







# Experimenting with Planning and Reasoning in Ad Hoc Teamwork Environments with Large Language Models

Polyana Bezerra da Costa<sup>1</sup> <sup>a</sup>, Pedro Thiago Cutrim dos Santos<sup>1</sup> <sup>b</sup>,  
José Matheus Carvalho Boaro<sup>1</sup> <sup>c</sup>, Daniel de Sousa Moraes<sup>1</sup> <sup>d</sup>, Júlio Cesar Duarte<sup>2</sup> <sup>e</sup>  
and Sergio Colcher<sup>1</sup> <sup>f</sup>

<sup>1</sup> Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil

<sup>2</sup> Instituto Militar de Engenharia, Rio de Janeiro, Brazil

**Keywords:** Ad Hoc Teamwork, Large Language Models, Prompt Engineering, Autonomous Agents.

**Abstract:** Ad Hoc Teamwork environments are dynamic spaces where agents engage in activities, make decisions, and collaborate with teammates without prior coordination or complete knowledge of tasks. To effectively operate in such an environment, an ad hoc agent must be equipped with robust reasoning and planning mechanisms. Since Large Language Models (LLMs) are known for their generalization abilities, this study showcases their application in ad hoc scenarios. By modeling the robot's actions using LangChain Tools, building a semantic map, and capturing human communication interactions, we tested the LLM reasoning capabilities in three simulated scenarios involving humans and a robot. In each case, after providing contextual information, we build a meta-prompt with the question: 'How can the Robot help?'. By conducting these tests, this study highlighted the LLM's ability to infer tasks and craft action plans even in the absence of explicit verbal commands.

## 1 INTRODUCTION


As the use of robots in our daily activities becomes more common, more versatile, and more general they must be able to engage in a wide range of activities that we expect them to assist us with. While there are still places for task-specific assistants, current technologies are ready to leverage a new generation of robots that can collaborate with humans without static or pre-defined rules or even pre-established roles.


In a so-called *Ad Hoc Teamwork (AHT)* type of cooperation (Ravula, 2019), environments can be regarded as spaces where agents (that may include robots and humans) are asked to make decisions, engage in activities, and cooperate without prior knowledge about the surrounding world, their teammates, or even the task they are addressing (Ribeiro et al., 2023). To adequately cooperate in such an envi-


ronment, participants, called ad hoc agents, must be highly adaptable while identifying the task at hand and collaborating with other agents to achieve a common goal without pre-defined protocols (Mirsky et al., 2022).


The versatility and adaptability necessary in AHT environments naturally require new mechanisms to enhance a robot's reasoning and planning abilities. Large Language Models (LLMs) have been used in many scenarios where generalizations and adaptations are necessary. Given the wide range of data these models are usually trained on, LLMs embed such a vast knowledge that enables them to achieve high generalization capabilities and perform tasks in diverse contexts, including acting as reasoning and planning agents (Zhao et al., 2023). But, while LLMs used for planning and reasoning have, up to now, mainly relied on prompts that already specify the task that should be executed (Zheng et al., 2023; Xu et al., 2023; Huang et al., 2023), this can not be the case in AHT scenarios.


In this work, we investigate the embedding of LLMs into ad hoc agents to enhance their reasoning capabilities and cooperation with human teammates on the fly. We designed a mobile robot and conducted


<sup>a</sup>  <https://orcid.org/0009-0005-8691-8107>

<sup>b</sup>  <https://orcid.org/0009-0002-8038-8713>

<sup>c</sup>  <https://orcid.org/0000-0002-4456-9050>

<sup>d</sup>  <https://orcid.org/0000-0003-2995-3115>

<sup>e</sup>  <https://orcid.org/0000-0001-6656-1247>

<sup>f</sup>  <https://orcid.org/0000-0002-3476-8718>

tests of the LLM’s reasoning and planning ability in three simulated scenarios involving human interactions. In each scenario, the absence of an explicit task and direct human commands required the robot to acquire a contextualization model of its surroundings, including objects, their places, people, and what they are currently saying. In our proposal, we submit the same prompt template (which we call a *meta-prompt*) to the LLM in every scenario, asking the robot to determine which action it would take given each specific situation and context. Although the meta-prompt structure remains the same, the LLM’s responses produced specific action plans corresponding to a set of possible actions tailored for each proper scenario.

More specifically, we use *LangChain Agents* to model the robot’s actions (Briggs and Ingham, 2023). With these agents, the LLM has access to *Tools*, which (in our case) are pieces of code that encapsulate the robot’s basic capabilities, such as moving, grabbing/releasing something, and speaking. To build the context of the environment in a certain timeframe, we construct a semantic map with the objects and agents (human and robotic) present in that environment and their positions. Additionally, we transcribe what the human agents are saying. We pass all this info in a meta-prompt, always asking the same question: “How can the robot help?”. Figure 1 shows an overview of our approach.

With this work, we aim to evaluate the suitability of LLMs as planning tools for ad hoc environments and how such environments could benefit from their remarkable ability to reason in different contexts and the large amount of knowledge embedded in them. Since our purpose was not to model or evaluate the techniques of acquiring context data, all the environmental information was manually described for each scenario (instead of being captured by microphones, cameras, or other sensors/devices).

The remainder of the paper is organized as follows. Section 2 presents an overview of recent works in Ad Hoc Teamwork and LLMs for Robotics. Section 3 details our main method of investigation, while Section 4 contains some of our results. Finally, Section 5 concludes the paper with our findings and ideas for future works.

## 2 RELATED WORK

This section is grouped into two categories. In the first subsection, we provide an overview of the latest research in Ad Hoc Teamwork. Conversely, in the second one, we detail the application of LLMs in robotics.

### 2.1 Ad Hoc Teamwork

To manage the uncertainty and high ability to adapt that ad hoc environments require from their agents, most work in AHT defines some information beforehand. Some works model a set of possible tasks, while others model the type of possible teammates, others store previous interactions to compare with the current one, etc.

The PLASTIC Model (Barrett et al., 2017), one of the classical approaches for ad hoc teamwork, models both the environment’s dynamics and the teammate’s behaviors. PLASTIC has a library with behaviors of previous teammates and policies of actions for the ad hoc agent based on these behaviors. This model assumes that there is some similarity between the way the current teammate acts in comparison with the previous one. However, this might not be true in every context. So, at the runtime, the ad hoc agent compares the actions of a teammate to the ones existing in its library to select an appropriate policy.

Another approach is to model all possible tasks in a context. Although the ad hoc agent might not know the task performed at the moment, in these works, it normally possesses a library of all possible tasks (Ribeiro et al., 2021). In (Melo and Sardinha, 2016), the authors define the set of possible tasks as matrices of actions, and the AHT problem is treated as a partially observable Markov decision problem. In this context, the agent must observe the behavior of the teammates to identify the task. The disadvantages of this approach are having to characterize each task in advance and expecting the teammates to behave accordingly and perform only tasks specified in the library.

In our work, differently from the others, we want to eliminate the need to explicitly model all possible tasks and teammate’s behavior. Instead, we let the LLM autonomously analyze the context to discover the task at hand and plan how the ad hoc agent can collaborate based on its own capabilities, allowing the agent to dynamically respond to evolving scenarios without the knowledge of predefined tasks or behavior specifications.

### 2.2 LLMs in Robotics

With the recent breakthroughs in the LLMs field, they have been applied across diverse contexts, including robotics. Even before the introduction of LLMs, works with initial language models such as BERT already catered to robotic-related problems (Devlin et al., 2019). In this scenario, FILM (Min et al., 2021) presented a modular method based on BERT that pro-

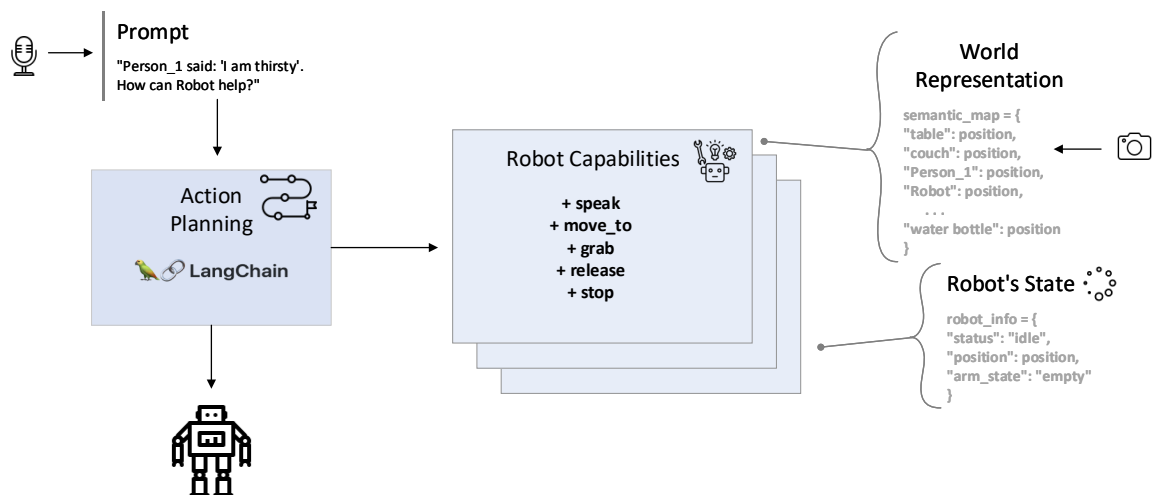


Figure 1: Overview of our method.

cesses language instructions and organizes visual input into a semantic map to fulfill a given task, outputting a series of navigation commands and actions.

Similarly, in LangSuitE (Zheng et al., 2023), the authors built a semantic mapping module to capture what is happening in the environment. LangSuitE specifies the task to be executed, characterizes the environment, and defines the robot's actuation all in the same prompt. They also included a task example and how it should be performed in the prompt for few-shot learning. The authors used OpenAI's GPT 3.5<sup>1</sup> for controlling and planning the robot's actions and for communication between agents.

Recently, more robust models for solving robotic problems were introduced, such as RT-2 (Brohan et al., 2023), a visual-language-action model. RT-2 is an end-to-end model based on PALM-2 that maps robot observations to actions using textual and visual data. This unified model combined the problem-solving and reasoning abilities of LLMs with the visual grounding/understanding abilities of visual-language models to help robots perform several tasks in real-world environments. RT-2 is able to receive a command in text, extract information from image data, reason over all this information, and output generalized instructions for robotic control, such as vectors for positional and rotational change.

Along with planning the robot's action, some works use LLM to generate actual code to execute those actions. RoboTool (Xu et al., 2023) is a system geared towards robotic arms and mobile robots that uses LLMs to map commands in natural language to executable code. Each single prompt encapsulates a description of the robot in itself (mobile or robotic arm), objects in an environment and their position,

as well as the robot's task and constraints that the robot must follow to fulfill it. Based on this prompt, RoboTool outputs a Python code that is responsible for controlling the robot's actions.

In addition, Instruct2Act (Huang et al., 2023) also uses an LLM to generate Python code for robotic manipulation, distinguishing itself by receiving multi-modal instructions. The system is composed of a perception module that uses foundation models to locate objects in a scene, classify and segment them, and an action module that interprets a given command along with the perception information; Then, it generates actions that are transformed into executable code.

Furthermore, (Vemprala et al., 2023) applied OpenAI's ChatGPT<sup>2</sup> to a pipeline for solving robotic tasks, also outputting Python code. At first, the authors define a basic Python API with the robot's actions, the function's names, and their parameters, but no actual code within these functions. Then, in the prompt, the authors provide a clear description of the task and the functions defined in the API (and their descriptions). ChatGPT is then asked to output a Python code that can fulfill the specified task. The authors tested this pipeline in ten different scenarios that involved logical, geometrical, and mathematical reasoning, navigation, and manipulation, among others. With extensive tests, the authors showcased the potential of ChatGPT for robotic tasks and how it can be able to reason and plan in the most different contexts.

Finally, in a work that is the most similar to ours, (Xiong et al., 2023) integrated texts, sensor data, prompts, LangChain's Agents, and LLMs to model the context that the robot is inserted and reason over all this information to plan the robots' actions. The

<sup>1</sup><https://platform.openai.com/docs/models/gpt-3-5>

<sup>2</sup><https://openai.com/blog/chatgpt/>

authors define categories of requests and examples of question-answering for each category. At first, the LLM has to classify the user's request based on previously defined categories using the right prompt template. With the final prompt, the LLM outputs a sequence of actions in natural language that fulfill the user's request.

Similarly to the works cited above, we also use LLMs (GPT-4) to reason over various information and generate actionable plans for robots. However, different from these works, in ad hoc scenarios, the task to be addressed is not always defined, and there may be no direct command or request from the user. In such cases, the LLM has to infer the task at hand and formulate a collaborative action plan for how the robot can fulfill the team's goal. With our work, we want to build a system that works for ad hoc scenarios, addressing tasks without predefined specifications. In addition, the way that we model our prompts saves several tokens. Section 3 explains our method in detail and shows three examples that illustrate the potential of our method for AHT scenarios.

### 3 PLANNING WITH LLMs

This section presents a detailed description of each part of our methodology. We start by defining what the robot (ad hoc agent) can do by coding its actions in Python. Subsequently, we characterize the environment in which the robot is inserted, identifying the objects, teammates, and their positions. We also capture what the teammates might be saying. We then use all this contextual information to build a generic prompt and submit it to GPT-4 to get an action plan.

Since we are not focused on the techniques for acquiring contextual data, the visual and audio data mentioned in the following sections were not captured by cameras and microphones; these scenarios have been manually described.

#### 3.1 Modeling the Robot's Capabilities with LangChain Tools

We start by defining the robot's capabilities – the actions it can perform. To do so, we used LangChain's Agents, a functionality that allows the creation of custom tools for an LLM Agent in Python.

Tools are functions or methods that receive their input from an LLM, and their output is sent back to the LLM (Briggs and Ingham, 2023). Anything can be done inside these functions, but their return should always be a prompt sent back to the LLM, so it should contain valuable information to the LLM.

In our context, each tool is a function that represents a robot's capability, along with an extra tool that handles a map of the environment. We defined a mobile robot that can move in spaces, grab or release objects, and speak utterances. So, we coded six tools and started by defining the inputs of our methods and the robot's capabilities, as shown in Listing 1 and Listing 2. We also defined a dictionary that contains the robot's position, its state (idle or not), and the state of the robot's arm (empty or not).

```
from pydantic.v1 import BaseModel, Field

class ObjectPositionInput(BaseModel):
    object_name: str = Field(description="Name of the object.")

class Text2SpeakInput(BaseModel):
    text_input: str = Field(description="Text that robot need to speak.")

class MapPositionInput(BaseModel):
    n: int = Field(description="Desired position described by an integer.")
```

Listing 1: Definition of the methods' inputs.

```
def speak(text_input: str):
    """
    Code logic to make the Robot speak
    """
    return "Robot spoke: '{}'.format(
        text_input)

def get_object_position(object_name: str,
    map_data: dict = getMapData()) -> str:

    k = match_key(object_name.lower(),
        map_data)
    return "Object '{}' is at position '{}'.format(k, str(map_data[k]))

def move_to(n: int) -> str:
    """
    Code logic to make the Robot Move
    """
    robot_info['status'] = 'moving'
    robot_info['position'] = n
    robot_info['status'] = 'idle'

    if robot_info['arm'] != 'empty':
        map_data[robot_info['arm']] = n
    return "Robot at position {}".format(
        robot_info['position'])

def grab(object_name: str) -> str:
    """
    Code logic to make the robot
    grab the object.
    """
    robot_info['arm'] = k
```

```

    return "Robot grabbed object '{}'.".format(k)

def release(object_name: str) -> str:
    """
    Code logic to make the robot
    release the object.
    """
    aux = robot_info['arm']
    robot_info['arm'] = 'empty'
    return "Robot released object '{}' at
    location {}".format(aux, robot_info['
    position'])

def stop(action: str) -> str:
    """
    Code logic to stop the robot.
    """
    robot_info['status'] = 'idle'
    return 'There is nothing the Robot can do
    in this situation, status switched to
    idle.'

```

Listing 2: Definition of the robot's capabilities.

### 3.2 Context Modeling

To provide contextual awareness to our robot, we represented the world that the robot is inserted in a semantic map, similarly to (Min et al., 2021; Zheng et al., 2023). The semantic map is a dictionary with the name and position (id) of every object and agent present in the scene. Since we were simulating AHT environments, we manually defined the objects and agents' positions. An example of the semantic map used in our experiments is shown in Listing 3.

```

map_data = {
    "Person_1": 1,
    "Person_2": 2,
    "Robot": 3,
    "water bottle": 4,
    "couch": 5,
    "chair": 6,
    "glass": 7,
    "table": 8,
    "computer": 8,
    "cellphone": 9
}

```

Listing 3: Definition of a semantic map.

For context modeling, we also include what the agents say.

### 3.3 Prompt Engineering

After modeling the robot's capabilities and environment, we need to craft the prompt that will be sent to the LLM Agent. The LLM we chose for our agent

was GPT-4 (OpenAI, 2023) due to its impressive performance in several benchmarks.

We defined an agent for zero-shot tasks, meaning it does not have previous examples to guide its response. With zero-shot agents, forcing the agent to follow instructions is easier.

Based on how we modeled the environment, the semantic map is not included in the prompt, which saves several tokens. With this, every time the LLM Agent wishes to know the position of a particular object or teammate, it needs to call the responsible tool (*getObjectPosition*) for the semantic map.

The LLM Agent chooses which tools to use based solely on their descriptions. The final prompt contains the Python objects for each tool, with their names, descriptions, and corresponding function names. The internal code of each tool is not included in the prompt, which also saves tokens. This code is shown in Listing 4.

```

from langchain import OpenAI
from langchain.agents import import
    initialize_agent, Tool

from langchain.agents import AgentType
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(temperature=0, model="gpt-4",
    openai_api_key=OPENAI_KEY)

tools = [
    Tool(
        name="Speak", func=speak, args_schema=
        Text2SpeakInput,
        description="Make the robot speak the
        input text."
    ),
    Tool(
        name="Get Object Position", func=
        get_object_position,
        args_schema=ObjectPositionInput,
        description="Get a position of a
        specific object by giving its name."
    ),
    Tool(
        name="Move", func=move_to, args_schema=
        MapPositionInput,
        description="Move to a specific
        position on the map. The input is
        an integer that defines the
        position (id)."
    ),
    Tool(
        name="Grab", func=grab, args_schema=
        ObjectPositionInput,
        description="Grab an object that is
        placed in a specific position on
        the map. The input is the object
        name to be grabbed."
    ),
]

```

```

Tool(
    name="Release", func=release,
    args_schema=ObjectPositionInput,
    description="Release something that is
    grabbed at the current robot
    location. The input is the object
    name to be released."
),
Tool(
    name="Stop", func=stop,
    description='When there is nothing an
    agent can do in a situation or when
    the agent has finished a task, use
    this function.',
)
]

agent = initialize_agent(
    tools, llm, agent=AgentType.
    ZERO_SHOT_REACT_DESCRIPTION, verbose=
    True)

```

Listing 4: Definition of a Langchain Agent.

After creating the agent, we can now run it by passing a prompt. The final prompt contains only what the teammates might have said and a generic question asking what the robot could do in that context, as illustrated by Listing 5.

```

agent.run("Person_1 said: 'I am thirsty.'.
How can Robot help?")

```

Listing 5: Submitting the final prompt to the agent.

After reasoning and discovering what is the task to be addressed, the agent may need to access the semantic map to get the position of a specific object. However, the LLM’s output may not always precisely match a term on the map. In one of our examples, the LLM agent decided that it needed to move to a ‘water dispenser’, which does not exist on the semantic map, but ‘water bottle’ does. To overcome this, we defined a term similarity function that always compares the LLM’s output to each term on the map and returns the most similar one.

## 4 EXPERIMENTS

This section describes our design and experiments in three scenarios that follow AHT environments’ requirements. The ad hoc agent (our mobile robot) has to infer the task on the fly and collaborate with human teammates to fulfill a common goal. There is no previous agreement on how they will cooperate, and the agent gathers information about its surroundings to help in its reasoning process by using a semantic map and speech recognition data. All three examples share the same semantic map depicted on Listing 3.

Figure 2 shows the prompt and action plan executed by the robot for the first example. In this example, the person asks if anyone has seen his phone. The LLM correctly inferred the task. First, the robot tries to find the cell phone in the environment. After finding it, the robot moves to the object’s position and grabs it. Finally, the action plan consists of leaving the cell phone in one place and communicating where it is. It is worth mentioning that, while all actions occur, the robot engages in continuous communication with the person, which can help the understanding and increase the person’s confidence in the robot’s actions.

**Prompt:**

"Person\_1 said: 'Has anyone seen my phone?'.  
How can Robot help?"

**LLM Agent Response:**

```

> Entering new AgentExecutor chain...

I need to find the phone for Person_1.
Action: Get Object Position
Action Input: Phone
Observation: Object 'cellphone' is at position '9'.

Thought: I know where the phone is. I need to move to that
position.
Action: Move
Action Input: 9
Observation: Robot at position 9

Thought: I am now at the position of the phone. I need to
grab it.
Action: Grab
Action Input: Phone
Observation: Robot grabbed object 'cellphone'.

Thought: I have the phone. I need to return it to Person_1.
Action: Speak
Action Input: I have found your phone, Person_1.
Observation: Robot spoke: 'I have found your phone,
Person_1.'.

Thought: I have informed Person_1 that I found their phone.
Now I need to release the phone.
Action: Release
Action Input: Phone
Observation: Robot released object 'cellphone' at location 9

Thought: I have released the phone. Now I need to inform
Person_1 where I left it.
Action: Speak
Action Input: I have left your phone at position 9.
Observation: Robot spoke: 'I have left your phone at position 9.'.

Thought: I have completed the task.
Action: Stop
Action Input:
Observation: There is nothing the Robot can do in this situation,
status switched to idle.

Thought: I now know the final answer
Final Answer: The robot helped by finding the phone and
informing Person_1 about its location.

> Finished chain

```

Figure 2: Prompt for task 1 and the resulting reasoning process and action planning.

The second example is shown in Figure 3. Similarly to example 1, a person asks about the location of an object in the environment, in this case, the couch. With the help of the LLM, the robot correctly infers that the current task is to find the couch’s position. Differently from example 1, in which the robot got the phone’s position but also grabbed it, the robot cannot grasp it due to the couch’s size and weight. Even though this restriction is not explicitly stated in

the definition of the robot’s capabilities, the LLM was able to infer that the ‘grab’ function was unnecessary.

In this instance, the LLM’s strategy involved locating the position of the couch and communicating this information to the person.

**Prompt:**

"Person\_2 said: "Where is the couch?". How can Robot help?"

**LLM Agent Response:**

```
> Entering new AgentExecutor chain...
I need to find the position of the couch.
Action: Get Object Position
Action Input: Couch
Observation: Object 'couch' is at position '5'.
Thought: I now know the location of the couch.
Action: Speak
Action Input: The couch is at position 5.
Observation: Robot spoke: 'The couch is at position 5.'.
Thought: I have successfully informed Person_2 about the location of the couch.
Action: Stop
Action Input:
Observation: There is nothing the Robot can do in this situation, status switched to idle.
Thought: I now know the final answer
Final Answer: The robot helped by informing Person_2 that the couch is at position 5.
> Finished chain.
```

Figure 3: Prompt for task 2 and the resulting reasoning process and action planning.

The last example is illustrated in Figure 4, where Person 1 mentions that he is thirsty, and after a detailed reasoning process, the ad hoc agent (robot) concludes that it can cooperate by getting a glass of water to Person 1. With the help of an LLM, the robot inferred the task and generated an action plan based on its capabilities. Even when the robot encountered an action beyond its predefined capabilities (fill the glass), the agent could adapt by invoking an alternative action (speak). Although this shows the remarkable adaptability of LLMs, by dynamically adjusting and employing available functionalities to address unanticipated tasks, it is also a point for improvement in our method since just saying "Filling the glass ..." does not equate to physically executing the task.

## 5 CONCLUSIONS

This paper presented a method for reasoning and planning in ad hoc teamwork (AHT) scenarios with Large Language Models (LLMs). Due to the fluidity of such environments, we wanted to equip ad hoc agents with mechanisms to allow them to adapt easily to different contexts and tasks. Since LLMs have been showing remarkable generalization abilities, we investigated their potential in AHT scenarios with robots and human teammates.

**Prompt:**

"Person\_1 said: 'I am thirsty.'. How can Robot help?"

**LLM Agent Response:**

```
> Entering new AgentExecutor chain...
The robot can help by getting a glass of water for Person_1. First, I need to find out where the glass is.
Action: Get Object Position
Action Input: Glass
Observation: Object 'glass' is at position '7'.
Thought: Now that I know where the glass is, I need to move to that position to grab it.
Action: Move
Action Input: 7
Observation: Robot at position 7
Thought: Now that I'm at the position of the glass, I can grab it.
Action: Grab
Action Input: Glass
Observation: Robot grabbed object 'glass'.
Thought: Now that I have the glass, I need to find out where the water dispenser is to fill the glass.
Action: Get Object Position
Action Input: Water Dispenser
Observation: Object 'water bottle' is at position '4'.
Thought: Now that I know where the water dispenser is, I need to move to that position to fill the glass.
Action: Move
Action Input: 4
Observation: Robot at position 4
Thought: Now that I'm at the position of the water dispenser, I can fill the glass.
Action: Speak
Action Input: Filling the glass with water.
Observation: Robot spoke: 'Filling the glass with water.'.
Thought: Now that I have filled the glass with water, I need to find out where Person_1 is to give them the glass.
Action: Get Object Position
Action Input: Person_1
Observation: Object 'person_1' is at position '1'.
Thought: Now that I know where Person_1 is, I need to move to that position to give them the glass.
Action: Move
Action Input: 1
Observation: Robot at position 1
Thought: Now that I'm at the position of Person_1, I can give them the glass of water.
Action: Release
Action Input: Glass
Observation: Robot released object 'glass' at location 1
Thought: I have given the glass of water to Person_1. My task is complete.
Action: Stop
Action Input:
Observation: There is nothing the Robot can do in this situation, status switched to idle.
Thought: I now know the final answer
Final Answer: The robot helped by getting a glass of water for Person_1.
> Finished chain.
```

Figure 4: Prompt for task 3 and the resulting reasoning process and action planning.

In our experiments, we used LangChain Tools to define the robot’s capabilities and modeled the environment in which the robot is inserted for contextual awareness. We showcased three scenarios in which this information is combined in a generic prompt asking how the agent can help in that context, submitting it to GPT-4.

For every example, the LLM could reason and infer the task to be performed, defining a plan of action to achieve it. This showed the potential of LLMs as reasoning and planning mechanisms in ad hoc environments.

For future works, we intend to adjust the prompt to prevent the LLM from 'cheating,' which is adjusting the functionalities to address unanticipated tasks that are not entirely equivalent. We also want to include more complex scenarios and more examples, especially the ones in real-world settings, receiving data from different sensors (cameras and microphones) and connecting the tools to the robot operating system so that it can actually perform the actions.

## ACKNOWLEDGEMENTS

This work was partially funded by the National Council for Scientific and Technological Development (CNPQ), under grant number 141809/2020-5. In addition, this material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-22-1-0475.

## REFERENCES

- Barrett, S., Rosenfeld, A., Kraus, S., and Stone, P. (2017). Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, 242:132–171.
- Briggs, J. and Ingham, F. (2023). *Langchain: Introduction and getting started*. Pinecone.
- Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Chormanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., et al. (2023). Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.
- Huang, S., Jiang, Z., Dong, H., Qiao, Y., Gao, P., and Li, H. (2023). Instruct2act: Mapping multi-modality instructions to robotic actions with large language model. *arXiv preprint arXiv:2305.11176*.
- Melo, F. S. and Sardinha, A. (2016). Ad hoc teamwork by learning teammates' task. *Autonomous Agents and Multi-Agent Systems*, 30:175–219.
- Min, S. Y., Chaplot, D. S., Ravikumar, P., Bisk, Y., and Salakhutdinov, R. (2021). Film: Following instructions in language with modular methods. *arXiv preprint arXiv:2110.07342*.
- Mirsky, R., Carlucho, I., Rahman, A., Fosong, E., Macke, W., Sridharan, M., Stone, P., and Albrecht, S. V. (2022). A survey of ad hoc teamwork research. In *European Conference on Multi-Agent Systems (EU-MAS)*.
- OpenAI (2023). Gpt-4 technical report.
- Ravula, M. C. R. (2019). *Ad-hoc teamwork with behavior-switching agents*. PhD thesis, University of Texas.
- Ribeiro, J. G., Faria, M., Sardinha, A., and Melo, F. S. (2021). Helping people on the fly: Ad hoc teamwork for human-robot teams. In *Progress in Artificial Intelligence: 20th EPIA Conference on Artificial Intelligence, EPIA 2021, Virtual Event, September 7–9, 2021, Proceedings 20*, pages 635–647. Springer.
- Ribeiro, J. G., Rodrigues, G., Sardinha, A., and Melo, F. S. (2023). Teamster: Model-based reinforcement learning for ad hoc teamwork. *Artificial Intelligence*, 324:104013.
- Vemprala, S., Bonatti, R., Buckner, A., and Kapoor, A. (2023). Chatgpt for robotics: Design principles and model abilities. *Microsoft Auton. Syst. Robot. Res.*, 2:20.
- Xiong, H., Bian, J., Yang, S., Zhang, X., Kong, L., and Zhang, D. (2023). Natural language based context modeling and reasoning with llms: A tutorial. *arXiv preprint arXiv:2309.15074*.
- Xu, M., Huang, P., Yu, W., Liu, S., Zhang, X., Niu, Y., Zhang, T., Xia, F., Tan, J., and Zhao, D. (2023). Creative robot tool use with large language models.
- Zhao, P., Jin, Z., and Cheng, N. (2023). An in-depth survey of large language model-based artificial intelligence agents. *arXiv preprint arXiv:2309.14365*.
- Zheng, Z., Wang, M., and Zixia Jia, B. T. (2023). Langsuite: Controlling, planning, and interacting with large language models in embodied text environments.