitation Learning Data Collection: For a simple chair example, the data is collected with the Franka Panda being given a point in space $\mathbf{x_{near}} \in \mathbb{R}^3$ and $\mathbf{x_{far}} \in \mathbb{R}^3$, where the former is the goal *near* the chair, and the latter is the goal position is *away* from the chair. To generate expert trajectories, a simple P controller is used that moves between the near position and far position. Random noise is added to the two waypoints to construct two spheres that allows for varied trajectories (instead of overfitting to one trajectory). We can also augment our dataset by *reversing* the command and velocity vector – for example, "move to" and its action $\dot{\mathbf{x}}_\mathbf{1}$ has the reverse of "move away from" and action $-\dot{\mathbf{x}}_\mathbf{1}$. The collected data is the action command a, object o, RGB RGB, Cartesian velocity command $\mathbf{v}$, and the pose of the camera $\mathbf{x}$. Each trial is stored in a Pickle file and saved to avoid RAM overload. The dataset is simply read in by reading and adding all of the trials, then shuffling them randomly. The total size of the data is 11GB. In total, this consists of a dataset of 13108 state-action samples and 3278 samples with an 80 percent train-test split. Our second dataset does the same motion with multiple objects in the scene – wall, painting, window, chair, and table. The second dataset size is around 16000 total samples.

5.2 Evaluation method

Action-Object Parsing: We perform a simple experiment on action-object parsing where we evaluate the parser's abilities. With each of the 500 prompts, we compare GPT-3.5's parsed output with the correct action phrase and object. Concretely, we treat the entire phrase as a list of logits, and compute an action loss and object loss. Each loss is computed as a multi-class binary cross entropy between output logits of the action and object. For example, if the action phrase is the words "move to" in the phrase "move to the chair", the ground truth action logits would be

$$\mathbf{a_{gt}} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}.$$

We do the same for the object phrase and compute a loss as $\mathcal{L}_{action}, \mathcal{L}_{object}$. We report the average loss of object and action phrase over the 500 trials. We explored ablating the experiment with small changes to the prompts but noticed a) minimal change or b) catastrophic change. The system context is listed in the appendix for brevity, but at a high level we prompt GPT to give us separated actions and phrases.

Imitation Learning: The chair scene, although simple for humans, is non-trivial for a robotic manipulator. We include three evaluation metrics: training and testing loss (mean-square-error) on the output velocities, cosine similarity between predicted and output velocities, and the amount of samples with the correct signs in all dimensions. Due to time constraints, we were not able to deploy on a robot in real-time; however, we still believe these results demonstrate promise.

5.3 Experimental details

Action-Object Parsing: We use GPT-3.5 from the OpenAI API and send each phrase to parse to the API every 0.5s to avoid throttling. The process takes in total 10 minutes.

OmniGibson Data Collection for 3DGS and 3DGS Training: The data collection takes about 5 minutes to collect data, and we use the default Gaussian Splatting implementation in Nerfstudio, called Splatfacto, for 30000 timestamps.

Imitation Learning Data Collection: We collect 20 trials of the robot moving between performing the "move to chair" and "move close to chair". This process takes only around 10 minutes. For the second dataset, we do 4 trials of moving back and forth near 5 objects of window, wall, chair, table, and painting, taking 13 minutes

Imitation Learning Training: We train our model with a train batch size of 32, 768 embedding size for the BERT model, the Adam optimizer with a learning rate of 0.001, and 5 epochs on the 4080 GPU. The loss function is mean square error between the expert velocities and predicted velocities. Training the full model for 5 epochs takes around 20 minutes.

5.4 Results

Action-Object Parsing: After running this experiment the average action loss is 0.3953, and the average object loss is 0.3132. A perfect score would be 0.3132 on both.

Imitation Learning Training Results. On the chair example, we first ablate using no language as a baseline. We check both using RGB from the robot and 3DGS. The results, shown in Table 1, are extremely poor – the mean square error is high for a robotic system, and there are few samples that get the correct signs for predicted velocities, which is poor due to the fact that the expert velocities are reasonably far from 0.

From this baseline, we then perform an ablation on using RGB images vs. using the novel view RGB images from GS. This consisted of training the model with the text commands – we simply train on "move to" and "move away from" as the language commands, instead of the full suite of synonyms, which are mentioned in the next experiment. The results are reported in Table 2. Using Gaussian Splatting for novel views notably improves the performance on the test set, as the lack of robot mask allows for the model to better identify the entire scene.

Metric	Robot RGB No Lang	**GS** RGB No Lang
Train Loss	0.0038	0.0040
Train Sign Correct	39.71%	39.45%
Test Loss	0.0040	0.0041
Test Sign Correct	39.84%	36.88%

Table 1: Results of using no language command, after 5 epochs, averaged over 5 trials.

Metric	Robot RGB	**GS** RGB Lang
Train Loss	0.00077	**0.00088**
Train Sign Correct	**93.57**%	92.59%
Test Loss	0.0037	**0.0007**
Test Sign Correct	86.25%	**90.44**%

Table 2: Results of using a binary language command, after 5 epochs, averaged over 3 trials. Best results are **bolded**.

The next experiment consists of integrating the entire framework, which includes the mask and depth, on two scenes. The first one is the same chair scene; however, this time the model is trained on a variety of *synonym phrases* to "move to" and "move away from". During test time, the model is evaluated on **unseen phrases** with the same object.

In Table 3, we take the last epoch of the first scene on the full model. The results are above 90% signs of every component of the velocity vector being the same; additionally, the cosine similarity on the test set is 0.97, indicating that the model learns a sense of direction in the scene. The second scene

```
SYNONYMS_MOVE_TO = ["move to" ,"go to", "move to",
                    "head to", "travel to",
                    "go over to", "move over to",
                    "navigate to", "proceed to",
                    "approach", "advance to", "approach", "get to",
                    "go up to", "move close to", "move towards", "go towards",]

SYNONYMS_MOVE_AWAY = ["go away from", "move apart", "back away from",
                      "retreat from", "diverge from", "move away from", "get away from", "back off from", "move back from",
                      "move far from"]
```

Figure 5: Synonyms

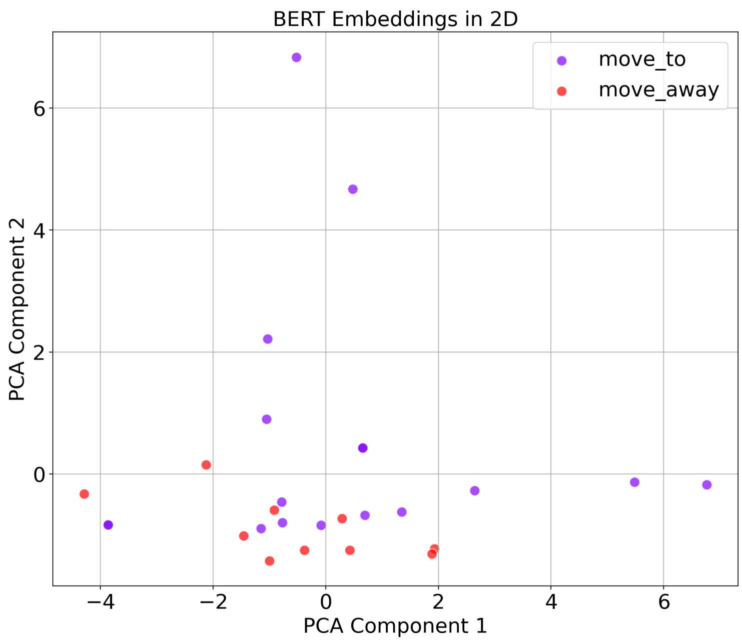


Figure 6: Visualization of BERT embeddings of words with the same meaning as "move to" and "move away". PCA is used to output human-readable embeddings.

consists of **multiple objects**. For brevity, we report the test results here – the test loss was 0.0011, the amount of signs correct was 68.17%, but importantly, the average cosine similarity was 0.964. Both of these metrics are comparable to the one object case. Interestingly, the model learns that direction is based on object; when the robot is reset, the object informs the direction, which is learned. Future work will address ablating over depth, which will become more useful when *interacting with objects*.

Metric	Full Model
Train Loss	0.0010
Train Sign Correct	91.50%
Test Loss	0.0009
Test Sign Correct	92.04%
Test Cosine Similarity	0.97

Table 3: Results of using a binary language command with synonyms, after 5 epochs.

6 Analysis

Action-Object Parsing: The action object parser was effectively able to parse simple action phrases and objects. Even with action phrases with 1 to 3 words, it was still able to classify the action correctly. While the object classification was perfect, the most common error in the action parsing was adding a "to" in front of the phrase, which could be dealt with in future work.

Imitation Learning Training: The simple chair example reveals the value of using the views from GS and adding extra examples of phrases to condition the model on. To understand the value of

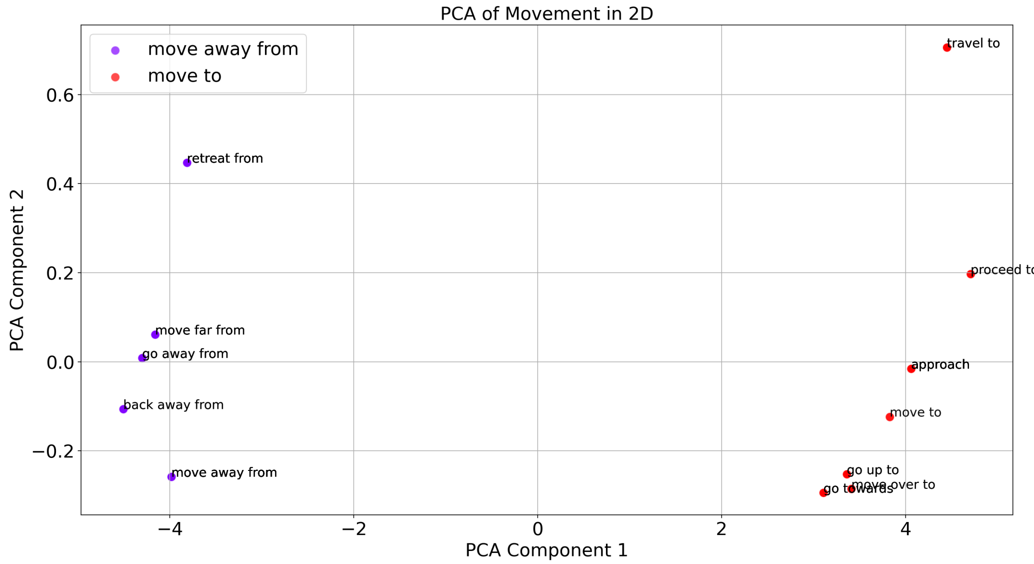


Figure 7: Language Embeddings after the Model Fully Connected Layers.

pretrained language models for the task, we include a visualization of the BERT embeddings of the "move away" and "move to" dataset. When we visualize the BERT embeddings of the two classes, there is not a straightforward decision boundary; however there seems to be some similarity in the "move to" words and "move away" ones. However, when we take the output of the language encoder (adding a fully connected layer to the BERT embedding), there is a clear, even linear, decision boundary between words similar to "move away from" and "move to". In fact, these phrases were never seen, which demonstrates the ability of BERT being able take care of a significant amount of semantic meaning between the phrases, where our model separates the two by a large margin. Even in this simple example, language embeddings are a clear necessity. Next, the similarity in performance between the simple and multi-object experiment indicates some value of the object being encoded into the mask. We note that the low sign correct is due to control actions in the P controller trajectories often being close to 0, so we rely on the other two metrics to inform our analysis. The object masks are motivated, GS novel views are motivated, and text is as well.

7 Conclusion

In this work, we proposed a simple framework for semantic novel views, depth without a depth camera, and language for training an imitation learning model. By collecting data in simulation to train a Gaussian Splat, outputting novel RGB views and depth for free, using the success of pretrained language models and visual-language models, and a simple network architecture, we are able to demonstrate the importance of conditioning on language and novel views. One limitation of our work is the experiments – they did not involve manipulation. We believe **depth** will be even more useful for grasping tasks that benefit from geometry of objects, which is provided by Gaussian Splatting's depth. Future experiments will extend the scene to involve manipulation tasks a broader group of language commands and to deeply understand the value of depth in a scene. However, this work is a promising first step in that direction, and, to the best of our knowledge, is the first work to use novel views and depth from 3DGS, CLIP-based object masks, and action command embeddings to train an imitation learning model end-to-end.

8 Ethics Statement

There is a strong ethical motivation to use language as control input in robotics. Teleoperation of robotic manipulators, which is one of the common methods of control, often requires physical dexterity of humans and a high amount of experience, whereas domestic robots are often used by people with mobility and dexterity challenges, presenting a challenge for these users. With language, there can now be intuitive controle, where people simply need to be able to speak or type instead of having the ability to operate a joystick. However, there is a concern with these physical actions being guided on language. This means that such systems need to be robust as to avoid physical harm. In this case, **a robot is directly conditioned on language embeddings**. If the language command is out of distribution of the test set, the robot may perform erratic and dangerous actions. This is one ethical concern. However, a mitigation strategy is that because safety and robustness in control are well-studied in robotics, it might reasonable to implement simple methods that guard against errors such as spilling liquid, dropping an object, or even colliding with a human. These safeguards could be after the output of a model; only suitable actions will be sent to a robotic manipulator.

One advantage of our work is that this work leverages depth information to inform robot velocities, even when depth sensors are not available. This is thanks to the abilities of Gaussian Splatting, which also can render depth along a ray. This can replace a depth sensor, which often requires spending hundreds of dollars for an effective sensor. However, one unintended impact of our method is that it requires heavy computation (running inference on the model, Gaussian Splatting, CLIP-Seg, and the LLM parser), which comes with it environmental and energy costs. Carbon footprint is a legitimate concern of using larger and larger models. While our method justifies each step of the pipeline, the energy usage is still high. A possible mitigation strategy for our method's compute is to gridify the scene a robot is in, turning it from a continuous space to a finely discrete one. We would lose a small amount of accuracy, but this would allow for storing the renderings of a scene apriori, reducing the need for an extra inference step.

Last, any system that incorporates language as input and movement leads to a risk of users not proficient in the input language. Such NLP systems, because many of the authors speak English and/or are required to write English for research, are inherently biased towards English. A naive first step for mitigation is to translate from the source language to English; however, it may be worth training a robot policy on the source policy; some languages are more compact in certain ways than others. The interior layout of home environments is also highly dependent on culture; the model in this work was trained in a Western style home that might not be suitable for other homes. A mitigation for this is to use the OmniGibson environment and generate new diverse environments, which is straightforward with the API.

References

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alex Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. 2023. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *arXiv preprint arXiv:2307.15818*.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. 2022. Rt-1: Robotics transformer for real-world control at scale. In *arXiv preprint arXiv:2212.06817*.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. 2023. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*.

Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Felipe Vieira Frujeri, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guangwen Yang, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi "Jim" Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minho Heo, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, Ning Liu, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R Sanketi, Patrick "Tree" Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundaresan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Mart'in-Mart'in, Rohan Baijal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shuran Song, Sichun Xu, Siddhant Haldar, Siddharth Karamcheti, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vincent Vanhoucke, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiangyu Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yansong Pang, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yongqiang Dou, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen Zhang, Zipeng Fu, and Zipeng Lin. 2023. Open X-Embodiment: Robotic learning datasets and RT-X models. `https://arxiv.org/abs/2310.08864`.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4).

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies.

Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, Andrey Kurenkov, C. Karen Liu, Hyowon Gweon, Jiajun Wu, Li Fei-Fei, and Silvio Savarese. 2021. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks.

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Martinez, Hang Yin, Michael Lingelbach, Minjune Hwang, Ayano Hiranaka, Sujay Garlanka, Arman Aydin, Sharon Lee, Jiankai Sun, Mona Anvari, Manasi Sharma, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Yunzhu Li, Silvio Savarese, Hyowon Gweon, C. Karen Liu, Jiajun Wu, and Li Fei-Fei. 2024. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation.

Yulong Li and Deepak Pathak. 2024. Object-aware gaussian splatting for robotic manipulation. In *ICRA 2024 Workshop on 3D Visual Representations for Robot Manipulation*.

Guanxing Lu, Shiyi Zhang, Ziwei Wang, Changliu Liu, Jiwen Lu, and Yansong Tang. 2024. Manigaussian: Dynamic gaussian splatting for multi-task robotic manipulation. *arXiv preprint arXiv:2403.08321*.

Corey Lynch and Pierre Sermanet. 2021. Language conditioned imitation learning over unstructured data. *Robotics: Science and Systems*.

Timo Lüddecke and Alexander S. Ecker. 2022. Image segmentation using text and image prompts.

Oier Mees, Lukas Hermann, and Wolfram Burgard. 2022. What matters in language conditioned robotic imitation learning over unstructured data.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*.

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel, and Jan Peters. 2018. An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1–2):1–179.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. 2021. Cliport: What and where pathways for robotic manipulation.

Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. 2020. Language-conditioned imitation learning for robot manipulation tasks.

Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. 2022. Habitat 2.0: Training home assistants to rearrange their habitat.

Hongkuan Zhou, Xiangtong Yao, Yuan Meng, Siming Sun, Zhenshan Bing, Kai Huang, and Alois Knoll. 2024. Language-conditioned learning for robotic manipulation: A survey.

A Appendix (optional)

Below is the Python content that is sent to the OpenAI API for GPT:

```python
def get_action_and_object(self, phrase):
        content_to_send = f"I have the phrase '{phrase}'. What is the entire action, and what is the object?"

        completion = self.client.chat.completions.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a word smith. You have been given a phrase and you must identify the full action phrase and object. Give the answer in the format 'action: <action>,\n object: <object>'."},
            {"role": "user", "content": f"{content_to_send}"}
        ]
        )
```

```
        parsed_message = completion.choices[0].message.content.split
            ('\n')
```

A.1 Model Parameters and Architecture

GPT-3.5o: No change or fine-tuning.

Splatfacto: GS does not have a deep learning model, however we will note that each Gaussian is parameterized by a color, opacity, scale, and rotation.

Main Imitation Learning Model

Below, we detail the architecture of our Imitation Learning Model.

A.1.1 Vision Encoder.

To encode the stacked channels of RGB, depth, and object mask, we use the pretrained torchvision Resnet-18, which is finetuned on our dataset. We modify the first convolution layer to take 5 channel input and remove the final fully connected layer to use it as an encoder.

A.1.2 Text Encoder.

From the outputted BERT embeddings of using the BERT base model, we add a few simple fully connected layers:

- **Linear1:** 768 to 128 dimensions.
- **ReLU:** Simple ReLU layer.
- **Linear 2:** 128 to 64 dimensions

A.1.3 Fusion and Regression Head.

We concatenate the output of the vision and text encoder, feeding into the following fully connected layers:

- **Linear1:** 128 (each encoder outputs a 64 dim vector in the last dimension) to 128 dimensions.
- **ReLU:** Simple ReLU layer.
- **Linear 2:** 128 to 64 dimensions
- **RELU:** Simple ReLU layer.
- **Linear 3:** 64 to 3 dimensions, with the output being a 3 dimensional velocity of the end-effector.

A.2 Example Transforms.json File For Nerfstudio.

Nerfstudio, the software that allows for the training of radiance fields, requires a transforms.json file that defines the camera parameters, poses, and image paths to each camera. An example is seen in Fig 8.

A.3 OmniGibson Visualization.

A sample view of OmniGibson and using the data collection tool is shown in Fig. 9, where the user can see the scene from a first and third person perspective. Each time they press the key "x", a new entry is added to the transforms.json mentioned above.

```
{
    "w": 720,
    "h": 720,
    "fl_x": 584.1088067078963,
    "fl_y": 584.1088067078963,
    "cx": 360.0,
    "cy": 360.0,
    "k1": 0,
    "k2": 0,
    "p1": 0,
    "p2": 0,
    "camera_model": "OPENCV",
    "frames": [
        {
            "file_path": "gs_images/gs_img0.png",
            "depth_file_path": "gs_depths/gs_depth0.png",
            "seg_file_path": "gs_segs/gs_seg0.png",
            "transform_matrix": [
                [
                    0.03526493347184795,
                    0.9041321773801075,
                    -0.42579500970905293,
                    0.2704782485961914
                ],
                [
                    -0.9992937765285196,
                    0.03743259093888035,
                    -0.0032786166247345816,
                    0.001954717095941319
                ],
                [
                    0.012974307634539689,
                    0.425609923476308,
                    0.9048137158442593,
                    0.5348368287086487
                ],
                [
                    0.0,
                    0.0,
                    0.0,
                    1.0
                ]
```

Figure 8: The beginning of the transforms.json. The camera instrinsics are provided in the simulaltor, and we also provide paths to images, depths and segmentation (optional), and the world to camera transform matrix.

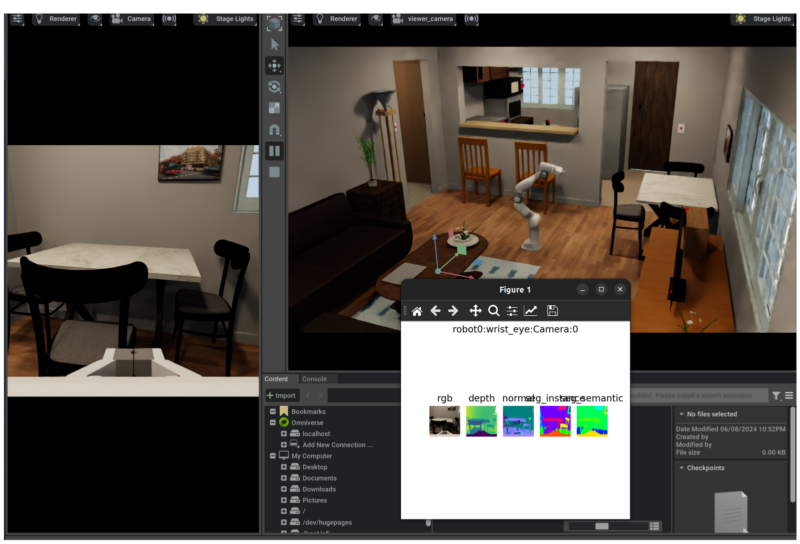


Figure 9: Example Setup in OmniGibson.

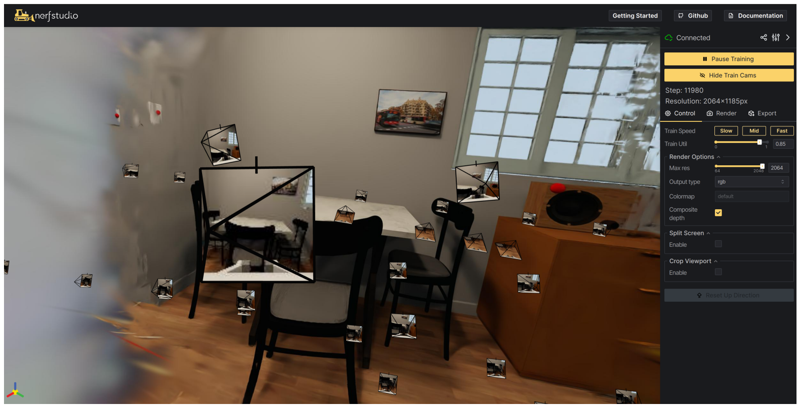


Figure 10: Nerfstudio RGB rendering

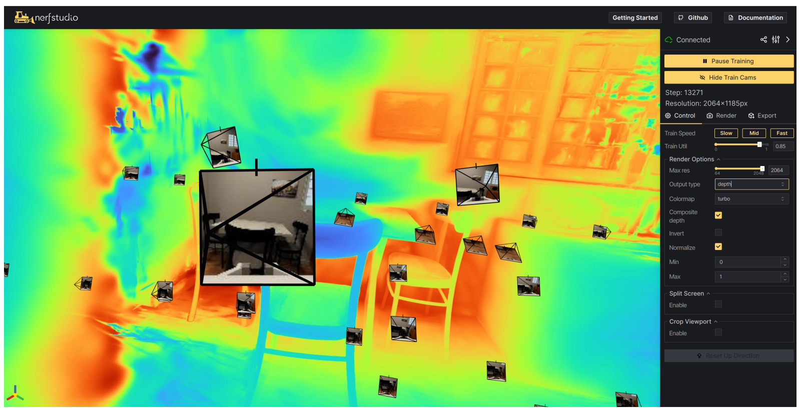


Figure 11: Nerfstudio depth rendering

A.4 Nerfstudio Visualization.

An example visualization of training a Gaussian Splat is shown. The trainer can view the training cameras and see how closely the scene is reconstructed. Note that we are able to mask out the robot gripper entirely in the scene! We also show a depth rendering at the same camera pose.

A.5 Sample CLIP Embedding Visualizations.

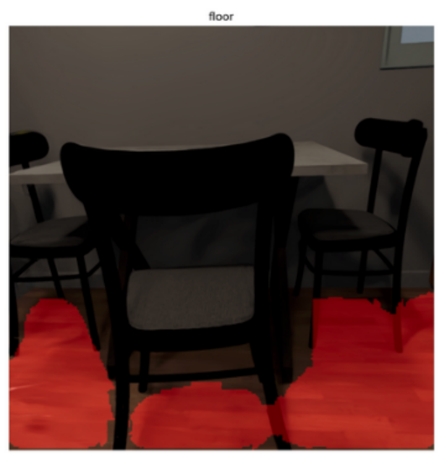
We show visualizations of **novel views** and the CLIP-Seg outputs.

Even a painting can be segmented out!

(a) Novel View

(b) Chair

(c) Wall



(d) Floor

(e) Window

Figure 12: Example CLIP-Seg Outputs.

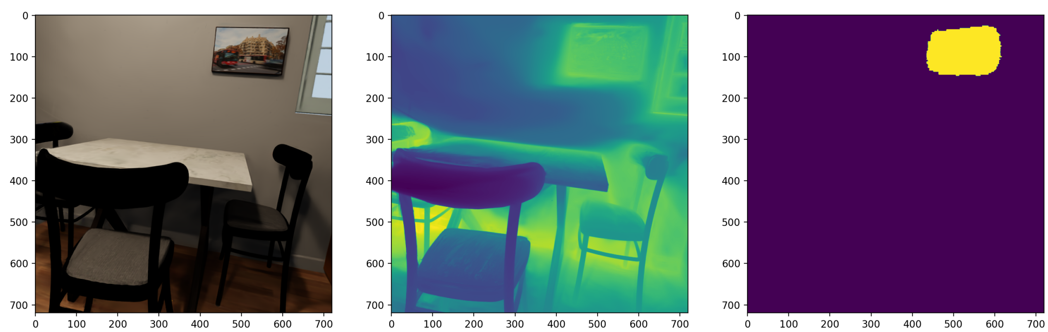


Figure 13: Novel View, Depth, and Segmented "Painting"