Large Language Model-Based Robot Task Planning from Voice Command Transcriptions

Afonso Certo[1], Bruno Martins[2], Carlos Azevedo[1,3], and Pedro U. Lima[1]

***Abstract*— One of the primary challenges in building a General Purpose Service Robot (GPSR), i.e. a robot capable of executing generic human commands, lies in acting upon natural language instructions. These instructions often contain speech recognition errors and incomplete information, complicating the extraction of clear goals and the formulation of an efficient and effective action plan. This work presents a pipeline that leverages a Large Language Model to directly translate instruction transcripts into coherent action plans. The pipeline also integrates environmental context into the model's input, allowing for the generation of more efficient and context-aware plans. The system's performance was evaluated using a simulator based on generalized stochastic Petri Nets, achieving a success rate of around 55% on the ALFRED dataset, even in unseen environments. The entire pipeline was also successfully deployed at RoboCup 2024 in Eindhoven, where it secured second place in the GPSR task. The code, dataset, and models are available at `https://github.com/socrob/llm_gpsr`.**

I. Introduction

In the field of domestic robotics, a major goal is developing a General Purpose Service Robot (GPSR), i.e. a robot capable of executing generic human commands. One such system has the potential to transform daily routines, providing convenience and assistance, especially for individuals with limited mobility or those requiring care. Achieving this involves overcoming challenges in interpreting natural language instructions and formulating action plans.

Traditionally, a GPSR pipeline starts by transcribing spoken instructions using an Automatic Speech Recognition (ASR) system, applies natural language processing to infer goals, and uses a planner to determine actions. Each stage presents challenges, especially in dynamic environments.

While state-of-the-art ASR models, such as OpenAI's Whisper [1], can accurately transcribe user instructions, extracting goals from natural language remains challenging due to vagueness and context-dependency. For example, commands like "bring me a drink" require the robot to infer missing details (e.g., which drink to choose and its location). Traditional planners, which rely on formal representations of the environment such as PDDL, can integrate some information gathered from the environment to fill in these gaps. However, they are limited by the natural language understanding module, which often struggles to generalize from specific ways of phrasing instructions. Moreover, these formal representations cannot generalize commonsense knowledge (e.g., that both a cup and a mug can be used to serve a drink) if is not explicitly encoded.

Recent studies have demonstrated the potential of Large Language Models (LLMs) in developing end-to-end planners that convert user instructions directly into action plans [2]–[4]. These approaches typically use few-shot learning, where the language model is given a few examples of the task and then generates a plan. However, this method relies on very large models that are computationally intensive and unsuitable for robots with limited resources.

This work introduces a lightweight pipeline that leverages a LLM to generate action plans directly from instruction transcripts. The model is fine-tuned using a dataset of instruction transcripts and corresponding action plans, allowing the use of lightweight models, like Phi-3 [5], that can be deployed on resource-constrained robots. The dataset is created by combining the existing ALFRED dataset [6] with instructions generated by the command generator used in the GPSR task of the annual RoboCup@Home competition [7], and respective plans generated using an existing rule-based pipeline previously used by the SocRob@Home team. The model is then fine-tuned using this dataset, to generate action plans directly from instruction transcripts.

The proposed approach also aims to enhance the robot's decision-making by integrating knowledge about the environment when generating the plans. By querying a continuously updated semantic map of the environment, the robot can plan more efficiently. For example, if the robot has previously seen a drink in the bedroom, it can prioritize searching the bedroom before moving on to other areas.

By combining LLMs with context-aware planning, this work aims to create a more robust and efficient approach to executing commands in real-world environments.

II. Related Work

A. Petri Net-based Approaches

An example of a traditional approach for the GPSR task is the pipeline from the SocRob@Home team. In this approach, goals are extracted from user commands using a rule-based NLU module, and a planner based on a Petri Net planning toolbox [8] is used to generate the actions

This research was supported by the Portuguese Recovery and Resilience Plan through project C645008882-00000055 (i.e., the Center For Responsible AI), and also by Fundação para a Ciência e Tecnologia (FCT) through INESC-ID and LARSyS funding (DOI:10.54499/UIDB/50021/2020, DOI:10.54499/LA/P/0083/2020, DOI:10.54499/UIDP/50009/2020, and DOI:10.54499/UIDB/50009/2020).

[1]Institute for Systems and Robotics, Instituto Superior Técnico, University of Lisbon, 1049-001 Lisbon, Portugal (e-mail: afonso.certo@tecnico.ulisboa.pt; cguerraazevedo@tecnico.ulisboa.pt; pedro.lima@tecnico.ulisboa.pt).

[2]INESC-ID, Instituto Superior Tecnico, University of Lisbon, 1000-029 Lisbon, Portugal (e-mail: bruno.g.martins@tecnico.ulisboa.pt)

[3]Laboratory for Automation and Systems, Instituto Pedro Nunes, 3030-199 Coimbra, Portugal

needed to achieve these goals [9]. Although this approach provides a generalized solution for planning, that can adapt to information gathered about the environment, it is heavily limited by the rule-based NLU module, which can only handle a predefined set of commands. This limitation makes it difficult to scale the system to handle a wider range of commands and environments.

B. Planning with Large Language Models

Recent developments in the field of LLMs have opened up the possibility of creating end-to-end planners that directly convert user instructions into executable actions for robots.

One approach to leveraging LLMs in planning is to use them for evaluating the likelihood of actions fitting into a plan, rather than directly generating text. The SayCan framework [2] exemplifies this by combining language models with affordance-based scoring to improve the feasibility of generated plans. This method involves using the LLM to score potential actions based on their relevance to the user's instruction. The affordance score evaluates the likelihood of successfully executing each action in the given context. By integrating these two scores, SayCan enhances the planning process, achieving higher success rates in task completion.

Another approach to leveraging LLMs is to have the model generate a complete plan directly. This reduces the number of LLM calls compared to scoring actions iteratively. For instance, Wake et al. [3] employs ChatGPT to produce a sequence of actions based on a JSON description of the environment and a user instruction. The prompt includes available actions, their effects, and in-context examples of inputs and outputs to guide the model in generating valid plans. By updating the predicted environment state after each step, this method supports long multi-step instructions, while minimizing context retention.

A similar method by Song et al. [4] uses GPT-3 but differs in key areas. Instead of always including the environment description, this approach re-prompts the model with the currently visible objects only when an action fails. Additionally, it selects examples dynamically. The most relevant ones are chosen based on the similarity between the current instruction and a dataset of prior instructions. This selection, determined using Euclidean distance in an encoded space, ensures the prompt adapts to the task at hand. Both approaches illustrate the potential of LLMs to efficiently plan multi-step tasks in dynamic environments.

Vision-Language Models (VLMs) have also been used in robot control. This is demonstrated by the Robotics Transformer 2 (RT-2) system [10]. Instead of working with primitive actions, RT-2 integrates VLMs directly into low-level control, enabling the generation of movement instructions for the robot's joints directly from sensorial data.

Our work aims to iterate on these advancements, by fine-tuning a LLM to generate plans composed of primitive actions, allowing for the use of smaller models that can be deployed on resource-constrained robots. Additionally, the model is integrated with a semantic map of the environment, to enhance planning efficiency by incorporating the robot's knowledge of the environment.

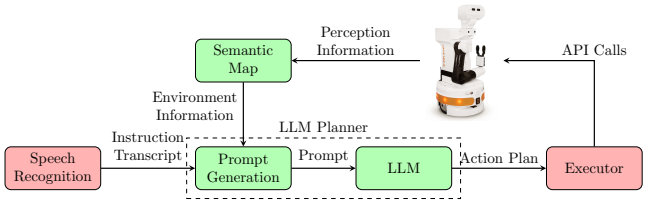


Fig. 1. Proposed pipeline.

III. Methodology

As illustrated in Figure 1, the proposed planner integrates several components to process user instructions. The workflow starts with the ASR module, which transcribes user instructions into text. This transcript, along with an environment representation, is fed into a prompt for the LLM. The LLM generates a list of actions to achieve the instruction's goals. This list is output as a JSON plan, parsed, and sent to the executor module for execution.

A. Instruction Representation

To improve resilience to external factors such as noise, the ASR module employs a diverse beam search algorithm [11] to generate multiple transcription hypotheses. This approach enables the system to capture a range of possible transcriptions, thereby enhancing robustness and accuracy.

The diverse hypotheses are integrated into the LLM prompt. Specifically, all four generated hypotheses are included in the prompt, allowing the model to consider different interpretations of the input. In Figure 2, this is represented in blue. By incorporating multiple hypotheses, the system can better handle ambiguous or noisy instructions, potentially leading to more accurate and effective responses.

B. Environment Representation

During its exploration of the environment, the robot gathers information and stores it in a semantic map. This semantic map organizes information into three levels, namely rooms, locations, and objects.

A room is the primary structure representing major divisions of the domestic environment, such as the kitchen, the bedroom, or the living room. Each room has a name and a list of associated locations.

Locations are specific places within a room, such as a kitchen table in the kitchen, a sofa in the living room, or a bedside table in the bedroom. There are two types of locations: supporting surfaces, which can contain objects (e.g., a table), and generic locations, to which the robot can move but cannot contain objects (e.g., a door).

Finally, the objects are entities the robot can manipulate, such as a wine bottle, a sponge, or a ball. Objects can have additional properties beyond their name and surface, such as a category, dimensions, and default locations.

```
<|user|>
You are Bob, a robot designed to execute general
tasks based on instructions from humans. Your
task is to choose the most appropriate spoken
instruction hypothesis from a list provided and
convert it into a list of actions that you can
execute, in a JSON format.
The following json represents the information you
know about the environment:
{"rooms": [
  {"name": "kitchen",
    "contains_surfaces": [{
      "name": "kitchen table",
      "distance": 1.977,
      "contains_objects": ["wine bottle"]
    }]
]
The instruction hypotheses are:
0. bring me a ring from the kitchen
1. bring me the ring from the kitchen
2. bring me a drink from the kitchen
3. bring me the drink from the kitchen
<|assistant|>
```

Fig. 2. Prompt example describing an environment with a room and surface, including four instruction hypotheses, three with ASR errors.

The data in the semantic map is converted into a JSON representation for inclusion in the prompt, represented in red in Figure 2. Currently, the JSON representation includes only the name and location of each object, but it can be expanded to include additional properties useful for planning. The representation also includes the straight-line distance to the locations within each room, allowing the model to choose the nearest option when faced with many possibilities.

C. Action Representations

The planner's actions are executed by the executor, which processes action messages and interfaces with the robot's API to run commands. An action is defined as a skill the robot can perform, carried out by a low-level controller, such as moving to a waypoint, picking up or placing an object, or following a person. An action can also involve active perception skills, such as finding an object or a person in the environment, or communication skills, such as describing an object or asking and answering questions.

For more complex behaviors, some actions produce outputs. For example, the "find object" action identifies a specified object and returns its unique identifier. These outputs can be stored in the executor's knowledge base and used in subsequent actions. For instance, the identifier from the "find object" action can be used as input for the "pick" action, enabling the robot to grasp the desired object.

Actions are represented as ROS messages, which include the name of the action, source and destination locations, object identifier, additional object information (e.g., object color, size, or shape), person identifier, additional person information (e.g., person's age, clothing color, or posture), voice command text, and the name of the output variable to store the action's result in the executor's knowledge base.

D. Language Model Adaptation

To generate the actions required to achieve the user's goals, the LLM must be adapted to the task. This adaptation involves two different methods: fine-tuning a Low-Rank Adaptation (LoRA) over the LLM and using a Grammar-Constrained Decoding (GCD) algorithm during inference.

LoRA [12] is a method for adapting a pre-trained LLM to a specific task, by fine-tuning a low-rank adaptation of the pre-trained model. This method allows the model to learn new tasks while retaining the knowledge acquired during pre-training, also requiring less resources in the process.

In our specific case, a dataset consisting of diverse object manipulation and human-robot interaction instructions, paired with corresponding plans, was generated using the rule-based approach from the SocRob team. These data was used to introduce the model to the planning task.

In addition to fine-tuning, to ensure that the generated plan can be successfully parsed, a GCD algorithm [13] is used during inference to enforce that the output adheres to the expected format. This algorithm uses a context-free grammar during the decoding process, restricting the model to only generate valid sequences of tokens. The grammar enforces the structure of the JSON file, validates action names and their corresponding values, and ensures that all fields are correctly populated. For example, it verifies that each action name is recognized and that destinations correspond to valid locations.

IV. Implementation Details

A. Used Datasets

Two primary datasets were employed as sources for ground truth plans: a dataset of instructions generated by the RoboCup command generator, and the ALFRED dataset.

RoboCup's GPSR Command Generator

The rule-based planning approach can generate ground-truth plans from instructions provided by the RoboCup@Home command generators. Two different versions of the generator are available.

An older generator[1] includes GPSR with 61 templates for 9 task types, and EGPSR with 86 templates for 16 task types. These tasks involve object manipulation and people interaction, such as finding, picking, placing objects, and interacting with people. The EGPSR tasks are more complex, with longer multi-step commands and more action types.

The newer generator[2] offers 37 templates for 15 task types, mainly reformulated from the older versions, maintaining task diversity aligned with competition objectives.

Both versions use randomized names for objects, people, and locations, within environments typically consisting of 4 rooms, 30 object types, and 15 to 20 person names.

To compile the dataset, 1000 commands were generated from each of the three available generators (GPSR, EGPSR, and the newer generator). Some commands from the older

[1]https://github.com/kyordhel/GPSRCmdGen
[2]https://github.com/johaq/CommandGenerator

TABLE I

NUMBER OF EXAMPLES FROM EACH SOURCE COMPILED, THE EXAMPLES IN BOLD WERE USED TO TRAIN THE MODEL.

Data Set		Number of Examples
Older Generator	GPSR	**962**
	EGPSR	**821**
Newer Generator	ISR Testbed	200
	RoboCup 2024	200
ALFRED	Train	**5077**
	Validation Seen	197
	Validation Unseen	148

version were filtered due to execution constraints, reducing the number of usable commands.

Commands from GPSR and EGPSR generators were created using the Institute for Systems and Robotics (ISR) testbed environment[3] for the training split. The newer generator produced commands for validation, divided into "ISR Testbed" (same environment as training) and "RoboCup 2024" (new environment for generalization evaluation). The distribution of examples is shown in Table I.

The ALFRED Dataset

The ALFRED dataset [6] was used to enhance the RoboCup@Home data, as it includes expert demonstrations created with comprehensive knowledge of the environment and its objects. It contains 25,743 language directives, corresponding to 8,055 expert demonstrations, covering seven task types based on object manipulation, and involving 58 unique object classes, 26 receptacle classes, and 120 indoor environments. Each environment is a single room from one of four types: kitchens, bathrooms, bedrooms, and living rooms.

The primary plans in ALFRED are defined at a lower level than our executor's actions, but the dataset also includes higher-level plans generated by a PDDL planner, which align more closely with the executor's actions and were used as ground-truth plans. Plans requiring fine motor skills, such as slicing objects, were excluded from the dataset used for fine-tuning the model.

The dataset is divided into training, validation, and test sets. However, the test set lacks the high-level plans needed for conversion, rendering it unusable. The validation set is further divided into "seen" and "unseen" environments to assess the model's generalization capabilities. The distribution of examples across these subsets is presented in Table I.

B. Dataset Expansion

To ensure the planner's resilience to ASR errors, and to evaluate its performance under such conditions, the datasets used for both training and evaluation were expanded to incorporate examples of these errors.

Experimentation revealed two primary types of errors in the ASR module: simple errors, where only one or two words in a sentence are incorrect, and catastrophic errors, which occur when the transcription is severely impacted, such as when the microphone cuts off mid-sentence. An example of a simple error is transcribing the instruction "Find John at the side table" as "Find John at the website table." In contrast, a catastrophic error might result in an incomplete or garbled transcription, making the command unusable.

Ideally, these two error types require different handling strategies. The model should, for instance, be robust to simple errors, correctly interpreting the command despite minor inaccuracies in transcription. However, in cases of catastrophic errors, where the transcript provides little to no useful information, it is crucial that the model avoids generating nonsensical plans. Instead, it should recognize the failure and prompt the user to repeat the instruction.

To simulate these scenarios, ASR errors were injected into the original instructions using a simulator [14]. This tool replaces words in a sentence with phonetically and semantically similar words, based on a probability determined by the Word Error Rate (WER). A WER of 10% was used to generate simple errors, introducing only a few incorrect words into the transcription, while a WER of 100% was employed to simulate catastrophic ASR failures.

C. Model Adaptation

The Phi-3-mini-4k-instruct model [5] was selected for its lightweight design and ability to be quantized to 4 bits, enabling it to run efficiently on a robot's laptop with an NVIDIA GeForce RTX 4070 and 8 GB of VRAM.

Model fine-tuning was carried out using the unsloth framework,[4] with the model quantized to 4 bits and LoRA parameters $r = \alpha = 16$. An 8-bit version of the AdamW optimizer was used with a learning rate of 2×10^{-4}, along with gradient checkpointing to reduce memory usage.

In addition to the examples from the training splits of the datasets, 689 simulated catastrophic ASR errors were used to train the model, totaling 7 458 examples. Training was performed with a batch size of 2 and 4 gradient accumulation steps. The fine-tuning ran on an NVIDIA A100 for up to 50 epochs, with early stopping based on the action success rate (a metric explained in the next section) every 50 steps.

V. EVALUATION METHODOLOGY

A. Evaluation Using Ground Truth Plans

To address the need for a simple evaluation metric during training, we adopted a straightforward approach: directly comparing the generated plan with the ground truth plan. Each action in the generated plan is matched to its counterpart in the ground truth plan, and the plan is deemed successful if all actions are identical. This metric is referred to as **action success rate**.

However, this metric does not provide a complete picture, since it assumes the ground truth plan is the only correct solution for each task, and does not account for valid alternative paths. For example, for the instruction "give me two newspapers," it is equally valid to retrieve one from the coffee table first and then from the shelf, or vice versa. This evaluation method can only consider one option as correct.

[3] https://isr.tecnico.ulisboa.pt/isrobonet/

[4] https://unsloth.ai

B. Evaluation Through Simulation

To address the limitations of the simple evaluation, a simulation-based approach was developed. This approach uses a Petri Net planning framework [8] to represent the environment state and simulate plan execution, checking at each step if the expected goal state has been reached.

A new dimension evaluated by this approach is the executability of a plan. Although a plan may initially seem correct, such as the plan "move(kitchen), find_person(John)" for the instruction "find John", it may not be feasible if one of its actions cannot be executed. For example, if there is no one in the kitchen, the action "find_person(John)" will fail. The execution success rate allows to evaluate this dimension, as defined next:

ExeCution success rate (EC): This metric represents the fraction of actions in a plan that can be executed in the simulator, providing insight into the feasibility of the specified actions given the simulated environment conditions.

Additionally, this approach allows for the use of two new metrics defined in Shridhar et al. [6], namely the success rate and the goal condition success rate.

Success Rate (SR): This metric is defined as the fraction of plans in a dataset that were executed successfully. A plan is considered successful if it can be executed in its entirety and achieves all the sub-goals implied by the instruction.

Goal Condition success rate (GC): This metric evaluates the partial completion of plans by representing the fraction of sub-goals needed to complete the task that were actually executed. It measures how well the plan achieves the necessary goals for task completion.

VI. Results

A. Pipeline Validation

The proposed pipeline was validated in competition environments at both the 2024 RoboCup@Home Portugal Open and RoboCup 2024 in Eindhoven. A simplified version of the planner, without environmental information, was deployed. This approach was chosen because, in competition, object locations may change between instructions, making environmental information less useful for planning.

The pipeline allowed the team to score bonus points for the first time by using a non-expert operator, i.e. someone with no robotics background who could rephrase commands. It also secured the second-best GPSR result at RoboCup.[5]

B. Generalization to New Environments

To evaluate the model's generalization across different environments, the seen subsets of the ALFRED and GPSR datasets, which use environments identical to those in the training phase, were used as baselines. This was compared against results from the unseen subsets of the ALFRED dataset and the GPSR commands from the RoboCup 2024 environment, which were not used for training.

[5]A video of the GPSR performance at RoboCup 2024 is available on the team's YouTube channel: https://www.youtube.com/watch?v=lOt5cVl01Lc

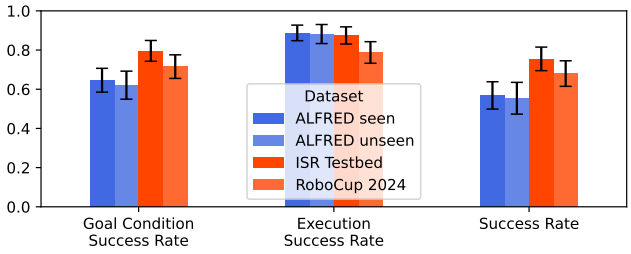


Fig. 3. Results in seen and unseen environments.

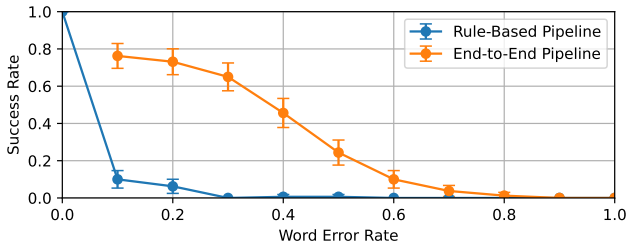


Fig. 4. Success rate for increasing levels of simulated ASR noise.

As expected, Figure 3 shows that the model performs better with datasets similar to those used in training. However, it generalizes well to new environments, with only a 1.5% drop in success rate on the ALFRED dataset and a drop of 7.5% for the RoboCup 2024 dataset.

C. Resiliency to ASR Errors

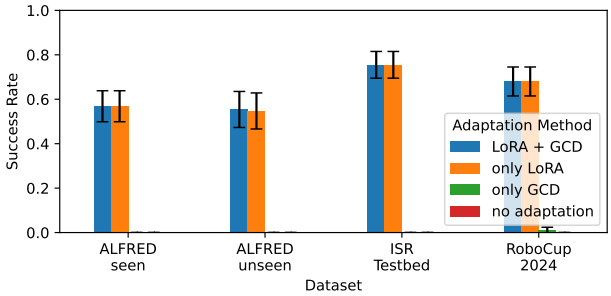
An additional experiment evaluated the planner's performance under ASR errors, by introducing varying levels of WER into the GPSR command dataset. The results presented in Figure 4 demonstrate that the new pipeline exhibits resilience to errors, maintaining a success rate of approximately 65% with a WER of 30%, while the rule-based approach, though achieving a 100% success rate in the absence of errors, quickly drops to 10% as WER increases.

Additionally, this experiment demonstrated that, as WER rises, the model generates more empty plans, recognizing when instructions become nonsensical and responding accordingly. Ideally, the model should generate a valid plan, or none at all when faced with incomprehensible commands, meaning execution success and empty plans should sum to 100%. The results show this combined metric never falls below 70%, even at higher error rates, highlighting the model's robustness in managing ASR errors.

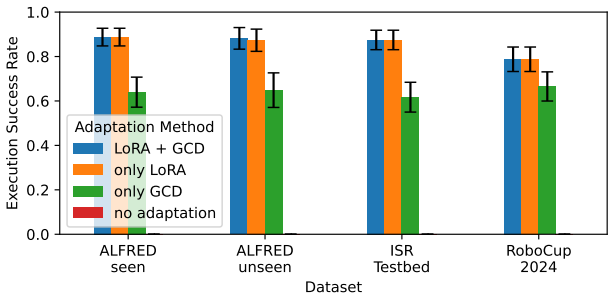
D. Ablation Studies

To evaluate the effectiveness of Grammar-Constrained Decoding (GCD) and fine-tuning with LoRA, ablation studies were conducted by evaluating the model's performance with no adaptation, each method individually, and both combined.

The results in Figure 5 show that the fine-tuned model alone generates correct and executable plans, with no significant gains when paired with GCD. Fine-tuning is crucial for



(a) Success Rate



(b) ExeCution success rate

Fig. 5. Success rate and execution success of plans generated with different model adaptation strategies. Bars for "no adaptation" are not shown, as this setup failed to generate JSON plans.

generating correct plans, as success rates drop to near zero with the original model, regardless of GCD.

However, GCD ensures plan executability. With the non-fine-tuned model, plans generated with GCD, though not fully achieving goals, were partially executable. This highlights the potential of grammars in constraining the model's output to ensure executable plans.

VII. Conclusions and Future Work

This work introduces a pipeline for robot task planning, transforming spoken instructions into executable plans. We describe a framework that integrates LLMs within the SocRob@Home GPSR pipeline, enabling the robot to interpret and act on natural language commands in dynamic environments. We specifically fine-tuned the Phi-3-mini-4k-instruct model with ALFRED and RoboCup@Home data, pairing natural language instructions with plans. Handling ASR errors with simulated error data, and using GCD during inference proved beneficial for consistent execution despite transcription inaccuracies.

Simulation tests demonstrated a success rate of around 70% for RoboCup instructions, and 55% for the more complex ALFRED instructions. This performance held, to a high degree, even in the presence of simulated ASR errors. However, improvements remain possible, especially in adaptability to real-time changes. Future work could explore re-prompting the model to adjust plans when facing unexpected conditions, enhancing task success rates through dynamic responses. Incorporating longer-term memory and conditional branching can also enhance robustness, enabling the robot to remember preferences and adapt tasks based on past interactions. This would allow robots to complete tasks more efficiently and provide more personalized interactions.

Another approach is to use the LLM to convert instructions into a generic formal language, processed by a classical planner. This hybrid method combines the flexibility of LLMs with the formal guarantees of classical planning algorithms.

References

[1] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *International Conference on Machine Learning*. PMLR, 2023, pp. 28 492–28 518.

[2] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on Robot Learning*. PMLR, 2023, pp. 287–318.

[3] N. Wake, A. Kanehira, K. Sasabuchi, J. Takamatsu, and K. Ikeuchi, "ChatGPT empowered long-step robot control in various environments: A case application," *IEEE Access*, vol. 11, pp. 95 060–95 078, 2023. [Online]. Available: https://doi.org/10.1109%2Faccess.2023.3310935

[4] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "LLM-planner: Few-shot grounded planning for embodied agents with large language models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 2998–3009.

[5] M. Abdin, S. A. Jacobs, A. A. Awan, J. Aneja, A. Awadallah, H. Awadalla, N. Bach, A. Bahree, A. Bakhtiari, H. Behl, *et al.*, "Phi-3 technical report: A highly capable language model locally on your phone," Microsoft, Tech. Rep. MSR-TR-2024-12, August 2024. [Online]. Available: https://www.microsoft.com/en-us/research/publication/phi-3-technical-report-a-highly-capable-language-model-locally-on-your-phone/

[6] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox, "Alfred: A benchmark for interpreting grounded instructions for everyday tasks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 740–10 749.

[7] J. Hart, A. Moriarty, K. Pasternak, J. Kummert, A. Hawkin, V. Hassouna, J. D. Pena Narvaez, L. Ruegemer, L. von Seelstrang, P. Van Dooren, J. J. Garcia, A. Mitzutani, Y. Jiang, T. Matsushima, and R. Polvara, "RoboCup@Home 2024: Rules and regulations," https://github.com/RoboCupAtHome/RuleBook/releases/tag/2024.1, 2024.

[8] C. Azevedo, A. Matos, P. U. Lima, and J. Avendaño, "Petri Net Toolbox for Multi-Robot Planning under Uncertainty," *Applied Sciences*, vol. 11, no. 24, p. 12087, Jan. 2021, number: 24 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2076-3417/11/24/12087

[9] P. Sarnadas, "Towards a general purpose service robot task planning system: Reasoning in a Petri net model with Monte Carlo tree search," Master's thesis, Instituto Superior Técnico, 2023.

[10] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning*. PMLR, 2023, pp. 2165–2183.

[11] A. K. Vijayakumar, M. Cogswell, R. R. Selvaraju, Q. Sun, S. Lee, D. Crandall, and D. Batra, "Diverse beam search: Decoding diverse solutions from neural sequence models," *arXiv preprint arXiv:1610.02424*, 2016.

[12] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[13] S. Geng, M. Josifoski, M. Peyrard, and R. West, "Grammar-constrained decoding for structured NLP tasks without finetuning," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 10 932–10 952.

[14] R. Voleti, J. M. Liss, and V. Berisha, "Investigating the effects of word substitution errors on sentence embeddings," in *2019 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2019, pp. 7315–7319.