RoboNLP: Task Sequencing in Robotics using Large Language Models

Felix Su su000111@umn.edu
Joseph Lisk liski003@umn.edu
Jiawei He he000065@umn.edu
Franklin Xavier antho264@umn.edu

1 Introduction

We aim to let everyday users instruct robots using normal speech so that the robot can complete multi-step tasks like stacking, placing, and sorting objects. Nowadays, integrating human-robot interaction (HRI) into automated systems has become increasingly vital since robots are now expected to assist humans with complex, everyday tasks. Despite advancements in robotic capabilities, interaction with robotic manipulators remains constrained by the need for intricate programming and a lack of intuitive, non-manual control interfaces. These barriers limit the adaptability of robots in changing environments for tasks that require autonomous, sequential actions, such as assembling or collecting multiple objects without manual control to free humans' hands. Today, programming a robotic arm even to stack three blocks can require dozens of lines of code and expert knowledge.

A promising approach to address these challenges is to use human speech as a natural, accessible means of communication with robotic systems. By enabling users to control manipulators through conversational language, we can reduce the complexity of traditional programming interfaces and make robotic assistance available to a broader audience, including non-experts. However, the challenge lies in developing an interface that allows users to intuitively assign sequential tasks in natural language while the robotic manipulator can accurately interpret and execute these commands. With advancements in computational power and the development of large language modeling technologies, the enhanced natural language understanding and advanced logical reasoning capabilities of large language models (LLMs) present a viable solution to this challenge. Hence, our project aims to leverage the natural language comprehension and semantic analysis capabilities of large language models alongside the adaptable trajectory control of a path planner to develop an intuitive approach to manipulator teleoperation.

Specifically, we propose a framework that integrates an LLM-based action sequence planner with a trajectory planner to enable operators to command a robotic manipulator using natural human speech. Within this framework, LLMs are tasked with generating a sequence of simple actions. We evaluated eight LLMs of varying parameter sizes on tasks from the robotic manipulation benchmark RLBench (James et al., 2019), employing two prompting strategies to assess their ability to generate effective action sequences. These sequences were subsequently validated in the Gazebo simulation environment, where a simulated Kinova arm was used to perform the tasks, providing insights into the effectiveness of the proposed approach.

Our work is novel in a way that we are systematically analyzing the impact of LLM parameter sizes on their ability to generate task sequences for robotic manipulators. This study provides critical insights into the scalability and effectiveness of LLMs in robotics applications.

2 Related Work

The use of Large Language Models (LLMs) in robotics, particularly in path planning and task execution for robotic manipulators, has garnered significant interest in recent years. This paper (Wang et al.) mentions that Traditional Task and Motion Planning (TAMP) in robotic approaches heavily relies on expert-designed interfaces, which makes the system domain independent and less adaptable to novel tasks in real-world settings. Hence, this paper introduces an LLM^3 framework that addresses this limitation by using a pre-trained LLM to generate symbolic actions and corresponding continuous motion parameters without requiring domain-specific designs. They used the GPT-4 Turbo as

the LLM planner to generate symbolic action sequences based on the task description provided by the human in natural language. Then, continuous parameters for each action are generated and fed to the BiRRT, the motion planner, to execute the action. The LLM framework also incorporates motion failure feedback to allow the LLM to update the action sequence or parameters iteratively when the motion planner gives failure feedback due to collision or unreachability.

Also, in this paper (Vemprala et al.), the authors explored frameworks and prompting strategies and a function library for utilizing ChatGPT to generate primitives for typical robotics tasks. The function library that the paper introduces contains action functions like pick_up("eggs"). The authors note that the advantage of the function library is that it allows ChatGPT to generate code without needing to account for specific coding libraries, tools, and frameworks. The paper then explains ways to create more accurate output from ChatGPT via prompt engineering, such as including constraints, the environment, the current state, goals and objectives, and solution examples. Finally, the paper proposes that the human user evaluates the code generated by ChatGPT and provides feedback for ChatGPT to improve the code iteratively before deploying the final version of the code.

Additionally, the paper (Ahn et al.) introduces the SayCan framework, which integrates large language models (LLMs) with robotic systems to perform real-world tasks based on high-level language instructions. Traditional LLMs lack physical experience, which limits their effectiveness in providing actionable instructions for robots. SayCan addresses this by grounding LLMs in robotic affordances, meaning the LLM's suggestions are constrained to actions that are feasible within the robot's physical capabilities and the environment's context. SayCan uses pretrained skills and reinforcement learning (RL) to develop value functions, which quantify the likelihood of successfully performing a given task in a specific state. These value functions help the system select appropriate actions by combining the LLM's high-level task knowledge with the robot's practical capabilities. This framework enables robots to interpret and execute long-horizon, abstract tasks by breaking down complex instructions into sequential, contextually relevant steps.

The existing works discussed above often rely on manually defined symbolic planners or require large amounts of hand-tuned code. Others fail when instructions are phrased in unconventional ways or require adaptation to new tasks. Our approach attempts to remove these limitations by leveraging LLMs that can understand natural language variability and quickly adapt to new environments defined in the system prompt.

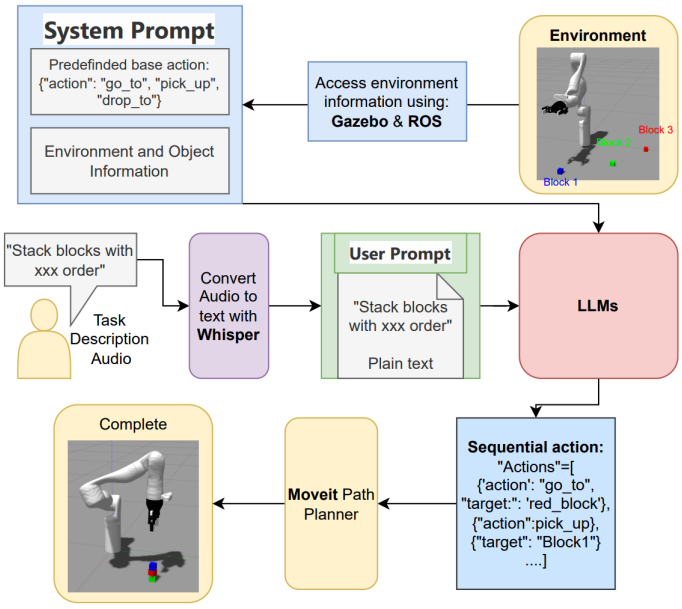

Figure 1: The pipeline of the proposed framework

Unlike prior work that focuses solely on language-conditioned policy generation, our work integrates speech transcription (for natural HRI), LLM-based textual reasoning (for flexible task specification), and a standardized interface to a trajectory planner. This three-step integration is novel in its simplicity and adaptability.

While previous works have explored the use of LLMs for generating task sequences, none have systematically analyzed the impact of model parameter size on task planning accuracy and reliability in robotic applications. Our study aims to address this gap

Also, Existing LLM-based planners have limited accessibility for non-technical users. We aim to solve this by providing a speech-to-action pipeline.

3 Approach

In this work, we propose a novel framework that utilizes natural language comprehension and the logical thinking ability of LLMs to automate the sequential action planning process for manipulators to complete tasks guided by human natural speech. The whole pipeline of the framework is shown in

One-shot Prompt Example
User_prompt: """You can perform the following actions: go_to(target): Controls the end effector of the robot arm to move to ..... pick_up(target): Opens the gripper, then controls the end effector of ..... drop_to(target): Assumes the gripper has already picked up an object ..... If I ask "Stack the blocks in the following order - blue_block at the bottom,\n red_block in the middle, ... Your response should be a list in the following format: [{'action': 'go_to', 'target': 'red_block'}, {'action': 'pick_up', 'target': 'red_block '},....] Remember, only give the action list as an example; I don't need any other suggestions\n:.. Pick up red block,"""
system_prompt = """ You are a robot arm action planner working with blocks in the environment. The environment includes three blocks as the objects: a red block labeled as number one,..... You can also consider the location of an object as the location of a target"""
One-shot + Chain-of-Thought Prompt Example
User_prompt: """You can perform the following actions: .....(same with One-Shot prompt) First, consider how you would perform the actions. Then, think about how you would format the action list in Python. Finally, give the action list. .....(same with One-Shot prompt) Pick up red block."""
System_prompt: .....(same with One-Shot prompt)
Expected Output from LLMs
[{'action': 'go_to', 'target': 'red_block'}, {'action': 'pick_up', 'target': 'red_block'}]

Table 1

figure 1. The manipulator we used in this work is Kinova (Gen3), a redundant seven degree of freedom (DoF) arm. Due to the hardware limitation, we spawn the manipulator and other interactive objects in the Gazebo simulation environment and use the Robot Operating System (ROS) to communicate with the proposed system. We use ROS messages to keep reading the position and orientation information for the end-effector and all the interactive objects to allow the system to be aware of the changes in the environment and make the framework able to work dynamically. Note that, in the current work, the name and the number of interactive objects are predefined, and we used human language to introduce the environment information to the LLMs in the system prompt as shown in table 1, including who they are (a robotic action planner) and what objects and other special locations are in the environment.

On the other hand, our framework allows the operator to start recording the description of the task that they want the manipulator to complete with natural English speech by pressing the Space key on the normal keyboard, and the recorded audio will be converted to text by Faster Whisper (SYSTRAN, 2024), an open-source transcription model implemented by a transformer models library, CTranslate2 (OpenNMT, 2024). The converted description text will be concatenated with the other part of the user prompt and fed to LLMs.

While it is true that large language models are now powerful and are iterating at a rapid rate, so far, having the model directly generate the code needed to accomplish the task or generate trajectories is still too complex and unreliable in the robotic domain. So, we simplified the work for LLMs in our framework. We provide some predefined basic action functions to the LLMs and ask them to use these basic actions like puzzle pieces to generate an action sequence based on the task. We have developed three basic actions:

1. **pick_up(target)**: Opens the gripper, then controls the end effector of the robot arm to approach the target object vertically from above. Once the end effector reaches the target, it closes the gripper to grasp and lift the object securely.

2. **go_to(target)**: Controls the end effector of the robot arm to move to the target object and position itself directly above the target, maintaining a vertical alignment.

3. **drop_to(target)**: Assumes the gripper has already picked up an object using the pick_up(target) action. Controls the end effector of the robot arm to approach the target object or location vertically from above. Once positioned, the gripper opens to release the object, and the end effector retracts, leaving the object at the target location.

These descriptions of each basic action are included in the user prompt template as the format in table 1. The last line of the user prompts the operator's speech, which describes what task they want the manipulator to execute. This part is concatenated from the output of the Whisper as described above. The LLMs take the system and user prompt as the input and are only asked to output the desired action sequence as a Python list. To execute this action sequence, we implement a Python script that reads this list and executes corresponding action functions. These functions utilize the Moveit Path Planner, which uses rapidly exploring random trees (RRT) to find an executable trajectory based on the action type and target. These trajectory commands will be sent through ROS to the manipulator, which will finally execute and complete the task.

In the initial idea of this framework, we want to integrate the LLMs and Variational autoencoder (VAE) to directly generate trajectories for the manipulator to execute. However, we found out it is time-consuming to collect data to train the VAE, and the LLMs are not reliable when they are asked to generate continuous trajectories, and it is hard to let the model understand all the spatial information. Hence, we decided to run the current approach at a very early stage since this problem is predictable. The other problem we encountered was we initially wanted to make the responses from the LLMs in JSON form in order to unify the output format. However, it actually makes it hard to write a Python script to read the output, so we decided to make the response in Python list format. Additionally, one anticipated challenge was ensuring the LLM produces a strictly formatted action sequence. To address this, we carefully engineered prompts and tested multiple prompting strategies (one-shot, one-shot+chain-of-thought, etc.) and enforced format constraints in the system prompt.

To evaluate the role of LLM parameter size in task sequence planning, we tested eight LLMs with varying parameter sizes, ranging from 1 billion to 70 billion parameters. This systematic evaluation provides unique insights into the relationship between model parameter size and task execution accuracy in robotic systems.

This approach benefits non-expert users who are not trained in robot programming, and disabled individuals who want to control assistive robots via speech. In industrial and domestic settings, it could drastically reduce setup time and make robots more accessible and flexible.

We also hypothesize that LLMs, trained on large-scale text data, can infer the correct sequence of actions even from vague or indirect instructions. This adaptability should reduce the need for expert-designed planning systems.

4 Experiments

To evaluate the usability of the proposed framework, we pick four tasks from a robot learning benchmark, RLBench (James et al., 2019), to test whether our framework can use sequential actions to complete these tasks. These tasks are Picking, Building a pyramid, Placing, and Stacking, as shown in the subfigure (b to e) in Figure 2. The reason for choosing these four tasks is they are possible to complete by the basic actions we develop for the framework, and we focus on researching whether a correct action sequence can be generated by the LLMs and correctly executed. Hence, we ignore tasks requiring more basic actions, such as sliding blocks to a certain area.

In RLBench, each task has three different descriptions. All sentences mean the same but are written differently. As described in the last section, we use them as the operator speech to form the user prompt. We then manually design a set of ground-truth action sequences to which we expect the LLMs to respond with equivalent sequences. Hence, three different descriptions mean three testing trials for each task on one LLM model. We would record the success rate of correctly generating action sequences for each task. In order to have a deeper analysis of what aspects will affect the performance of LLMs in this action sequence generation task, we evaluated eight LLMs of varying parameter sizes in this experiment. Moreover, to explore how prompting

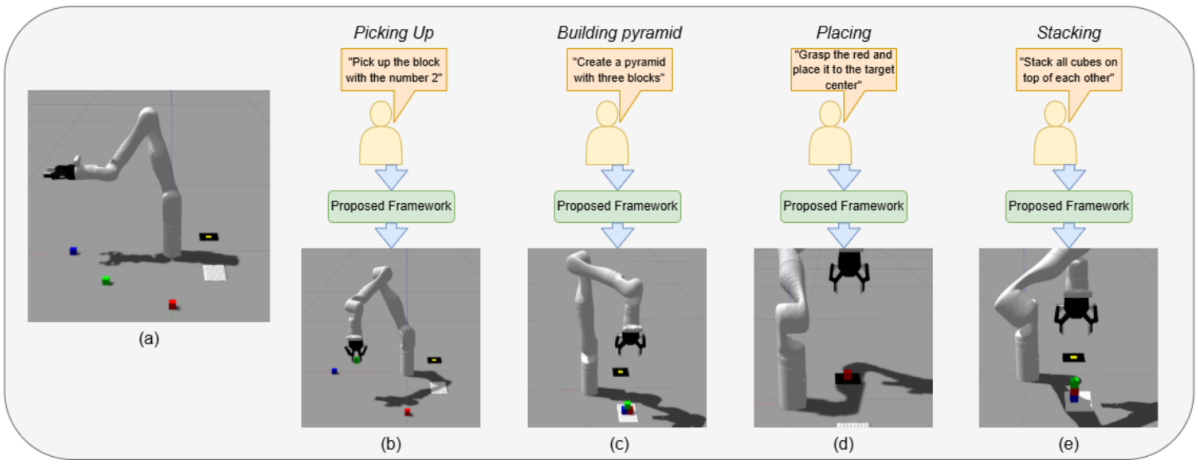

Figure 2

techniques can affect performance, we use the One-Shot and One-Shot plus Chain-of-Thought in testing. The reason why we did not directly use the Chain-of-Thought is that it can not generate a consistent format for the action sequence. This might cause the execution script to fail. Therefore, we assume the One-Shot prompting is necessary to unify the format of the action sequence.

To assess the performance of the large language models (LLMs) in generating task sequences for robotic manipulators, we have used two evalutaion metrics

Action Correctness: This metric evaluates whether the sequence of actions generated by the LLM is logically valid and aligns with the task description provided in the user's prompt. For example, for the task "Stack the red block on top of the blue block," the LLM must generate actions that move the red block to the correct position relative to the blue block without including unnecessary or invalid steps.

Format Correctness: This metric evaluates whether the output from the LLM adheres strictly to the predefined Python list format. Since the framework relies on this format for task execution, any deviations—such as additional explanations, improperly formatted actions, or missing fields—are considered format errors.

4.1 Set Up

We ran the experiment in the Gazebo simulation environment, which has a built-in physics engine for simulating the physical interaction between the manipulator and the objects. We predefined three blocks that have different colors and initial positions and a set of special locations that do not have objects on them initially. These special locations are used to help the LLMs navigate the environment for some particular tasks. The subfigure (a) in Figure 2 shows the initial states of the experiment environment, in which the blue, green, and red blocks spawned in different positions, and the manipulator is in the initial pose; the black area with a yellow dot on the center shows the location of the "target center" for placing task; the white area shows where to build the pyramid for building pyramid task.

5 Result

5.1 Action Correctness Analysis

Baseline Performance (One-Shot) :

As shown in Figure 3, the results indicate that even smaller top-performing models (7B or 9B) can be on par with a 70B model under one-shot conditions, highlighting the importance of training quality and possibly fine-tuning procedures. Smaller models like llama-3.2-1b-preview (8.25%) and llama-3.2-3b-preview (8.25%) perform really bad. The mixtral-8x7b-32768 model stands out negatively with a 0% success rate, indicating that size alone (7B) does not guarantee success. Factors like architecture, prompt style, or training data matter significantly.

Impact of Chain-of-Thought (CoT) on Actions:

Adding chain-of-thought reasoning steps generally improves action correctness: llama-3.2-3b-

preview jumps from 8.25% to 33% a substantial improvement, llama-3.3-70b improves from 66.5% to 75%. gemma-7b and gemma2-9b remain stable but still high. Some models (like gemini-1.5-flash-latest) remain unchanged and mixtral-8x7b-32768 stays at 0%, suggesting that reasoning isn't always beneficial or that these models cannot effectively leverage the CoT prompt. Overall, CoT often helps less capable models improve significantly and can even boost already strong models like the 70B model.

5.2 Format Correctness Analysis

Baseline Format Performance (One-Shot):

The plot 3 shows that certain models are already well-aligned and capable of producing correctly formatted outputs without additional reasoning steps. Smaller models seem to struggle more with formatting. This could be due to less overall capacity, fewer examples seen during training, or less robust formatting tendencies baked into their training data or instructions.

Impact of Chain-of-Thought on Format: Chain-of-thought has mixed effects on formatting. llama-3.2-3b-preview improves dramatically from 49.5% to 74.5%, indicating that reasoning can help moderately sized models structure their output correctly. llama-3.2-1b-preview actually drops from 24.75% to 16.5%, showing that reasoning can sometimes introduce complexity that confuses a smaller model's ability to produce the correct format. Models already at 100% one-shot formatting (such as gemma-7b-it, gemma2-9b-it, llama-3.3-70b-versatile, and gemini-1.5-flash-latest) remain at or near 100%, indicating no detrimental effect from adding CoT and no room for further improvement.

Overall, while CoT can help some models format better, others may find it more challenging. The benefit to formatting is less consistent than the benefit to actions.

5.3 Model Size vs Performance

When comparing model size to average correctness (the mean of one-shot and CoT performance), we see a general trend that larger models (the 70B model) and well-tuned mid-sized models (7B or 9B) perform better overall. The 11B model (llama-3.2-11b-text-preview) also shows strong performance in both action (62% average) and format (62.5% average) correctness. However, the relationship is not strictly linear. A 3B model (llama-3.2-3b-preview) performs poorly without CoT but improves dramatically with reasoning prompts. A large parameter count alone (mixtral-8x7b) does not guarantee action correctness, as shown by its consistently low scores.Quality and methodology of training matter.

5.4 Error Analysis

For models with smaller parameter sizes, errors tended toward several different types. First, there appeared to be a kind of "bleeding" that happened with the smaller models, where they failed to recognize which were the relevant objects and locations in the environment for the specific problem. It was often the case where the task was "pick up the block with the number 2" and the model provided a list that involved picking up all of the objects. Second, there were formatting errors where the model tended to add additional explanations in addition to the list despite explicitly being told to provide only the list. This was most pervasive with the Llama-3.2-1B parameter model with chain-of-thought + one-shot prompting. A potential solution to these failure cases is reducing the system prompt to only include environment information relevant to the specific task being presented. For extra explanations, it seems that smaller models refuse to obey requests to only provide the action list, so either chain-of-thought should be avoided with smaller models (when the output of the LLM is executed as live code) or introduce better parsing in the Python program to remove the extraneous text.

Although larger models were generally more accurate and consistent with formatting, there were still some tasks they struggled with. The task of stacking blocks in a pyramid caused the most errors. For example, the LLama-3.3-70B model did not generate a successful action sequence for the pyramid-stacking task. In each instance, the model came close to generating a correct sequence (it placed one block on the left and one block on the right, then placed the final block on one side instead of on the top). A potential solution for these types of errors is making it clearer to the LLM what a successful example looks like, either through more examples with few-shot prompting or further clarifying the goal beyond a one-shot example.

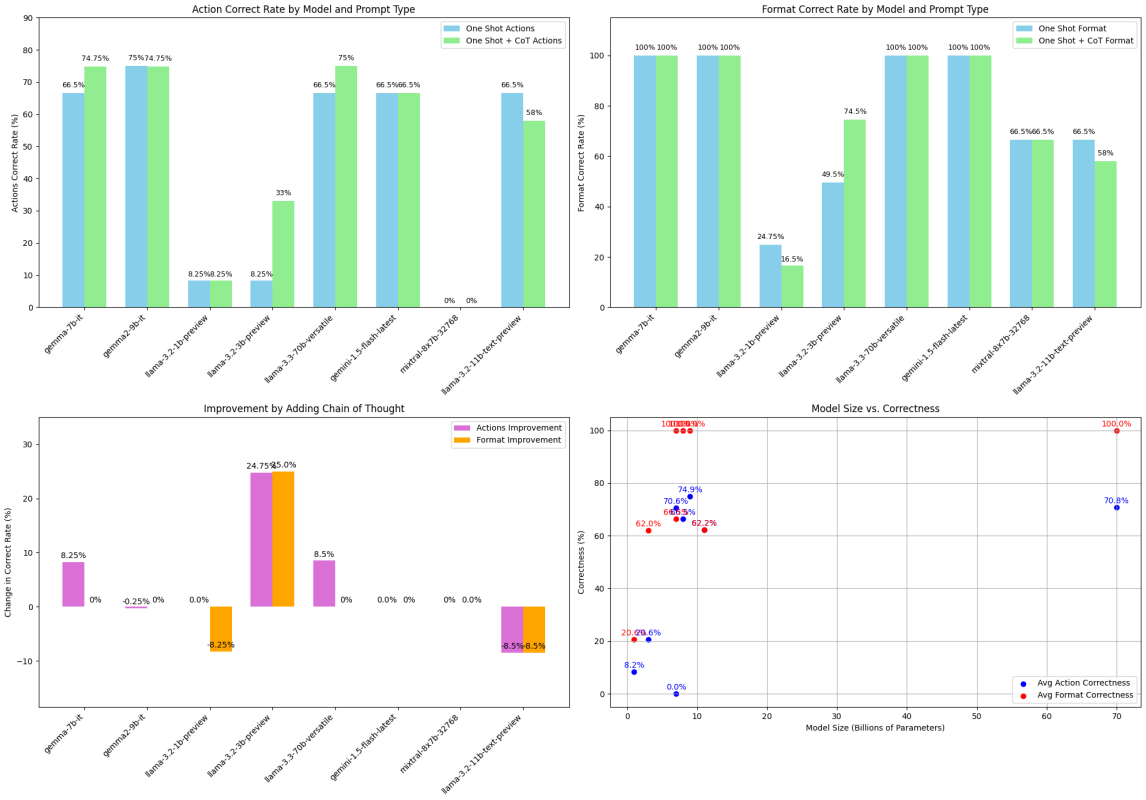

Figure 3: Qualitative Results

6 Discussion

6.1 Reproducibility of Results

Our results should be easily reproducible by others, as we tested widely available LLMs and an existing benchmark. Additionally, we added our scripts for prompting the LLMs and controlling the robot arm simulation to our GitHub repository.

6.2 Datasets

Using the RLBench benchmark to define tasks and prompt the LLM was outside of the intended purpose of RLBench, which was intially designed for reinforcement learning. Further research could explore the full capabilities of RLBench by integrating it with API calls to LLMs. Additionally, more complex tasks could be created to test the "breaking point" of larger LLMs and these tasks could be added to the RLBench repository.

6.3 Limitations and Future Work

With regard to the errors with the smaller models, it is reasonable to believe that the relatively small number of parameters was the main culprit. However, there were also limitations on the prompting strategies, the definition of user prompts, and what information was put into the system prompt. The overall prompting strategy of combining chain-of-thought with one-shot prompting appeared to cause problems for the smaller models. Possibly due to the smaller number of parameters, it may have been difficult for the models to both consider a plain-text action sequence and pseudo-code while also remembering to only output the action sequence. The decision to describe the entire environment in the system prompt likely caused additional errors, as it was common for models to include the "pyramid base" locations in the action sequences despite the task not being the pyramid-stacking task. However, these errors may have revealed that there is a possible limitation of how much of the environment can be described in the system prompt. Finally, there may have been limitations in how the tasks were described in the prompts. For example, the task "stack the blocks on top of each other" did not specify whether all of the blocks needed to be stacked or if a subset was sufficient. These limitations can be addressed in future research by systematically testing the limitations of system prompts for various model sizes and possibly augmenting the capabilities of the LLM with techniques to guide them towards "figuring out" what a successful example should look like when the task description is vague.

Finally, to give the system more control over the movement of the robot arm, future research could attempt to integrate an LLM with a Variational Autoencoder as was originally intended with this project.

6.4 Ethical Consideration

While allowing humans to control robots via speech can increase accessibility, it also raises concerns. For example, what if speech commands are misunderstood, leading to dangerous actions? We must ensure robust safety checks. Also, as LLMs can contain biases, it's important to filter instructions that might lead to harmful actions. Future work could include policy-based filters to ensure safe and ethical robot behavior.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances. *Preprint*, arxiv:2204.01691 [cs].

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. 2019. Rlbench: The robot learning benchmark & learning environment. *arXiv preprint arXiv:1909.12271*.

OpenNMT. 2024. Ctranslate2: Fast inference engine for transformer models. `https://github.com/OpenNMT/CTranslate2`. Accessed: 2024-12-11.

SYSTRAN. 2024. Faster whisper: Faster and smaller whisper transcription and translation model. `https://github.com/SYSTRAN/faster-whisper`. Accessed: 2024-12-11.

Sai H. Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. ChatGPT for robotics: Design principles and model abilities. 12:55682–55696.

Shu Wang, Muzhi Han, Ziyuan Jiao, Zeyu Zhang, Ying Nian Wu, Song-Chun Zhu, and Hangxin Liu. LLM3:large language model-based task and motion planning with motion failure reasoning. *Preprint*, arxiv:2403.11552 [cs].