Resolving Ambiguities in LLM-enabled Human-Robot Collaboration

Ulas Berk Karli
Department of Computer Science
Yale University
ulasberk.karli@yale.edu

Tesca Fitzgerald
Department of Computer Science
Yale University
tesca.fitzgerald@yale.edu

Abstract: Large Language Models demonstrate exciting reasoning capabilities that can be utilized in translating user instructions to robot actions in Human-Robot collaboration context. Yet this approach is still prone to failure due to ambiguities in user instruction or interpretation of these instructions in the process of generating actions for a robot. In this extended abstract, we summarize recent work on programming robots through natural language, identify a key research gap, and propose directions for future work with the aim of resolving ambiguities to robustly interpret and clarify natural language instructions.

Keywords: LLM, Active Learning

1 Introduction

Recent advancements in Large Language Models have opened the doors for robotics researchers to bridge the gap between a user's natural language instructions and the corresponding robot actions. Even though LLMs have been shown to be very powerful in interpreting natural language instructions and translating them into robot actions [1, 2, 3], they are not well equipped to interpret the ambiguities that are common to natural language descriptions of a task [4, 5, 6].

Suppose a user verbally requests for a robot to *"help me with the dishes"*. The user hasn't specified what kind of help they need: washing, drying, or putting away dishes. Without clarification, the robot might not know how to assist effectively. Furthermore, people are unlikely to fully specify the parameters of a task through natural language instructions due to their implicit goals for the robot's behavior or their belief about the robot's capabilities [7, 8] . As a result, using LLMs to translate natural language instructions into robot actions requires that we address several new research problems at the intersection of natural language processing, robotics, machine learning, and human-robot interaction.

In this abstract, we present the problem of *embodied ambiguity resolution* and situate it within current research on LLMs in robotics. We identify this gap in the literature, and propose directions for addressing this problem via active learning.

2 Background

Large Language Models show impressive general reasoning capabilities [10] that are naturally relevant to robotics applications. A common goal of using LLMs in robotics involves converting natural language user instructions into robot action. In this section, we group these methods into three categories, also shown in Fig.1. Within each category, we further distinguish between two main methods for generating prompts: *Template-based* methods require the LLM to provide the information missing from a pre-defined template. *Free-form* methods, in contrast, involve providing the LLM with a set of instructions and allow it to generate free-form text or code as output.

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

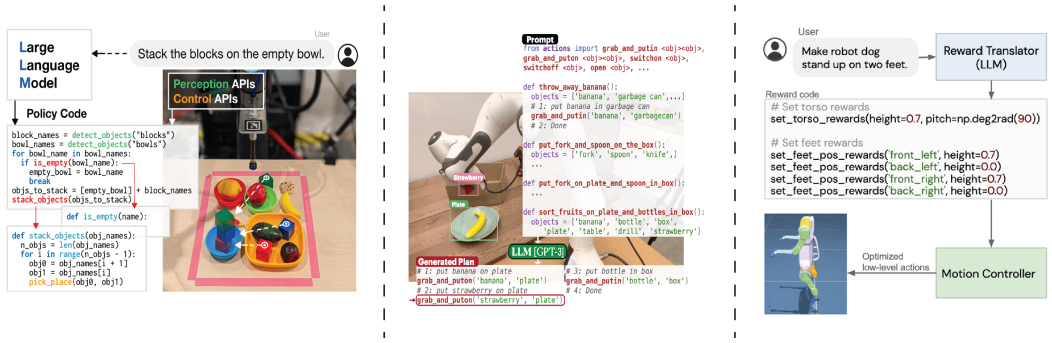

Figure 1: a) Code generation from work of Liang et al. [2]. User prompts the LLM with an instruction and LLM generates appropriate policy code that can be executed on the robot platform. b) Planning from work by Singh et al. [9]. LLM generates a plan using abstract code that is not executable but outlines the plan based on the user instruction. c) Feature extraction from work by Yu et al. [3]. LLM generates necessary reward function features from user instructions and a motion controller learns the control the robot based on this.

2.1 Robot Action Through Code Generation

LLMs are capable of directly writing code that corresponds to user instructions. This category of work harnesses that capability to convert natural language instructions from users into generated code blocks, which can be executed later on the intended robot platform. Within this category, approaches differ in how they prompt the LLM for code generation.

In *free-form generation*, the LLM is left to generate code that are not expected to fit to certain templates, they generate computer code in a natural language-like manner without rigid templates or predefined structures. They might be prompted with rules but they are not required to manipulate the input sequence to generate the output sequence.One example for this is work by Vemprala et al. [1] where the LLM is given a highly descriptive function library that is written in python as a wrapper for the underlying robot architecture. The LLM is prompted to assume an identity, such as an assistant that should produce code directly for the specified robotics platform. This prompt also specifies a function library in terms of their names, inputs, outputs and any special rules or considerations that LLM must make while using them. After this setup, the LLM is able to generate code that can fulfill user instructions.

Vemprala et al. [1] sends an initial prompt to the LLM that explains all the functions that it has access to and also specifies some rules, such as always use SI units, only use the functions defined and common python libraries.This prompt is sent at the initialization stage and gives the LLM necessary context, such as telling LLM what kind of a robot it is writing code for, what is available in the environment and what it is required of it. Later on when the user prompts the system such as *"put the apple inside the bowl"*, it generates Python code using the previously-specified API as well as any common Python libraries (e.g. numpy, math, os, etc.).

In contrast to free-form generation, *template-based generation* approaches involve turning natural language instructions from user to a template code block with hints and examples, then this is fed to the LLM as the prompt. In the work of Liang et al. [2], an LLM is given API imports and few examples as inputs and then restructures the example to generate new policy code for the robot.

One major benefit of this kind of code generation regardless of prompting method, is that the output of the LLM can be directly used for execution of the robot action with minimal overhead for some safety checks of the generated code. While these methods enable the generated output of LLM to be directly used, the execution success of the output generated by LLM is highly dependent on the APIs it is supplied. When the API requires numerical inputs or inputs that require reasoning about robot states, this approach may result in robot actions that are incorrectly parameterized. Another drawback in these methods are the inability to resolve ambiguities in instruction or generation phases.

When the user comes up with request such as *"Bring the coffee mug closer to me."* the system would generate a code that would pick the mug and move it to position relative to the user by filling the necessary API function calls but there is ambiguity here about how close or on which direction it should be closer. As a result the system will move it to a distance and direction that might not be what the user wanted.

2.2 Robot Action Through Planning

In order to avoid relying on carefully engineered APIs or having to generate API function parameters, another category of work instead utilizes the general reasoning capabilities of LLMs to translate natural language goals into a series of high-level steps.

Within the planning domain, *free-form generation* pertains to crafting a plan that doesn't conform to a predetermined template or output format. Instead, it may specify a particular output type, such as pseudo-code or a step-by-step plan, but the structure of this output is open and largely depends on the capabilities of the LLM. In the work of Ahn et al. [11], LLMs are paired with learned value functions to predict affordances for each step of the plan generated by the LLM to introduce plans grounded in the physical world. Each step of the plan in this context is a skill that is sampled by the LLM based on the probability of it being applicable to the plan. This skill then gets evaluated by the learned value function to generate the probability of executing that skill. This ensures the skills in the plan are useful and executable. Zeng et al. [12] combine Visual Language Models (VLM) with LLMs and language-conditioned robot policies to generate robot action. LLM in this work is acting as planner based on the scene representation from the VLM. Silver et al. [13] tried using LLMs for few shot planners for PDDL domains and showed that they were insufficient but can be used as a aid to a heuristic-search planner. In the work by Huang et al. [14], LLMs are utilized to generate action plans that are then executed using a closed-feedback loop based on collection of perception models. This method also offers a way for the robot agent to query the user if the prompt is ambiguous about the scene by using the scene descriptor.

Unlike free-form generation, *template-based generation* approaches in the planning domain involve supplying the LLM with a predefined template that it can populate. This approach aims to facilitate the transformation of the output into actionable steps more seamlessly.Huang et al. [15] established the use of LLMs for generating mid-level action plans, and used this plan as an input for a masked LLM to generate admissible actions from these plans. These actions are then appended to the initial prompt and again used for generating further actions via step-by-step autoregressive generation. Another work that utilizes LLMs as planners but utilizes capabilities to generate code is by Singh et al. [9], where the LLM is fed with a Python program header with imports, objects in the environment, and an implementation of a function for an example task. Even though this work tries to generate code via an LLM, the code is actually used as an abstraction for planning, an example of an output generated by an LLM for a task such as *"sort fruits on the plate and bottles in the box"* might include function calls resembling *grab_and_puton('banana', 'plate')*. This function is highly abstract and needs a concrete implementation for actual execution .Driess et al. [16] introduced PALM-E, a new integrated model trained in conjunction with an LLM on multi-modal sentences that are made up of visual, continuous state estimation and textual input. The output to any user instruction in robotics domain is again a plan for the robot to execute. Another example of this group of work is given by Mees et al. [17], where the LLM is used as a high-level planner for the affordance model that guides the robot to the vicinity of the desired area and after that point a 7-DoF language-conditioned visuomotor policy controls the robot action.

These methods offer a way to harness LLMs' reasoning capabilities without having to create highly specific and engineered APIs. In doing so, these methods create a need for a executor system that will take the plan generated by the LLM and convert it into robot actions since their output is more abstract and not directly executable. Just like code generation systems, planning systems are also prone to the problem of not being able resolve ambiguities coming from instructions or interpretations. An example instruction such as *"I have guests coming tidy up the house."* would result in a plan like tidying the house starting from the living room and going through all rooms.

This might end up being undesirable for the user because the user might had plans for having guests outside or there might have been some room that was not required tidying. There might be detailed set of expectations that the user had but the plan generated from LLM will attempt to resolve that ambiguity by just relying on the information it has which is not complete.

2.3 Robot Action Through Feature Extraction

One way to use LLMs to generate robot actions is to use them as feature extractions to be used in downstream models or paired with other methods, such as Model Predictive Control [3], to get the actions. This pairing eliminates the need to create methods that translate LLM-generated plans into robot actions, while still leveraging LLMs' reasoning capability toward identifying important features in a task.

In feature extraction approaches, *free-form generation* resembles approaches from the prior categories. In this process, the LLM identifies the essential feature from the user's natural instruction and presents it in multiple formats, including natural language, without adhering to a predetermined format. Ding et al. [18] used LLMs to extract object symbolic and geometric spatial relationships and used these in a downstream task and motion planner to generate robot actions. Chen et al. [19] use LLM as both a feature extractor and planner in a navigation scenario. In this work the first LLM is tasked with providing object proposals from users instruction and the second LLM is tasked with coming up with a plan to be executed. Another work in the domain of navigation is by Shah et al. [20], where the LLM is used extract landmarks from user instructions. Another work in LLMs as feature extractors are done by Ren et al. [21] where LLM provides a detailed description of a given feature for the given tool. This description is then used downstream to train an actor-critic model.

In contrast to free-form generation, *template-based generation* for feature extraction involves providing the LLM with a pre-established template that it can populate. The primary goal here is to streamline the conversion of the generated output into a format that downstream models can readily utilize. Lin et al. [22] used LLM to infer goal condition for planning as a set of predicates and then also utilized LLM again but this time to infer a sequence of skills, after the set is evaluated and if not feasible a greedy search is initiated by again utilizing LLM to generate top k skills and sample from that based on score. Wu et al. [23] used LLMs summarizing capabilities to generate a preference summary of the user for a room tidying task. They then used these summaries with the LLM to come up with parts of a plan. Yu et al. [3] showed another way of using LLMs to extract features for downstream training of a model. In their work there are two LLMs, first LLM generates a detailed motion description from a given user instruction and the second LLM uses this motion description to set the rewards of a motion controller such as MJPC.

Feature extraction methods aim to blend traditional models and techniques with LLMs to harness the advantages of both. They create a seamless pipeline but require careful design to utilize LLMs to their fullest without becoming excessively reliant on their outputs. It's important to note that, like other systems, they face challenges in resolving ambiguities. When the system encounter user instruction such as *"Cut the vegetables for dinner."* the LLM will give a set of features related to the task such as reward values for this task. It might give high reward for cutting all the vegetables into cubes but user might have expected thin strips or might have had different expectations for each vegetable.

2.4 Summary

We have summarized a wide variety of approaches for translating natural language commands into robot actions, categorizing them based on *what* they produce from those commands. Within each category, we have also outlined differences in how approaches prompt the LLM. Table 1 summarizes these categories and prompt methods. Irrespective of how LLMs are utilized or prompted, all existing approaches encounter difficulties when it comes to resolving ambiguities because they were not originally designed for this purpose.

	Code Generation	Planning	Feature Extraction
Template-based	[2]	[15, 9, 17]	[3, 23, 22]
Free-form	[1]	[11, 12, 13, 14]	[18, 21, 19, 20]

Table 1: We categorize existing work on LLM-enabled action generation based on how they form prompts and what they use LLMs to produce.

Despite the impressive number, variety, and quality of the work in this field, there are still unaddressed problems that are required to enable robust translation of natural language instructions into robot actions using LLMs. In the next section we identify open problems in this field.

3 Research Gap: Embodied Ambiguity Resolution

A common shortcoming among all three categories of related work is their inability to *introspect* when interpreting ambiguous natural-language instructions. Consider the example of a user requesting a robot to *"help me with the dishes"*. This instruction could reference widely-differing tasks such as washing, drying, or putting away dishes. Furthermore, the user is likely referring to a particular subset of objects with their command, rather than referencing all available dishes that the robot could interact with. Finally, the robot may not know how to complete the requested task with a particular object, such as drying a dish with an unusual geometry. This example illustrates how ambiguity may appear in user instructions at several levels: goal specification, grounding object references, and action parameterization. We refer to this challenge as *embodied ambiguity resolution*.

Ideally, a robot should attempt to identify and resolve these ambiguities before performing any actions. However, current work on LLM-enabled code generation will rely on assumptions of the user's intentions and produce actions accordingly. LLMs are designed to sample from their representation space to generate a relevant output; they do not introspect, but rather, will generate output even if it does not match the users wishes. In NLP research, this phenomenon is referred to as *hallucination*, as presented in detail by Ji et al. [24]. Recent attempts to control these hallucinations employ several methods such as ambiguity detection and asking clarification questions [5] or directly altering the user prompt with methods such as chain-of-thought prompting [25].

Resolving ambiguities is essential for robots to exhibit robust and desirable behavior according to natural language input. This requires that the system as a whole can perform introspection: evaluating its *distribution* of potential outputs (rather than just the most-probable one), detect what information it needs to disambiguate, and query the user in an effective manner. While there has been work on verifying the factual accuracy of an LLM's responses using web search [26], there has been no such work on verifying the correctness of physical robot actions resulting from LLM-generated code. We now identify three key research questions that must be addressed in order to solve this challenge.

RQ1: How can the ambiguity be detected during the response-generation phase? One of the primary challenges when implementing introspection in state-of-the-art language models like GPT-4 [27] is the absence of direct access for API users to the raw token probabilities or a confidence score. This limitation restricts users' ability to fine-tune the generation process, which can be problematic. One potential work-around is to utilize models that offer more transparency in their prompt embeddings and response-generation process (such as Llama2, CodeLlama, etc.), while maintaining comparable generation capabilities. We anticipate that as more researchers begin to address this issue, developers of such LLM models may consider providing users with increased control. Future research should explore systematic methods for detecting ambiguities in language model generations, possibly by leveraging token-level probabilities, confidence scores, or even employing other language models to assess and identify ambiguities.

RQ2: How can we categorize the cause and effect of language ambiguities? Assuming we can detect the ambiguity in the generation phase, this alone might not be enough to resolve it. The ambiguity may have been caused due to the user's wording choice in their instructions, uncertainty over the task requirements, a mismatch between objects referenced in the prompt and those actually in the robot's environment, or biases in the data used to train the LLM. Classifying the cause of ambiguity can help in forming an effective strategy to resolve it. One potential method for classifying ambiguities is to identify token-level ambiguities and map them to known challenges with LLM-produced code. For example, identifying that the LLM has low confidence in providing a numerical parameter for a function call it is writing that might be due to its lack of knowledge of numerical manipulation. An example user instruction *"Fill this cup by pouring the coffee from the carafe inside the coffee machine."* may have all the information necessary from the user, assuming LLM knows where the coffee machine is and understands where the cup is from the users *"this cup"* part of the instruction by utilizing a perception API. There might still be an ambiguity due to the task which is pouring. How to pour in terms of how to rotate the object and how high it should be is a difficult problem for LLM since this is hard to encode in text.

RQ3: How can the system effectively query the user to gather more information that resolves ambiguity? After detecting and categorizing ambiguities, a robot should be able to gather clarifying information from the user. We anticipate that Active Learning methods may be applied to this problem to formulate informative queries. However, a key challenge will involve grounding the robot's query in both language *and* action, as not all queries can be easily conveyed through language. For example, it may be more effective for a robot to gesture to an object it has a question about, rather than try to describe it through language. Generating such multi-modal queries will require a robot to optimize over several query formats in a manner similar to that posed by Fitzgerald et al. [28].

4 Conclusion

In this paper, we have summarized recent work in the new and growing field of LLM-enabled action generation for robots. We have identified an important gap in this research field: *embodied ambiguity resolution*, which entails the problem of identifying and resolving ambiguity present in the user instructions, robot environment, task constraints, and/or prior knowledge. Finally, we have proposed a set of research questions to guide future research toward solving this challenge of creating LLM-supported robots that are robust to naturally occurring ambiguities.

Acknowledgments

References

[1] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor. Chatgpt for robotics: Design principles and model abilities. Technical Report MSR-TR-2023-8, Microsoft, February 2023. URL https://www.microsoft.com/en-us/research/publication/chatgpt-for-robotics-design-principles-and-model-abilities/.

[2] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[3] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

[4] K. Keyvan and J. X. Huang. How to approach ambiguous queries in conversational search: A survey of techniques, approaches, tools, and challenges. *ACM Comput. Surv.*, 55(6), dec 2022. ISSN 0360-0300. doi:10.1145/3534965. URL https://doi.org/10.1145/3534965.

[5] L. Kuhn, Y. Gal, and S. Farquhar. Clam: Selective clarification for ambiguous questions with generative language models, 2023.

[6] M. Mishra, P. Kumar, R. Bhat, R. M. V. au2, D. Contractor, and S. Tamilselvam. Prompting with pseudo-code instructions, 2023.

[7] M. Kwon, M. F. Jung, and R. A. Knepper. Human expectations of social robots. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 463–464, 2016. doi:10.1109/HRI.2016.7451807.

[8] S. Paepcke and L. Takayama. Judging a bot by its cover: An experiment on expectation setting for personal robots. In *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 45–52, 2010. doi:10.1109/HRI.2010.5453268.

[9] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[10] J. Huang and K. C.-C. Chang. Towards reasoning in large language models: A survey, 2023.

[11] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[12] A. Zeng, M. Attarian, B. Ichter, K. Choromanski, A. Wong, S. Welker, F. Tombari, A. Purohit, M. Ryoo, V. Sindhwani, J. Lee, V. Vanhoucke, and P. Florence. Socratic models: Composing zero-shot multimodal reasoning with language, 2022.

[13] T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling. PDDL planning with pretrained large language models. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. URL https://openreview.net/forum?id=1QMMUB4zfl.

[14] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. R. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:250451569.

[15] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, 2022.

[16] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[17] O. Mees, J. Borja-Diaz, and W. Burgard. Grounding language with visual affordances over unstructured data, 2023.

[18] Y. Ding, X. Zhang, C. Paxton, and S. Zhang. Task and motion planning with large language models for object rearrangement. *arXiv preprint arXiv:2303.06247*, 2023.

[19] B. Chen, F. Xia, B. Ichter, K. Rao, K. Gopalakrishnan, M. S. Ryoo, A. Stone, and D. Kappler. Open-vocabulary queryable scene representations for real world planning, 2022.

[20] D. Shah, B. Osinski, B. Ichter, and S. Levine. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action, 2022.

[21] A. Z. Ren, B. Govil, T.-Y. Yang, K. Narasimhan, and A. Majumdar. Leveraging language for accelerated learning of tool manipulation, 2022.

[22] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*, 2023.

[23] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. A. Funkhouser. Tidybot: Personalized robot assistance with large language models. *ArXiv*, abs/2305.05658, 2023. URL https://api.semanticscholar.org/CorpusID:258564887.

[24] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, mar 2023. doi:10.1145/3571730. URL https://doi.org/10.1145%2F3571730.

[25] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[26] Z. Jiang, F. F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig. Active retrieval augmented generation, 2023.

[27] OpenAI. Gpt-4 technical report, 2023.

[28] T. Fitzgerald, P. Koppol, P. Callaghan, R. Q. J. H. Wong, R. Simmons, O. Kroemer, and H. Admoni. INQUIRE: INteractive querying for user-aware informative REasoning. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=3CQ3Vt0v99.