

Context-Aware Language Understanding in Human-Robot Dialogue with LLMs

Svetlana Stoyanchev and Youmna Farag and Simon Keizer
and Mohan Li and Rama Doddipatla

Cambridge Research Laboratory, Toshiba Europe LTD
Cambridge, UK

svetlana.stoyanchev, youmna.farag, simon.keizer,
mohan.li, rama.doddipatla @ toshiba.eu

Abstract

In this work, we explore the use of large language models (LLMs) as interpreters of user utterances within a human-robot language interface. A user interacting with a robot that operates in a physical environment should be able to issue commands that interrupt the robot’s actions, for example, corrections or refinements of the task. This study addresses the context-aware interpretation of user utterances, including those issued while the robot is actively engaged in task execution, exploring whether LLMs, without fine-tuning, can translate a user’s directive utterances into corresponding sequences of robot actions. Using an interactive multimodal interface—combining text and video—for a virtual robot operating in a simulated home environment, we collect a dataset of user utterances that guide the robot through various household tasks, simultaneously capturing manual interpretation when the automatic one fails. The collected dataset is used to compare the interpretive performance of the proprietary gpt-5-mini model with an open-source Qwen3 model. Overall, GPT and Qwen achieve exact-match accuracy of 67% and 59% with only five dynamically selected few-shot examples, which is above the accuracy achieved with forty randomly selected examples. Our findings reveal that utterances issued during plan execution pose a challenge for both models.

1 Introduction

We envision that robotic assistants will be working side-by-side with people helping us to perform physical tasks at homes, workplaces, warehouses, and hospitals. Language is an ideal natural interface for people to interact with their assistants, human or robotic. Through task-specific instructions, real-time feedback, and iterative correction, users could guide robots in mastering the tasks and enriching their understanding of the domain and its requirements. While robots would need to under-

stand all types of utterances, including questions or general comments, understanding task-specific user ‘directives’ – utterances that instruct a robot to perform physical actions – is a basic requirement for the human-robot interface and the focus of this work. Understanding of a directive by a robot can be viewed as the translation of an utterance in the given context into a sequence of robot actions. The set of possible actions is based on the capabilities of the robot. Our virtual household robot can pick-up and place objects, open cupboards, toggle a switch, and navigate to objects. Directives vary in their specificity ranging from high-level generic, e.g. ‘make coffee’, to specific, e.g. ‘pick up the tomato’, and a robot should be able to handle both high-level and specific directives.

A highly experienced assistant familiar with the tasks and the environment may only need high-level instructions. However, when training an assistant to perform a new task, a user may need to issue specific directives, e.g. ‘go to the sink and empty the cup’ or a correction in the context of ongoing actions ‘not this one, use a different mug’. For high-level directives, research addressing planning for embodied agents shows that LLMs, not without occasional hallucinations, can produce plans and even executable code (Ahn et al., 2022; Huang et al., 2022a; Wang et al., 2023; Dalal et al., 2024; Shridhar et al., 2020). Unlike a high-level generic directive, a specific directive may not require complex decomposition making it easier to interpret on its own. However, a specific directive may be given when the user observes the robot’s actions and notices a mistake. Such directives may reference previous utterances and objects in the current scene. The contextual dependence of a specific directive poses a challenge for its interpretation. All directives are decomposed into sequences of actions based on knowledge of the task and understanding of the limitations of the physical environment, and we envision that a single model would

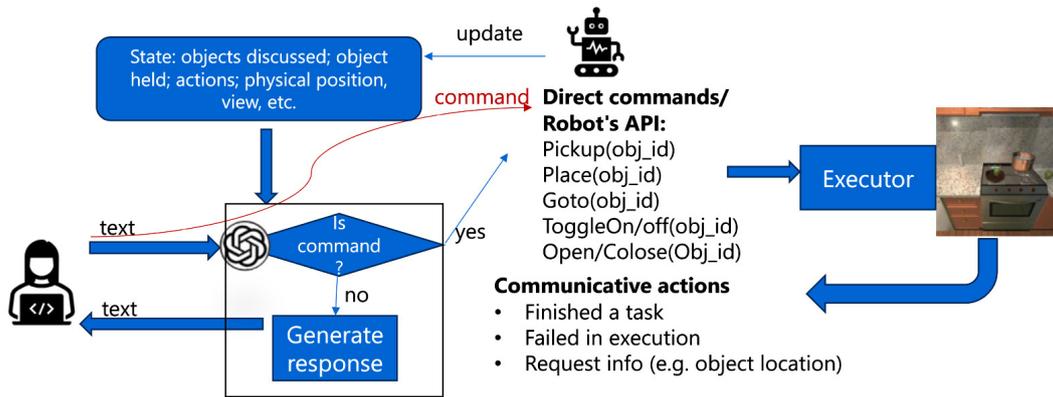


Figure 1: System diagram.

perform interpretation for both generic and specific directives alike.

Previous studies use LLMs as planners in synchronous interactions where a user provides a directive, the robot comes up with a plan of actions, executes it without interruptions, and responds to the user (Shridhar et al., 2020; Padmakumar et al., 2022b; Sarch et al., 2023; Zhang et al., 2022; Jiang et al., 2025). These paradigms assume a linear interaction flow that may not reflect real-world dynamics. In reality, users would be engaging with robots in a more fluid and asynchronous manner by issuing new directives while the robot is still executing ongoing actions.

We implement a multimodal interface – combining text and video – for a virtual robot performing tasks in a home environment and use it to collect human-robot interactions annotating them using an *expert-in-the-loop* method (see Section 2.3). As far as we are aware, this is the first human-robot interaction dataset that includes context-dependent utterances issued during plan execution.

While in the past language understanding for dialogue systems required collecting domain-specific training data for custom models, today pretrained LLMs can eliminate the need for such training data. We use the collected dataset to analyse the strengths and limitations of using an LLM as an interpreter of utterances in human-robot dialogue, addressing both generic and context-dependent utterances.

The contributions of this work are:

- A multimodal interface (text and video) for controlling an in-house robot in a virtual environment and collecting data using an *expert-in-the-loop* method (Section 2).
- A dataset of human-robot interactions, with 484 utterances, including plan-interrupting and failure-correcting utterances annotated by

experts with ground-truth interpretation (Section 3).¹

- An evaluation of two LLMs on the collected dataset and an analysis of results across contextual categories (Section 4).

2 Method and System

2.1 Examples Repository

In this work, we experiment with using LLMs to generate a sequence of API calls that would execute a user’s natural language command (or directive) to a robot, a type of code generation task that LLMs are capable of performing. We use a pretrained LLM with few-shot learning to interpret user commands directing a robot in completing domestic tasks.

While previous work explored the use of LLMs for planning in physical environments (Ahn et al., 2022; Huang et al., 2022b; Song et al., 2023; Sarch et al., 2023; Singh et al., 2023), in this work we address user command interpretation in context of an ongoing task execution. High-level directives such as ‘*make coffee*’ require planning, while corrections such as ‘*use another cup*’ require adjusting the current plan. For both types of directives, the model outputs a sequence of API calls. We evaluate if providing LLMs with relevant few-shot examples can help the model to generate execution plans in response to both high-level and plan-interrupting directives. In the rest of the paper we refer to the sequence of API calls generated by the LLM as ‘*plan*’.

Central to our approach is the *examples repository* – a database storing annotated examples from previous interactions. Each example consists of the state of the dialogue and the environment (includ-

¹<https://github.com/sstoyanchev/Human-directing-robot-dialogue.git>

ing discussed or visible objects and the previous plan), the user utterance, and the annotated ground-truth system response, represented as a sequence of API calls (see Figure 2-b). At test time, the few-shot learning examples are dynamically extracted from the repository and appended to the prompt. Dynamic selection of relevant few-shot examples has been shown effective for many tasks, including embodied planning (Pecher et al., 2024; Adiga et al., 2024; Sarch et al., 2023; Song et al., 2023). In this work, we evaluate this method for interpretation of user utterances in human-robot interaction where the context includes visual information, discussed objects, and the previous plan, which may have been interrupted by the user.

We collect and annotate, using an *expert-in-the-loop* approach (see Section 2.3) and manual correction, a dataset of 484 user utterances. Because of the small size of the dataset, we use the leave-one-out method to evaluate the LLMs’ interpretation accuracy: for each test instance in the dataset, the set of all remaining instances serves as the pool from which few-shot examples are selected.

We explore two methods of example selection: random and similarity-based. To extract few-shot examples using a similarity metric for the example i , we select the $K \in (3, 5, 10)$ most similar examples. The semantic similarity $S(i, j)$ between a pair of examples i and j is stored in an $N \times N$ matrix S (N = the number of examples in the dataset) computed using a linear combination:

$$S = \alpha \cdot \mathbf{O} + \beta \cdot \mathbf{P} + \gamma \cdot \mathbf{U} + \delta \cdot \mathbf{C} \quad (1)$$

where $\mathbf{O}, \mathbf{P}, \mathbf{U}, \mathbf{C}$ are $N \times N$ matrices of semantic similarity for object descriptions, previous plans, utterances, and context-label strings (e.g. ‘Empty execution stack & successful last command execution’ or ‘Non-empty execution stack & failed last command execution’).² While the parameters of Equation 1 can be learned to optimize performance, in these experiments we heuristically set $\alpha = \beta = \gamma = 1$ equally weighting objects, plans, and utterances.³ We set $\delta = 100$ to prioritize selection of examples with the matching context type (e.g., interruption, correction, or new command). This ensures that when a user utterance interrupts a plan execution, the few-shot examples selected to interpret this utterance are also interruptions.⁴

²Similarity matrices were computed between strings using the Sentence Transformers model all-MiniLM-L6-v2.

³Parameter optimization remains future work.

⁴Except in cases where where K exceeds the number of

The dataset includes all types of directives that a user may issue to a robot, including high-level directives (e.g., ‘make coffee’) and more specific contextual directives (e.g., ‘use another cup’). We hypothesize that, especially for smaller models, selecting relevant few-shot examples is key to improving interpretation accuracy. Next, we describe the system used to collect the data.

2.2 System Overview

Figure 1 outlines the functionality of our multi-modal system, which includes a text input/output and a video output interface integrated with the AI2THOR simulator (Kolve et al., 2022), further extended by the TEACH project (Padmakumar et al., 2022a). The simulator is instantiated with photorealistic scenes of home spaces designed in previous projects (Blukis et al., 2022). A virtual robot moves within this environment interacting with objects.

Through its text-based user interface, the system accepts both natural language and direct calls to the executor’s API. Natural language input is processed by an LLM, which first recognizes the type of utterance (*directive* vs. other). If the utterance is a directive, it is interpreted by the LLM as a sequence of API calls. For utterances other than directives – such as question, social or non-task-related input – the LLM is instructed to directly generate a response to the user.⁵ Alternatively, a user can directly enter a sequence of API calls (e.g., ‘Goto(Table_0);Pickup(Potato_1)’), which are detected using a regular expression and are **not** interpreted by the LLM.

The sequence of API calls, either interpreted from an NL utterance or directly entered by a user, is passed to the *Executor*, which invokes them one by one in the virtual environment. When a command is executed successfully, the state of the virtual environment is updated. If command execution fails – either due to a logical error (e.g., an object is too far to interact with) or a limitation of the simulator⁶ – the execution stops and the system notifies the user of the failure, requesting help to recover.

The prompt to the LLM that interprets a user utterance includes a general instruction with a description of the executor API (see Appendix A),

examples with the matching context in the repository.

⁵Evaluating responses to non-task-related natural language input remains future work.

⁶We encounter cases where placing objects fails even when it should be possible.

few-shot examples (see Appendix B), the last user utterance, and the current state.⁷ As directives are interpreted in the context of previous actions, the state (both historical in the examples and current) includes a list of discussed, held, and visible objects, as well as the previous plan with the execution status (see Figure 2-b).

In addition to *direct API calls* and *natural language directives* used to communicate with the robot, a user may type a meta-command to retrieve state information or reset the system state. These meta-commands are designed to help expert users to effectively interact with the system. The input types accepted by the system are summarized in Table 1.

User Input	Example/Description
Direct API call	Goto(Table_1);Pickup(Potato_1) pass to the executor
NL command (directive)	'get a potato from the table' convert to <i>direct command</i> using LLM
Question/social	<i>Hello there!</i> response is generated by LLM
Meta- commands	
objects	show the list of discussed/held objects
plan	show the current plan
resetobjs	reset the set of objects
resetplan	reset current plan
resetall	reload the environment and reset plan and objects

Table 1: Summary of the user input types.

2.3 Expert-in-the-loop Mode

During data collection, the system operates in *expert-in-the-loop* mode, where after each natural language *directive* utterance, the system displays the sequence of API calls interpreted by the LLM and prompts the user to confirm or correct it. If the user confirms (by pressing 'Enter' or typing 'y'), the API call sequence generated by the LLM is sent to the executor. If, in response to the system's confirmation, the user types an API call sequence, it is treated as the correct interpretation of the previous NL utterance. It is sent to the executor and a triple <State, User Utt, API Sequence> is saved into the examples repository.

To communicate naturally with a robot, a user

⁷The actual model type is a configurable parameter.

should be able to issue high-level commands as well as specific commands that interrupt the robot's actions with a correction or a new directive. We simulate interruptions by pausing after the execution of every action and prompting the user to decide whether to continue execution or type an interrupting directive.

Figure 2-a illustrates a script of an expert user interacting with the system in *expert-in-the-loop* mode. The initial NL command (line 1) is correctly interpreted by the LLM. On line 15, the user issues another command, '*actually take it to the pot*', which interrupts the current plan execution. Given the context of the interrupted plan (line 12) and the discussed objects (line 14), the LLM incorrectly interprets the last utterances, producing an incorrect plan (line 17), which is corrected by the user on line 18. After this interaction, a new entry is saved to the examples repository (Figure 2-b).

For this data collection, we used GPT4.1 as the language interpreter with 70 static examples collected previously through using the same interface.

3 Data and Experiments

3.1 Data set

Executor stack	Execution status	Total	Manual Correct
Empty	Success	246	20
	Failed	37	5
	Null	74	9
Not empty (has actions)	Success	39	39
	Fail	38	38
	Null	50	8
Overall		484	119

Table 2: Statistics on the collected dataset across six context categories, based on the executor stack and execution status of last command (see Appendix B for examples).

Prior to data collection, four users (co-authors of the paper) were trained to control the virtual robot by completing ten tasks using API calls that were passed directly to the executor. We refer to them as 'experts'. Next, the experts interacted with the system using the *expert-in-the-loop* method described in Section 2.3. Their goal was to instruct the virtual robot with natural language commands to perform household tasks such as making coffee, collecting objects, and cleaning dishes by providing step-by-step instructions or high-level directives.

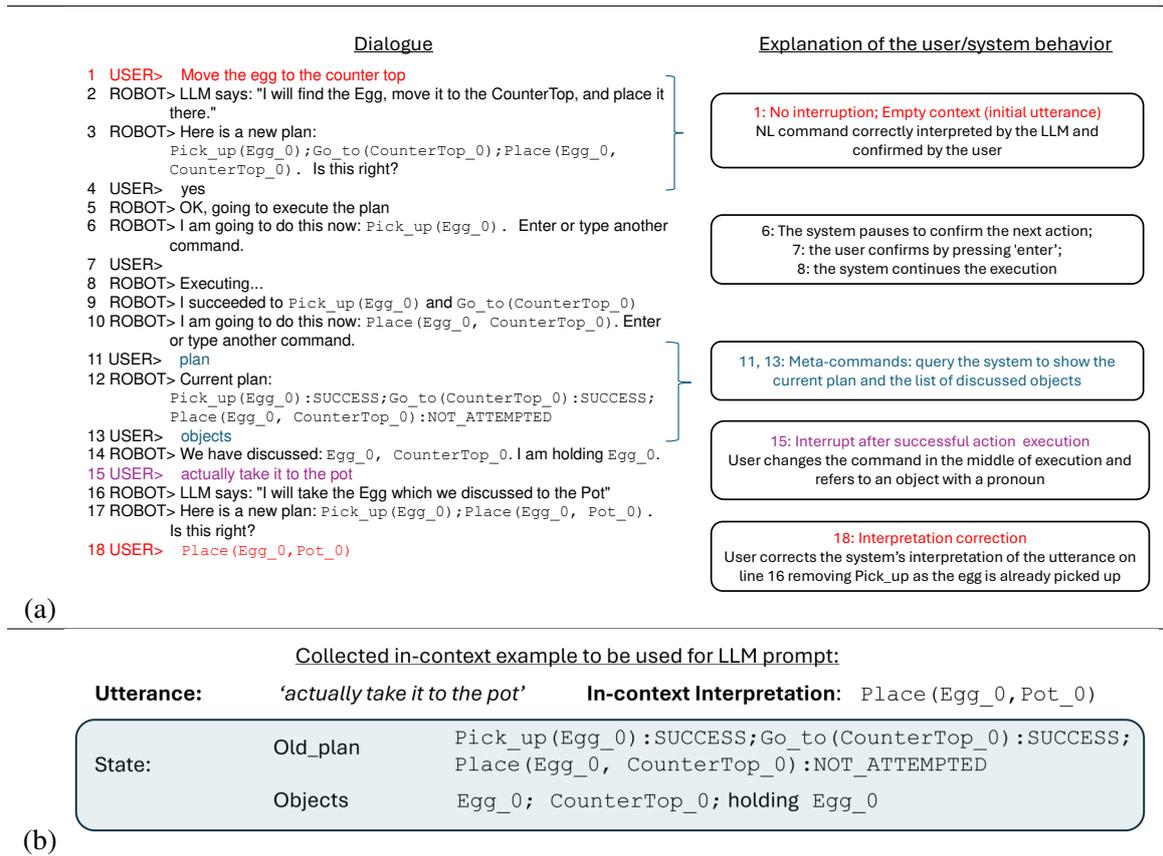


Figure 2: (a) Expert-system interaction with explanations of user and system behaviour. (b) In-context example annotated by the expert resulting from this interaction and added to the examples repository.

The experts were explicitly encouraged to issue interrupting directives while the system was performing actions. In addition, they provided manual corrections by typing the correct API calls when the automatic interpretation was wrong.

We recorded the NL utterances, their context, and their corresponding automatic (GPT-4.1) and manually corrected interpretation (when provided by the expert), resulting in 484 examples summarized in Table 2. To better understand the capabilities of LLMs, we distinguish six contextual categories in which a user utterance may be issued. These categories are defined based on two factors: (a) whether the robot’s execution stack is empty, and (b) the execution status of the last directive—Success, Fail, or Null.⁸ The execution stack is empty at the start of the interaction (Null execution status) or when the previous directive has completed executing (with Success or Fail status). In both cases, the robot is idle until it receives a new directive. Conversely, the stack is non-empty if the

⁸User directives are processed after each unit action of the simulator (e.g., pick-up, place). We do not handle interruption of a unit action, which in a physical environment may still be incomplete at the time the directive is issued.

robot is in the middle of executing a plan—either before the first action has begun (Null status) or after completing a non-final action (Success or Fail status). When a directive is issued while the robot’s execution stack is not empty, it is likely intended to modify or augment the ongoing execution plan. Such directives require contextual interpretation that accounts for the current state of execution, which we hypothesize to be more cognitively demanding for both the user and the system.

In our dataset, as expected, the majority of user utterances were issued when the execution stack was empty, following a successful execution.

The dataset contains 39 instances of interrupting commands issued with a non-empty execution stack that follow a successful execution and 38 that follow a failed execution. For example, while making coffee, after the robot picks up a cup, a user observes that it is dirty and instructs the robot to wash the cup first. The correct interpretation for such an instruction should result in a new plan to wash the cup and then continues making coffee.

During their interaction with the system, experts corrected 20% of the automatically generated plans. After collecting the data, we further man-

ually checked and corrected all instances where an expert had provided a correction, as well as all instances with non-empty stack after a failed or successful execution.⁹

To simplify annotation without losing information, we introduced symbolic shortcut annotations: CONTINUE_NA and CONTINUE_FAIL, which indicate that the *old plan* should be restarted from the point of interruption, beginning with the first not-attempted or failed action. Overall, we manually checked a quarter of the instances in the dataset. For each checked utterance, the annotator also added an explanation - a natural language description which explains the users' instruction in further detail.

After manually checking the expert annotations, we note that the *expert-in-the-loop* annotation method can be improved by providing experts with a more user-friendly annotation interface and additional training.

3.2 Evaluation Metrics

To evaluate LLM interpretation on the collected dataset, we use exact match by comparing the generated plan with the annotated ground truth sequence of actions (GT). An NL command is interpreted as a sequence of 'grounded' actions (applied to specific object instances in the virtual space). A model's interpretation is considered correct if both the predicted actions and the object parameters match the GT. For example, `Pickup(potato_1)` is not the same as `Pickup(potato_2)` because it refers to a different instance of the object. We opt for a strict exact match instead of commonly used LLM-based metrics because the nuances of object distinction may not be reliably detected by the LLM. While our strict metric may downplay the model performance, it is still useful for comparison across models and contextual categories (see Appendix C for preprocessing details).

We report accuracy as the proportion of correctly interpreted examples in the entire dataset, analyzing it across the contextual categories.

3.3 Models

User utterance interpretation is performed on every turn of the dialogue. As it is desirable to balance the trade-off between cost and performance, we aim for a smaller model to handle interpretation rather

than consulting OpenAI for each user turn. We are nevertheless interested in how the performance of a 'reasoning' OpenAI model compares to that of smaller public models which could run on the robot. In our experiments, we evaluate gpt-5-mini and Qwen3-4B-Instruct-2507 (Yang et al., 2025). We chose gpt-5-mini from the suite of GPT options for its lower cost and reported reasoning ability. We chose Qwen3 with 4B parameters tuned to follow instructions as a smaller alternative that we still expect to perform the interpretation task.

For the gpt-5-mini model, we use the *medium* setting for reasoning. For the Qwen3 model, we use the *temperature* = 0 and *tp* = 0.95.

4 Evaluation Results

To determine whether selecting similar in-context learning examples improves the accuracy of directive command interpretation, we compare two selection methods: (1) random from the matching contextual category (CAT+RAND) and (2) similarity-based selection computed using Equation 1 (SIM). To prime the model more effectively, the examples are presented in reverse similarity order, with the most similar example appearing last in the prompt.

Table 3 shows that, across the board, increasing the number of examples leads to higher accuracy and that the SIM method is more effective than CAT+RAND. We observe that for both GPT and Qwen models, selecting 5 few-shot examples based on similarity achieves higher accuracy (0.672/0.592) than randomly selecting 40 examples (0.657/0.583).

Overall, both models achieve their best accuracy in the SIM condition with 10 examples with Qwen slightly below GPT (0.613 vs 0.672).

Answer in the Prompt. To better understand model behavior, we ask the question "How would the models behave if provided with the test example itself in the prompt?". The bottom part of Table 3 shows the difference between GPT and Qwen in this 'cheating' setting. Despite being provided with the test example in the prompt, both models fail to reach a 100% accuracy. The GPT model even scores below 90%, probably due to its focus on reasoning.

Accuracy Across Categories Table 4 shows the breakdown of accuracy per utterance category for GPT and Qwen using similarity-based example se-

⁹Some of the manual corrections were due to typos. In some cases experts failed to identify LLM interpretation errors in the non-empty stack cases.

Model	Num examples	Acc SIM	Acc CAT+RAND
GPT	3	0.634	-
	5	0.672	0.543
	10	0.672	0.592
	20	-	0.638
	30	-	0.621
	40	-	0.657
	50	-	0.686
Qwen	3	0.535	-
	5	0.592	0.413
	10	0.613	0.476
	20	-	0.533
	30	-	0.564
	40	-	0.583
	50	-	0.596
Answer in the prompt (cheating)			
GPT	10	0.882	-
Qwen	10	0.943	-

Table 3: Comparison of example selection methods: random from the same category (CAT+RAND) or using similarity (SIM).

lection. The last two columns correspond to directives issued with a non-empty stack while the robot is executing an action sequence, where a user interrupts an ongoing successful execution or attempts to recover from a failed execution. As both models achieve low accuracy on these directives (≤ 0.5), we conclude that even with GPT’s reasoning capabilities and the presence of similar examples, adjusting the plan after an interruption remains challenging for the models.

We observe that Qwen has much lower accuracy than GPT on examples following an execution failure, as well as on interruptions with a Null context, where an expert corrects the interpretation of the initial command. This indicates that GPT’s reasoning capability may be useful for interpreting error-recovery directives.

Qwen degrades more than GPT as the number of few-shot examples in the prompt decreases. For non-interruptions with a Null context (initial utterances in dialogue), when the number of examples is reduced from 10 to 3, Qwen’s accuracy drops by 20 percentage points, from 0.616 to 0.411, while GPT drops only by 7 points, from 0.575 to 0.507. These utterances are likely high-level commands requiring the model to plan. As GPT is a larger model with an innate capability to produce plans for high-level commands, it is able to do so with

fewer examples.

Effect of Shortcuts During annotation of interrupting utterances, we introduced the ‘CONTINUE’ shortcut (see Section 3.1). Instead of annotating the full sequence of actions, the shortcut indicates that the plan should continue from the point of interruption or failure. If the model predicts a CONTINUE label, it is expanded during execution to the full action sequence, extracted from the symbolic state. The few-shot examples in both conditions are identical except that the interpretation without the shortcut consists of full sequences of API calls. Table 5 shows that without shortcuts, both models struggle to generate a plan for interrupting utterances, demonstrating the effectiveness of simplifying the LLM’s task and combining it with symbolic processing.¹⁰

5 Related Work

Building embodied communicative agents capable of following user instructions while assisting in physical spaces has gained significant attention in AI research. These agents follow user instructions navigating in virtual spaces (Anderson et al., 2017), performing tasks (Blukis et al., 2022), and answering questions about the environment (Das et al., 2018; Gao et al., 2022). Agent-initiated interaction with the user has been shown to be effective for navigation (Shen et al., 2025; Chi et al., 2020; Thomason et al., 2020; Anderson et al., 2017) and object-search tasks (Nguyen et al., 2019). In this work, we address interpretation of utterances initiated by the user that may be interrupting the agent in its task execution.

As LLMs encode rich semantic knowledge about the world, they have been used to solve long-horizon manipulation tasks for robots (Ahn et al., 2022; Lin et al., 2023). Our goal aligns closely with code generation using LLMs, as we aim to produce sequences of API calls to instantiate pretrained robot functions (Huang et al., 2022a; Wang et al., 2023; Dalal et al., 2024; Shridhar et al., 2020).

A small number of relevant examples can have a large impact on the model performance leading to studies on the effect of dynamic example selection methods (Pecher et al., 2024; Adiga et al., 2024). As dynamic example selection has been shown effective for embodied planning (Sarch et al., 2023;

¹⁰Since shortcuts were only used for interruptions after successful or failed execution, we report comparison only for these categories.

Execution Stack	Empty <i>Action sequence finished executing</i>			Not empty <i>User interrupts or execution fail</i>			
	Last execution (Utt type)	Null (New cmd)	Success (New cmd)	Fail (final) (New/Corr.)	Null (Corr.)	Success (Interrupt)	Fail (Recover)
MODEL # ex							
GPT 10	0.575	0.742	0.784	0.680	0.500	0.424	
Qwen 10	0.616	0.713	0.595	0.420	0.500	0.303	
GPT 5	0.575	0.754	0.703	0.640	0.553	0.424	
Qwen 5	0.548	0.672	0.622	0.420	0.579	0.333	
GPT 3	0.507	0.721	0.649	0.640	0.447	0.455	
Qwen 3	0.411	0.643	0.595	0.380	0.474	0.242	

Table 4: Performance of GPT and Qwen across categories using the similarity-based (SIM) example selection method with 3, 5, and 10 examples. The utterance type (new command, correction, or interruption) is inferred from the context type.

MODEL	Shortcuts	After Success	After Fail
GPT	YES	0.500	0.424
	NO	0.237	0.333
Qwen	YES	0.500	0.303
	NO	0.211	0.121

Table 5: Accuracy with and without CONTINUE shortcuts in the annotations for interrupting utterances. The prompt includes 10 few-shot SIM examples.

Song et al., 2023), we employ it in our system to improve the performance of contextual interpretation of utterances issued during plan execution.

Padmakumar et al. (2022a) introduce a dataset of task completion human-human interactions in a virtual environment where the *commander* gives instructions and the *follower* plays the role of a robot. The majority of commander utterances in this dataset are issued while the follower is executing a task, which serves as motivation for our work. While human-human data may not be directly suitable for automatic agents (Min et al., 2022), we collect a dataset of human-system interactions which, to our knowledge, is the first such dataset containing user utterances that interrupt robot actions.

Recent advances in vision-language-action models map language directly to robot joint manipulation (Kim et al., 2025; Gemini Robotics Team et al., 2025). When combined with reasoning, these models are likely to supersede neuro-symbolic approaches.

6 Summary and Future Work

We build an interactive multimodal interface for controlling a robot in a virtual environment and use it to collect a human-robot interaction dataset. The

dataset includes user utterances that interrupt ongoing plan execution and provide corrections during execution failures. A pretrained LLM, provided with few-shot examples interprets users’ natural language directives into executable robot manipulation calls.

We explore a novel *expert-in-the-loop* methodology to collect and annotate an interactive dataset, where an expert user directs a robot in a physical task while simultaneously providing annotations. Using the collected dataset, we compare the performance of a reasoning-enabled GPT model and a non-reasoning open-source model.¹¹

We confirm that using similarity-based method to select few-shot examples, along with symbolic shortcuts, improves interpretation accuracy. Our experimental results indicate that action-interrupting and failure-recovery commands are more challenging for the models to interpret. While both models achieve similar accuracy on the user directives following successful execution, interpreting utterances after an execution failure benefits from ‘reasoning’, suggesting the need for an approach adaptive to the situation.¹²

Next, we plan to evaluate the proposed interactive robot interface by analysing objective task success and subjective user experience. Our next avenue of research will focus on multimodal and continuous state representation with the use of Vision and Speech Language Models to natively process scenes and spoken commands.

¹¹We evaluate gpt5-mini and Qwen3-4B, deliberately avoiding very large models, which are impractical for an interactive system that must process utterances at every turn.

¹²GPT-5 already does this by choosing when to ‘reason’.

References

- Rishabh Adiga, Lakshminarayanan Subramanian, and Varun Chandrasekaran. 2024. [Designing informative metrics for few-shot example selection](#). In *Annual Meeting of the Association for Computational Linguistics*.
- Michael Ahn et al. 2022. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*.
- Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel. 2017. [Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments](#). *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3674–3683.
- Valts Blukis, Chris Paxton, Dieter Fox, Animesh Garg, and Yoav Artzi. 2022. [A persistent spatial semantic representation for high-level natural language instruction execution](#). In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 706–717. PMLR.
- Ta-Chung Chi, Minmin Shen, Mihail Eric, Seokhwan Kim, and Dilek Hakkani-tur. 2020. [Just ask: An interactive learning framework for vision and language navigation](#). *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2459–2466.
- Murtaza Dalal, Tarun Chiruvolu, Devendra Chaplot, and Ruslan Salakhutdinov. 2024. [Plan-seq-learn: Language model guided rl for solving long horizon robotics tasks](#). In *The Twelfth International Conference on Learning Representations ICLR 2024*.
- Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. 2018. Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Xiaofeng Gao, Qiaozi Gao, Ran Gong, Kaixiang Lin, Govind Thattai, and Gaurav S. Sukhatme. 2022. [Dialfred: Dialogue-enabled agents for embodied instruction following](#). *IEEE Robotics and Automation Letters*, 7(4):10049–10056.
- Gemini Robotics Team et al. 2025. [Gemini robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer](#).
- Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.
- Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022b. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.
- Chenxi Jiang, Chuhan Zhou, and Jianfei Yang. 2025. [REI-bench: Can embodied agents understand vague human instructions in task planning?](#) *Preprint*, arXiv:2505.10872.
- Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. 2025. [OpenVLA: An Open-Source Vision-Language-Action Model](#). In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 2679–2713. PMLR.
- Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. 2022. [Ai2-thor: An interactive 3d environment for visual ai](#). *Preprint*, arXiv:1712.05474.
- Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. 2023. [Text2motion: From natural language instructions to feasible plans](#). In *ICRA2023 Workshop on Pretraining for Robotics (PT4R)*.
- So Yeon Min, Hao Zhu, Ruslan Salakhutdinov, and Yonatan Bisk. 2022. [Don’t copy the teacher: Data and model challenges in embodied dialogue](#). *Preprint*, arXiv:2210.04443.
- Khanh Nguyen, Debadeepta Dey, Chris Brockett, and Bill Dolan. 2019. [Vision-based navigation with language-based assistance via imitation learning with indirect intervention](#). In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12519–12529.
- Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, and Dilek Hakkani-Tur Gokhan Tur and. 2022a. [TEACH: Task-driven Embodied Agents that Chat](#). In *Conference on Artificial Intelligence (AAAI)*.
- Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. 2022b. [Teach: Task-driven embodied agents that chat](#). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2017–2025.
- Branislav Pecher, Ivan Srba, Maria Bielikova, and Joaquin Vanschoren. 2024. [Automatic combination of sample selection strategies for few-shot learning](#). *Preprint*, arXiv:2402.03038.
- Gabriel Sarch, Yue Wu, Michael Tarr, and Katerina Fragkiadaki. 2023. [Open-ended instructable embodied agents with memory-augmented large language models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*.
- Ying Shen, Daniel Biś, Cynthia Lu, and Ismini Lourentzou. 2025. [Elba: Learning by asking for embodied visual navigation and task completion](#). In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 5177–5186.
- Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. [ALFRED: A Benchmark for Interpreting Grounded Instructions](#)

- for Everyday Tasks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. [Prog-prompt: Generating situated robot task plans using large language models](#). In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530.
- Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2023. [Llm-planner: Few-shot grounded planning for embodied agents with large language models](#). In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.
- Jesse Thomason, Michael Murray, Maya Cakmak, and Luke Zettlemoyer. 2020. [Vision-and-dialog navigation](#). In *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 394–406. PMLR.
- Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian (Shawn) Ma, and Yitao Liang. 2023. [Describe, explain, plan and select: Interactive planning with llms enables open-world multi-task agents](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 34153–34189.
- An Yang et al. 2025. [Qwen3 technical report](#). *Preprint*, arXiv:2505.09388.
- Yichi Zhang, Jianing Yang, Jiayi Pan, Shane Storks, Nikhil Devraj, Ziqiao Ma, Keunwoo Yu, Yuwei Bao, and Joyce Chai. 2022. [DANLI: Deliberative agent for following natural language instructions](#). In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1280–1298, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

A System prompt

This is the instructions prompt provided to the LLM.

Instruction Prompt

You are a household robot. You are given a dialogue snippet that contains information about the task you should execute.

You job is as follows:

Please respond in json format with the fields: intent, plan, and explanation.

Intent is one of:

- Social
- Command
- Question

if the intent is a command, generate a sequence of subgoals from the following list to execute

- Find(Object)
- Go_to(Object)
- Pick_up(Object)
- Place(Object,Receptacle)
- Open(Object)
- Close(Object)
- Toggle_on(Object)
- Toggle_off(Object)
- Slice(Object)
- Pour(Object)
- Motion(MotionType)

Any MotionType generated in the subgoals SHOULD be chosen from the following:

RotateRight, RotateLeft, MoveAhead, MoveBack, MoveLeft, MoveRight

Any Object or Receptacle generated in the subgoals or objects of interest SHOULD be chosen from the following list:

alarmclock, aluminumfoil, apple, applesliced, armchair, baseballbat, basketball, bathtub, bathtubbasin, bed, blinds, book, boots, bottle, bowl, box, bread, breadsliced, butterknife, cd, cabinet, candle, cellphone, chair, cloth, coffeemachine, coffeetable, countertop, creditcard, cup, curtains, desk, desklamp, desktop, diningtable, dish sponge, dogbed, drawer, dresser, dumbbell, egg, eggcracked, faucet, floor, floorlamp, footstool, fork, fridge, garbagebag, garbagecan, handtowel, handtowelholder, houseplant, kettle, keychain, knife, ladle, laptop, laundryhamper, lettuce, lettucesliced, lightswitch, microwave, mirror, mug, newspaper, ottoman, painting, pan, papertowelroll, pen, pencil, peppershaker, pillow, plate, plunger, poster, pot, potato, potatosliced, remotecontrol, roomdecor, safe, saltshaker, scrubbrush, shelf, shelvingunit, showercurtain, showerdoor, showerglass, showerhead, sidetable, sink, sinkbasin, soapbar, soapbottle, sofa, spatula, spoon, spraybottle, statue, stool, stoveburner, stoveknob, tvstand, tabletopdecor, targetcircle, teddybear, television, tennisracket, tissuebox, toaster, toilet, toiletpaper, toiletpaperhanger, tomato, tomatosliced, towel, towelholder, vacuumcleaner, vase, watch, wateringcan, window, winebottle

Append suffix `_X` to the Object and Receptacle when the user refers to an object mentioned in the state.

User input includes a user utterance followed by the current system STATE.

STATE includes discussed_objects, holding, and old_plan

The discussed objects is a list of string descriptions with each string describing an discussed object referenced by its id.

The ids have the form `Name_suffix` where name is the the Name is a string corresponding to the object name (e.g. pot, potato) and suffix is an integer followed `_` e.g. `Pot_0`, `Plate_0`, `Plate_1`. Each object id is unique.

holding is the object that the robot is holding.

old_plan is the plan that the robot is currently executing or has finished executing.

Each step of the plan has a suffix `_SUCCESSFUL`, `_FAILED`, or `NOT_ATTEMPTED`.

If all plan steps have status `_SUCCESSFUL`, then the old plan has finished executing.
If a step of the plan has status `_FAILED`, then this step has failed to execute. In this case, the user's sentence provides correction that should fix the plan.
The steps with status `NOT_ATTEMPTED` have not been executed. The user provides a correction or change of plan.

Your job is to interpret the user utterance and provide a new plan.
Look carefully at the `old_plan`. Write an explanation that describes your reasoning to derive interpretation of the user instruction.
Look carefully at the `discussed_objects`. If user is referring to the objects or receptacles in this list, use the appropriate IDs in the plan.

If the user refers to a new object, the plan should use the object name with the suffix `_X` where `X` is not one of the suffixes in the `discussed_objects`.

You can only place an object if you are holding it. `Place(X)` if `X` is listed as held object or following `Pick_up(X)`.
You can hold one object at a time. `Pick_up(X)` is valid iff no objects is held or after `Place(X)` for the held object.

You can slice an object such as apple, potato, etc. The object to be sliced should be placed on a receptacle such as a table or a cupboard. You should be holding a knife.

B Few-shot Examples

To guide the system, we append a set of in-context examples to the prompt. These examples were collected through interactions between an expert user and the system.

An example of an initial command of a dialogue: the execution stack is **Empty**, as `old_plan` contains no `NOT_ATTEMPTED` actions (i.e., is empty), and execution status is **Null**.

In-context Example: new command

```
<USER> lets fill a bottle; STATE: {"discussed_objects": [], "old_plan": [], "holding": ""}  
<ASSISTANT>{"explanation": "", "intent": "Command", "plan": ["Pick_up(bottle_0)",  
"Go_to(sinkbasin_0)", "Toggle_on(faucet_0)", "Toggle_off(faucet_0)"]}
```

An example where a user corrects an execution failure: the execution stack is **Not empty** as the `old_plan` contains `NOT_ATTEMPTED` actions. The execution status is **Fail** because the status of last executed action in the `old_plan` is `FAIL`.

In-context Example: correction

```
<USER> maybe use table instead;  
  
STATE: {  
  "discussed_objects": ["cup_0 is FilledWithLiquid, is on top of a countertop_0 ; ",  
"countertop_0 , has on top Knife, Cup, Pan, DishSponge, Toaster, Bread, SoapBottle,  
ButterKnife, Faucet, Knife, Faucet; ", "mug_0 is FilledWithLiquid, is Dirty, is inside a  
Cabinet ; ", "cabinet_0 , has inside Bread; ", "cabinet_1 ; ", "cabinet_2 , has inside Tomato;  
", "cabinet_3 , has inside SaltShaker, Tomato; ", "cabinet_4 ; ", "mug_1 is FilledWithLiquid,  
is on top of a CounterTop and on top of a coffeemachine_0 ; ", "sinkbasin_0 , has inside Cup;  
", "faucet_0 is on top of a countertop_0 ; ", "coffeemachine_0 is Toggled, is on top of a  
CounterTop , has on top Mug; ", "apple_0 is PickedUp, ; ", "fridge_0 is Open, is on top of a  
Floor , has inside SoapBottle, Tomato, Apple, Lettuce, Bowl, Apple, Knife, SoapBottle, Spoon,  
Potato, DishSponge; "],  
  "old_plan": [{"command": "Place(apple_0, countertop_0)", "status": "SubgoalStatus.FAILED"},  
{"command": "Pick_up(knife_0)", "status": "SubgoalStatus.NOT_ATTEMPTED"}, {"command":  
"Slice(apple_0)", "status": "SubgoalStatus.NOT_ATTEMPTED"}], "holding": "apple_0"  
}  
  
<ASSISTANT>{
```

```

    "explanation": "replace countertop_0 with diningtable_0 in the plan. Perform all steps
starting with the failed one.",
    "intent": "Command",
    "plan": ["Place(apple_0, diningtable_0)", "Pick_up(knife_0)", "Slice(apple_0)"]
}

```

An example where the user interrupts ongoing (successful) execution: the execution stack is **Not empty** because `old_plan` contains `NOT_ATTEMPTED` actions. The execution status is **Success** because the status of last executed action in the `old_plan` is `SUCCESS`.

In-context Example: interruption

```

<USER> clear the sink first and then wash the mug;

STATE: {
  "discussed_objects": ["cup_0 is FilledWithLiquid, is on top of a countertop_0 ; ",
"countertop_0 , has on top Spatula, Cup, Toaster; ", "mug_0 is Dirty, is on top of a Sink ; ",
"sinkbasin_0 , has inside SprayBottle, PepperShaker; "],
  "old_plan": [{"command": "Place(mug_0, sinkbasin_0)", "status": "SubgoalStatus.SUCCESSFUL"},
{"command": "Toggle_on(faucet_0)", "status": "SubgoalStatus.NOT_ATTEMPTED"}, {"command":
"Pick_up(mug_0)", "status": "SubgoalStatus.NOT_ATTEMPTED"}], "holding": ""
}

<ASSISTANT>{
  "explanation": "take all objects out of the sinkbasin, then place the mug into the sinkbasin
to wash and continue making coffee", "intent": "Command",
  "plan": ["Pick_up(spraybottle_0)", "Place(spraybottle_0, countertop_0)",
"Pick_up(peppershaker_0)", "Place(peppershaker_0, countertop_0)", "Pick_up(mug_0)",
"Place(mug_0, sinkbasin_0)", "Toggle_on(faucet_0)", "Pick_up(mug_0)",
"Place(mug_0, coffeemachine_0)"]
}

```

C Output Pre-processing

‘Goto(object)’ is one of the S actions of the robot’s API that navigates the robot towards object X. The executor treats the ‘Goto(object)’ command as implicit, executing it when necessary: for an object manipulation command, such as ‘Pickup(object)’, if a robot is not near object X, the executor first executes ‘Goto(object)’.

Hence, the two possible interpretations for a command “*pick up the apple*” ‘Goto(apple);Pickup(apple)’ and simply ‘Pickup(apple)’ are semantically equivalent. During the evaluation of LLM’s interpretation, to avoid penalizing models for inserting superfluous Goto(object), we remove ‘Goto(object)’ when it precedes an object manipulation command.