IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 22 February 2024, accepted 3 April 2024, date of publication 15 April 2024, date of current version 24 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3387941

RESEARCH ARTICLE

ChatGPT for Robotics: Design Principles and Model Abilities

SAI H. VEMPRALA[1], (Member, IEEE), ROGERIO BONATTI[2], (Member, IEEE), ARTHUR BUCKER[3], (Student Member, IEEE), AND ASHISH KAPOOR[1]

[1]Scaled Foundations, Kirkland, WA 98033, USA
[2]Microsoft Corporation, Redmond, WA 98072, USA
[3]Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Corresponding author: Sai H. Vemprala (mail@saihv.com)

ABSTRACT This paper presents an experimental study regarding the use of OpenAI's ChatGPT for robotics applications. We outline a strategy that combines design principles for prompt engineering and the creation of a high-level function library which allows ChatGPT to adapt to different robotics tasks, simulators, and form factors. We focus our evaluations on the effectiveness of different prompt engineering techniques and dialog strategies towards the execution of various types of robotics tasks. We explore ChatGPT's ability to use free-form dialog, parse XML tags, and to synthesize code, in addition to the use of task-specific prompting functions and closed-loop reasoning through dialogues. Our study encompasses a range of tasks within the robotics domain, from basic logical, geometrical, and mathematical reasoning all the way to complex domains such as aerial navigation, manipulation, and embodied agents. We show that ChatGPT can be effective at solving several of such tasks, while allowing users to interact with it primarily via natural language instructions. In addition to these studies, we introduce an open-sourced research tool called *PromptCraft*, which contains a platform where researchers can collaboratively upload and vote on examples of good prompting schemes for robotics applications, as well as a sample robotics simulator with ChatGPT integration, making it easier for users to get started with using ChatGPT for robotics. Videos and blog: aka.ms/ChatGPT-Robotics PromptCraft, AirSim-ChatGPT code: https://github.com/microsoft/PromptCraft-Robotics

INDEX TERMS Large language models, robotics, language understanding, code generation, perception.

I. INTRODUCTION

The rapid advancement in natural language processing (NLP) has led to the development of large language models (LLMs), which started with models such as BERT [2], GPT [3], [4], and Codex [5], and more recently, LLaMa [6], [7], Mistral [8] that are revolutionizing a wide range of applications. These models have achieved remarkable results in various tasks such as text generation, machine translation, and code synthesis, among others. A recent addition to this collection of models was the OpenAI ChatGPT [1], a pretrained generative text model which was finetuned using human feedback. Unlike previous models which operate mostly upon a single prompt, ChatGPT provides particularly impressive interaction skills through dialog, combining text generation with code synthesis. Our goal in this paper is to investigate if and how the abilities of ChatGPT can generalize to the domain of robotics.

Robotics systems, unlike text-only applications, require a deep understanding of real-world physics, environmental context, and the ability to perform physical actions. A generative robotics model needs to have a robust commonsense knowledge and a sophisticated world model, and the ability to interact with users to interpret and execute commands in ways that are physically possible and that makes sense in the real world. These challenges fall beyond the original scope of language models, as they must not only understand the meaning of a given text, but also

The associate editor coordinating the review of this manuscript and approving it for publication was Hongli Dong.

translate the intent into a logical sequence of physical actions.

In recent years there have been different attempts to incorporate language into robotics systems. These efforts have largely focused on using language token embedding models, LLM features, and multi-modal model features for specific form factors or scenarios. Applications range from visual-language navigation [9], [10], [11], language-based human-robot interaction [12], [13], and visual-language manipulation control [14], [15], [16]. However, despite the potential advantages of using LLMs in robotics, most of the existing approaches are restricted by a rigid scope and limited set of functionalities, or by their open-loop nature that does not allow for fluid interactions and behavior corrections from user feedback.

Large language models also show promise in zero-shot robotics scenarios when tasked with high-level agent planning [17], [18] or code generation [19], [20]. These early demonstrations inspired us to investigate ChatGPT as a potentially more versatile tool for the robotics domain, as it incorporates the strengths of natural language and code generation models along with the flexibility of dialogue. ChatGPT's ability to engage in a free-form dialog and capture long context allows users to interact with the model in a more natural fashion, with flexible behavior correction. At the same time, ChatGPT's extensive knowledge of mathematical constructs, geometry, commonsense afford it a much higher ability to understand and reason about the physical world compared to older LLM-based approaches which mainly focused on task planning that was dependent on a fixed set of functions and behaviors.

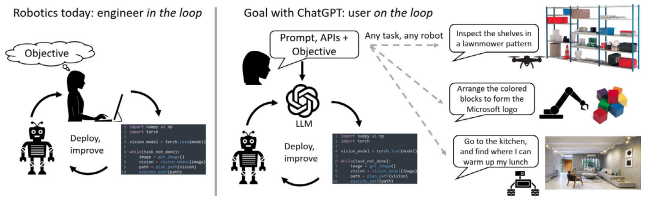


FIGURE 1. Current robotics pipelines require a specialized engineer *in the loop* to write code to improve the process. Our goal with ChatGPT is to have a (potentially non-technical) user *on the loop*, interacting with the language model through high-level language commands, and able to seamlessly deploy various platforms and tasks.

In this paper, we aim to demonstrate the potential of ChatGPT for robotics applications. We outline a key concept that unlocks the ability to solve robotics applications with ChatGPT, which is the creation of a high-level function library. Given that robotics is a diverse field where several platforms, scenarios, and tools exist, there exists an extensive variety of libraries and APIs. Instead of asking LLMs to output code specific to a platform or a library, which might involve extensive finetuning, we instead create a simple high-level function library for ChatGPT to deal with which can then be linked in the back-end to the actual APIs for the platforms of choice. Thus, we allow ChatGPT to parse user intent from natural dialog, and convert that to a logical chaining of high-level function calls. We also outline several prompt engineering guidelines that help ChatGPT solve robotics tasks.

Recent large models such as BLIP-2 [21], LLaVa [22] have shown that fusing visual and language features into the same representation allows for capabilities such as visual question answering and captioning. While this is a very powerful capability, these models excel at general descriptions of images and to an extent reasoning about them. However, robotics tasks often require a more precise and detailed understanding of the environment, such as the ability to extract object locations, or to perform geometric reasoning, which potentially requires more specialized models. In this work, we assume that the large language model can be provided with the necessary information about the environment using appropriate tools, and focus on the analysis of LLMs on how they can use this information to formulate high level behaviors and the necessary code for solving the task at hand.

Our research shows that ChatGPT is capable of solving various robotics-related tasks in a zero-shot fashion, while adapting to multiple form factors, and allowing for closed-loop reasoning through conversation. In addition, we aim to show current model limitations, and provide ideas on how to overcome them. Our main contributions are listed below:

- We demonstrate a pipeline for applying ChatGPT to robotics tasks. The pipeline involves several prompting techniques such as free-form natural language dialogue, code prompting, XML tags, and closed-loop reasoning. We also show how users can leverage a high-level function library that allows the model to quickly parse human intent and generate code for solving the problem;
- We experimentally evaluate ChatGPT's ability to execute a variety of robotics tasks. We show the model's capabilities and limitations when solving mathematical, logical, and geometrical operations, and then explore more complex scenarios involving embodied agents, aerial navigation, and manipulation. We include both simulation and real-world experiments that result from ChatGPT's plans;
- We introduce a collaborative open-source platform, *PromptCraft*, where researchers can work together to provide examples of positive (and negative) prompting strategies when working with LLMs in the robotics context. Prompt engineering is a mostly empirical science, and we want to provide a simple interface for researchers to contribute with knowledge as a community. Over time we aim to provide different environments where users can test their prompts, and welcome new contributions;
- We release a simulation tool that builds on Microsoft AirSim [23] combined with a ChatGPT integration. This *AirSim-ChatGPT* simulation contains a sample environment for drone navigation and aims to be a starting point for researchers to explore how ChatGPT can enable robotics scenarios.

With this work we hope to open up new opportunities and avenues for future research fusing LLMs and robotics. We believe that our findings will inspire and guide further research in this exciting field, paving the way for the development of new, innovative robotics systems that can interact with humans in a natural, intuitive manner. For more details, we encourage readers to view detailed videos of our experiments in the project webpage.

II. ROBOTICS WITH CHATGPT

Prompting LLMs for robotics control poses several challenges, such as providing a complete and accurate descriptions of the problem, identifying the right set of allowable function calls and APIs, and biasing the answer structure with special arguments. To make effective use of ChatGPT for robotics applications, we construct a pipeline composed of the following steps:

1) First, we define a high-level robot function library. This library can be specific to the form factor or scenario of interest, and should map to actual implementations on the robot platform while being named descriptively enough for ChatGPT to follow;
2) Next, we build a prompt for ChatGPT which describes the objective while also identifying the set of allowed high-level functions from the library. The prompt can also contain information about constraints, or how ChatGPT should structure its responses;
3) The user stays on the loop to evaluate code output by ChatGPT, either through direct analysis or through simulation, and provides feedback to ChatGPT on the quality and safety of the output code;
4) After iterating on the ChatGPT-generated implementations, the final code can be deployed onto the robot.

We show a visual depiction of this pipeline in Figure 2 for the example of a household robot.

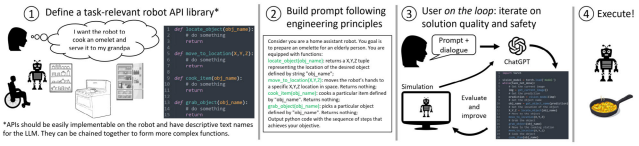


FIGURE 2. Robotics pipeline employing ChatGPT with the user *on the loop* to evaluate the output's quality and safety.

A. CONSTRUCTION AND DESCRIPTION OF THE ROBOTICS API LIBRARY

Robotics being a well-established field, there already exists a multitude of libraries, either black-box or open-source, that can be used for basic functionalities in both the perception and action domains (*e.g.* object detection and segmentation, mapping, motion planning, controls, grasping). If properly specified in the prompt, the LLM is able to use these pre-defined functions for robot reasoning and execution.

One important prompt design requirement is that all API names must be descriptive of the overall function behavior. Clear names are essential to allow the LLM to reason over functional connections between APIs and produce the desired outcome for the problem. Hence, we can define high-level functions, which act as wrappers over actual implementations from the respective libraries. For example, a function named `detect_object(object_name)` could internally link to an OpenCV function or a computer vision model, whereas something like `move_to(x, y, z)` could internally invoke a motion planning and obstacle avoidance pipeline along with the appropriate low-level motor commands for a drone. Listing such a collection of high-level functions in the prompt is key in allowing ChatGPT to create logical sequences of behavioral primitives, and in generalizing to different scenarios and platforms.

Depending on the context, we recommend explaining the function of APIs and if needed, breaking them down into sub-components with clear inputs and outputs, similar to code documentation. In Figure 3 we present an example of a good API prompting strategy for a home cook robot scenario. The strategy presented allows ChatGPT to reason about the order and content of tasks according to the functions the robot is actually able to execute. In contrast, we refer the interested reader to Appendix A-A for an example of how ChatGPT reasons when no API guidance is given, which leads to a unbounded text-based answer, or to Appendix A-B for an example of API under-specification, which leads to hallucinations over function call parameters.

We note that unlike the brittle structure of classical symbolic AI, which required rigid pre-defined relationships between objects and functions, LLMs are capable of defining new functions and concepts altogether when relevant to a particular problem. This capability confers flexibility and robustness to LLMs when dealing with robotics applications. Figure 4 shows how ChatGPT can create new high-level concepts and even low-level code when needed to solve a problem, even fusing existing APIs. The user on the loop can take advantage of this ability as a design strategy, and iteratively define new APIs with the help of the LLM when the current ones are not sufficient to solve the task at hand.

B. CLEAR DESCRIPTION OF THE TASK DETAILS IN THE PROMPT

By providing a clear and concise description of the desired robotics task and its context, ChatGPT can generate more accurate responses. A good context description should contain, besides robot APIs:

- Constraints and requirements: specify constraints or requirements that are relevant to the task. If the task involves moving objects, you might specify the weight, size, and shape of the objects to be moved.
- Environment: describe the environment in which the robotics task is taking place. For example, if the task is to navigate a maze, you might describe the size and shape of the maze, as well as any obstacles or hazards to avoid.
- Current state: describe the current state of the robotics system. For example, if the task is to pick up an object,

Imagine we are working with a household robot. The job of this robot is to make an omelette. The objects available around are:
fridge, bowl, pan, oil, stove
The main functions you can use are:
`locate_object(object_name)`: Returns the XYZ coordinates of an object of interest.
`go_to_location(object_name)`: Moves robot to a location specified by XYZ coordinates. Returns nothing.
`pick_up(object_name)`: Picks up the object of interest. Returns nothing.
`use_item(object_name)`: Takes the name of an object as input. Performs an appropriate action on the object at hand (pour oil, crack eggs). Returns nothing.
Can you make use of these to write code to go to the kitchen and make an omelette?

Sure, here is an example Python code that utilizes the provided functions to prepare an omelette for an elderly person:

```
fridge_location = locate_object("fridge")
bowl_location = locate_object("bowl")
pan_location = locate_object("pan")
stove_location = locate_object("stove")
oil_location = locate_object("oil")

# Start the omelette-making process
go_to_location(fridge_location)
pick_up("eggs")
go_to_location(bowl_location)
use_item("eggs")  # Crack eggs into the bowl
go_to_location(pan_location)
pick_up("pan")
use_item("pan")  # Place the pan on the stove
go_to_location("oil")
pick_up("oil")
go_to_location(pan_location)
use_item("oil")  # Pour oil into the pan
go_to_location("bowl")
pick_up("bowl")
go_to_location(pan_location)
use_item("bowl")  # Pour the egg mixture into the
    pan
go_to_location(stove_location)
use_item("stove")  # Turn on the stove
print("Enjoy your omelette!")
```

FIGURE 3. Example of a good prompting practice when defining different APIs for a robotics problem. Clear function definitions that can be executed by the robot, and the task objectives are explicitly stated.

you might describe the current position and orientation of the robot and the object.

- Goals and objectives: state the goals and objectives of the task. If the task is to assemble a puzzle, you might specify the number of pieces that need to be assembled and the desired completion time.
- Solution examples: demonstrate how similar tasks can be solved as a means to guide the LLM's solution strategy. For example, if a task involves interactions with the user, we can describe an example of how and when the robot should be asking for the user's input (see Figure 5). Note that priming can also introduce biases, so we should provide a diverse range of examples and avoid overly prescriptive language.

Even a well designed prompt might not contain all necessary information needed to solve the problem, or in some cases ChatGPT is not able to generate the correct response in a *zero-shot* fashion. In these cases, we find that a simple and effective strategy a user can take is to send additional instructions to ChatGPT in the chat format describing the issue, and have it correct itself. Previous approaches that rely on GPT-3 or Codex models [19], [20] require the user to re-engineer the input prompt and

Imagine we are working with a household robot. The job of this robot is to make an omelette. The objects available around are:
fridge, bowl, pan, oil, stove
The main functions you can use are:
`locate_object(object_name)`: Returns the XYZ coordinates of an object of interest.
`go_to_location(object_name)`: Moves robot to a location specified by XYZ coordinates. Returns nothing.
`pick_up(object_name)`: Picks up the object of interest. Returns nothing.
`use_item(object_name)`: Takes the name of an object as input. Performs an appropriate action on the object at hand (pour oil, crack eggs). Returns nothing.
Can you make use of these to write code to go to the kitchen and make an omelette? Try to use these low level functions to construct more high level behaviors and then solve the task using those high level primitives.

Sure, here is an example Python code that utilizes the provided functions to prepare an omelette for an elderly person:

```
def get_ingredients():
    go_to_location(locate_object("fridge")) # Go
     to the fridge and get the eggs.
    pick_up("eggs")
    go_to_location(locate_object("bowl")) # Go to
     the counter and get the bowl.
    pick_up("bowl")
    use_item("eggs") # Crack the eggs into the
     bowl.
def heat_pan():
    go_to_location(locate_object("pan"))
    pick_up("pan")
    use_item("pan")  # Put the pan on the stove.
    go_to_location(locate_object("stove"))
    use_item("stove")  # Turn on the stove.
```

(rest of the response omitted for brevity)

FIGURE 4. Example of new API development using LLM intrinsic reasoning capabilities. The model improves its answer in comparison to Figure 3, which contained the same task but with a fixed set of APIs.

Imagine you are helping me interact with a mobile ground robot. You are in a room that contains several objects: three chairs, a glass table, a mirror, and two couches. All of your outputs need to be identified by one of the following tags:
<question> Always ask me a clarification questions if you are unsure </question>
<reason> Explain why you did something the way you did it </reason>
<command> Output code command that achieves the desired goal </command>
For example:
Me: Go to the chair
You: <question> There are three chairs in the world, which one do you want me to go to? </question>
Me: Go to "chair_3"
You: <command> target_location = `get_location`("chair_3"); `move_to_location`(target_location); </command> <reason> I found the location of the chair with the get_location function, and moved the robot towards the desired location </reason>
Are you ready?

Yes, I am ready. Please give me your first command.

Go to the couch

<question> There are two couches in the room. Which couch do you want me to go to? </question>

Go to the second sofa

<command> target_location = get_location("second_sofa"); move_to_location(target_location); </command> <reason> I used the get_location function to find the location of the second sofa and then moved the robot towards that location. </reason>

Now move to the other one

<command> target_location = get_location("first_sofa"); move_to_location(target_location); </command> <reason> I used the get_location function to find the location of the first sofa and then moved the robot towards that location. </reason>

FIGURE 5. Example of how user prompting can bias the model's response.

generate new outputs from scratch. The dialog ability of ChatGPT, however, is a surprisingly effective vehicle for behavior correction. Section III-B and the supplementary videos show examples of interactive behavior between the user and ChatGPT.

C. SPECIAL ARGUMENTS TO BIAS THE ANSWER'S STRUCTURE

Different prompting methodologies can be used to force the output of the model to obey specific patterns. For instance, a user might want to automatically parse ChatGPT's output and extract code which can be used as part of other scripts for real-time execution. As shown in Figure 3, a simple strategy is to directly ask ChatGPT to produce code in a specific language (*e.g. Python, C++*). What usually follows is free-form text followed by a code block.

A more structured response can be produced by requesting the model to use XML tags to help us parse the output automatically, as shown in Figure 5.

In other settings users might want to force the model to output answers following a list pattern as opposed to code or free-form text. Appendix A-C displays an example of such approach, where the last line of the user's prompt directs the model's output.

III. CHATGPT ABILITIES FOR SOLVING ROBOTICS PROBLEMS

In this section, we take a deeper look into ChatGPT's abilities for solving robotics problems. Specifically, we investigate how well ChatGPT performs when dealing with a variety of robotics-related tasks, from simple spatio-temporal reasoning problems all the way to real-world aerial agent and manipulation deployments. We highlight several interesting capabilities we observed during these experiments.

While ChatGPT's capabilities are impressive, practical deployment safety considerations should not be overlooked, especially in the case of physical robotics deployments. As shown in Figure 2, we find it necessary to have a human on the loop to monitor and intervene in case of unexpected behaviors generated by ChatGPT. Additionally, the use of simulators can be particularly helpful to evaluate the model's performance before deployment in the real world. We emphasize that the use of ChatGPT for robotics is not a fully automated process, but rather acts as a tool to augment human capacity. We provide all initial prompts for the problems in this section in Appendix B. For the sake of brevity we only include brief excerpts from each task. The full conversations can be found in the following repository: https://github.com/microsoft/PromptCraft-Robotics.

A. ZERO-SHOT TASK PLANNING

First, we note that ChatGPT is able to solve several robotics tasks in a zero-shot fashion, with access only to the prompt and the description of a function library, and no requirement of actual code examples.

[Initial model prompt found in Appendix B-A]

Write a python script that executes a visual servoing approach towards catching a basketball in a court. You can use opencv functions to detect the ball as an orange blob.

Here's a sample implementation of the script:

```
import cv2
import numpy as np

def detect_ball(frame):
# Convert the image to HSV color space for better
    color detection
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
# Define the range of orange color in HSV
lower_orange = np.array([10,100,100])
upper_orange = np.array([30,255,255])
# Threshold the image to get only the orange color
mask = cv2.inRange(hsv, lower_orange, upper_orange
    )
# Find the contours of the thresholded image
contours, _ = cv2.findContours(mask, cv2.
    RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
# If a contour is found, return its center
if len(contours) > 0:
c = max(contours, key=cv2.contourArea)
M = cv2.moments(c)
if M["m00"] > 0:
center = (int(M["m10"] / M["m00"]), int(M["m01"] /
     M["m00"]))
return center
return None

def catch_ball():
# Continuously get images from the camera
while True:
frame = get_image()
# Detect the ball in the image
ball_center = detect_ball(frame)
# If the ball is detected, move towards its
    location
if ball_center is not None:
x, y = ball_center
robot_x, robot_y = get_location()
vx = (x - robot_x) * 0.5
vy = (y - robot_y) * 0.5
move_by_velocity(vx, vy)

if __name__ == '__main__':
catch_ball()
```

This implementation uses a simple visual servoing approach where the robot moves towards the ball at a constant velocity. The velocity is proportional to the difference between the ball's and robot's locations, so the robot will move faster as it gets closer to the ball. You can adjust the gain (i.e., the multiplication factor of 0.5) to control the speed of the robot.

Generate SVG code that depicts how the ball would look in the camera image exactly in the center

Here's an example of an SVG code that depicts a ball in the center of a camera image:

```
<svg width="100" height="100">
<rect x="0" y="0" width="100" height="100" fill="
    lightgray"/>
<circle cx="50" cy="50" r="25" fill="orange"/>
</svg>
```

[*SVG image rendered directly from the code generated by ChatGPT*]

1) SPATIO-TEMPORAL REASONING: CATCHING A BASKETBALL WITH VISUAL SERVOING

In this example, we ask ChatGPT to control a planar robot equipped with an upward-facing camera. The robot is expected to catch a basketball using a visual servoing method based on the appearance of a basketball. We see that ChatGPT is able to appropriately use the provided API functions, reason about the ball's appearance and call relevant OpenCV functions, and command the robot's velocity based

on a proportional controller. Even more impressive is the fact that ChatGPT can estimate the appearance of the ball and the sky in the camera image using SVG code. This behavior hints at a possibility that the LLM keeps track of an implicit world model going beyond text-based probabilities.

2) AERIAL ROBOTICS: REAL-WORLD DRONE FLIGHT WITH INTUITIVE HUMAN-ROBOT INTERFACE (CLICK FOR VIDEO)

We also used ChatGPT with a different API implementation to control a real drone. ChatGPT provided an extremely intuitive natural language interface between the user and the robot, writing robot code based on ambiguous and ill-defined instructions, and asking clarification questions when necessary. The model was also able to write complex code structures for drone navigation (circular and lawnmower inspection) based solely on the prompt's base APIs.

[Initial model prompt found in Appendix B-B]

I am thirsty. Find me a drink

Question: What type of drink are you looking for? There are several options in the scene, including "coconut water" and "diet coke can".

I just came back from the gym and am looking for a healthy option.

In that case, you might like "coconut water". Here is the code to fly to the "coconut water" object:

```
drink_position = self.dict_of_objects["coconut
    water"]
self.tello.fly_to(drink_position)
```

Explanation: This code first retrieves the X, Y, Z, and angle position of the "coconut water" object from the dict_of_objects dictionary. The fly_to method is then used to fly the drone to the retrieved position. The drone will fly to the position of the "coconut water" object and look in the direction of the angle.

I am still thirsty. Can you find something with sugar and a red logo?

I'm assuming you are referring to a can of "regular coke". Here is the code to fly to the "regular coke can" object:

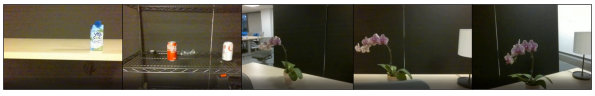
```
drink_position = self.dict_of_objects["regular
    coke can"]
self.tello.fly_to(drink_position)
```

Fantastic. Now I want to take a look at my orchid. However, the chair might be blocking its view, so I want to inspect it from 3 points on a half-circle from -60deg to -120, 1m away. Make sure that the drone's heading points towards the orchid from each point.

Here's the code to inspect the orchid from 3 points on a half-circle:

```
import numpy as np
# Get the position of the orchid
orchid_position = self.dict_of_objects['orchid']
# Define the angle range for the half-circle
angles = np.linspace(-60, -120, num=3, endpoint=
    False) * np.pi / 180
# Define the distance from the orchid to inspect
    from
distance = 1.0
# Define the inspection points
inspection_points = [orchid_position[:3] + [
    distance * np.cos(angle), distance * np.sin(
    angle), 0] for angle in angles]
# Fly the drone to each inspection point and look
    at the orchid
for point in inspection_points:
self.tello.fly_to(point + [np.arctan2(
    orchid_position[1] - point[1],
    orchid_position[0] - point[0]),
    orchid_position[3]])

\textrm{(text-based explanation of the response
    omitted for brevity)}
```



[*snapshots from drone video*]

3) AERIAL ROBOTICS: AIRSIM INDUSTRIAL INSPECTION (CLICK FOR VIDEO)

We also applied ChatGPT to a simulated domain, using the Microsoft AirSim [23] simulator. We explored the idea of a potentially non-technical user directing the model to control a drone and execute an industrial inspection scenario. We observe from the following excerpt that ChatGPT is able to effectively parse intent and geometrical cues from user input and control the drone accurately.

[Initial model prompt found in Appendix B-C]

Fly to the turbine.

Question - There are two turbines in the world, which one do you want me to fly the drone to? Turbine 1 or Turbine 2?

Turbine 2, please. Fly to the turbine while keeping 12 meters away from the turbine along the X axis, and reach an altitude of 50 meters.

Here's the code to fly the drone to Turbine 2 while maintaining a distance of 12 meters along the X-axis and reaching an altitude of 50 meters:

```
# Get the position of Turbine 2
turbine_position = get_position(''Turbine 2'')
# Set the target position to be 12~meters away
    from the turbine along the X\text{-axis} and
    50~meters high
target_position = [turbine_position[0] - 12,
    turbine_position[1], 50]
# Fly the drone to the target position
fly_to(target_position)
```

To inspect the turbine blade, I need you to fly in the bottom-right direction at an angle of -30 degrees from the horizontal axis. You should fly in the YZ plane for a distance of 30 meters, and then back to the current position.

Here's the code to fly the drone in the bottom-right direction at an angle of -30 degrees from the horizontal axis, in the YZ plane for a distance of 30 meters, and then back to the current position:

```
import math
# Get the drone's current position
current_position = get_position(''Drone'')
# Calculate the target position, which is 30~
    meters away in the YZ plane at an angle of
    -30 degrees from the horizontal axis
angle = math.radians(-30)
target_position = [
current_position[0],
current_position[1] + 30 * math.cos(angle),
current_position[2] + 30 * math.sin(angle)
]
# Fly the drone to the target position
fly_to(target_position)
# Fly the drone back to the current position
fly_to(current_position)
```

[*Snapshots from inspection video*]

B. USER ON THE LOOP: INTERACTIVE CONVERSATIONS FOR COMPLEX TASKS

As a next step, we focused on the ability of ChatGPT to perform more complex tasks with a user on the loop offering textual feedback. We observed that this modality of interaction allows for complex constructs such as curriculum learning, where the system is taught smaller-scale skills which can be combined towards larger and more intricate tasks. In addition, we found that ChatGPT is able to take high-level textual feedback regarding generated code or its performance and map it to the required low-level code changes, thus enabling potentially non-technical users to interact with it with ease.

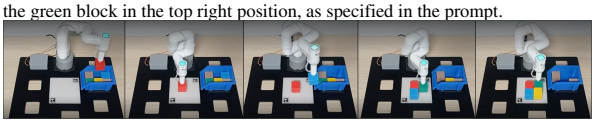
[Initial model prompt found in Appendix B-G]
[The beginning of the conversation is omitted for brevity. The following excerpt requires ChatGPT to build a new function using previously defined APIs, and later use this function for more complex tasks.]
I want you to learn how to place an object in a position. Considering you are holding a generic object, you first need to first move to a safe position above the target location, bring the object down and then release it.
Understood. Here's a Python function that takes the position where the object should be placed, moves the robot arm above the position, brings the object down, and releases it. Note that the function assumes that the robot arm is holding an object.

```
def place_object(position):
# move the robot arm to a safe position above the
    target location
safe_pos = [position[0], position[1], position
    [2]+100, position[3], position[4], position
    [5]]
move_to(safe_pos)
# move the robot arm to the target location
move_to(position)
# release the object
release()
# move the robot arm back to the safe position
    above the target location
move_to(safe_pos)
```

What does the Microsoft logo look like?
The Microsoft logo is a colored rectangular with four squares arranged in a grid pattern. The colors used in the logo are blue, green, orange, and yellow. The logo has gone through several iterations over the years, but this is the most recent and recognizable design.
Can you generate an SVG with those colors?
Yes, I can generate an SVG file with the Microsoft logo colors. Here is an example

```
<svg width="100" height="100" viewBox="0 0 100 100
    " xmlns="http://www.w3.org/2000/svg">
<rect x="0" y="0" width="50" height="50" fill="#
    F25022"/>
<rect x="50" y="0" width="50" height="50" fill="#7
    FBA00"/>
<rect x="0" y="50" width="50" height="50" fill="
    #00A4EF"/>
<rect x="50" y="50" width="50" height="50" fill="#
    FFB900"/>
</svg>
```


[*Reproduction of SVG generated by ChatGPT*]
Awesome! I want you now to use the blocks to build the logo from Microsoft on top of the white pad.
Sure! Here's the Python code to build the Microsoft logo on top of the white pad using the blocks:
[Code not shown for brevity. Please consult the link for the full conversation]
This code gets the positions of the white pad and all four blocks using the get_position function, picks up each block in turn using the pick_up_object function we defined earlier, calculates the position to place the block on the white pad, and places the block on the white pad using the place_object function we defined earlier. We use the block colors to ensure that the logo is built correctly. We place the blue block in the bottom left position, the yellow block in the bottom right position, the red block in the top left position, and the green block in the top right position, as specified in the prompt.


[*snapshots from manipulator video*]

[Initial model prompt found in Appendix B-D]
Assume I give you some XYZ coordinates as a goal. I want you to control the drone such that it moves towards the goal while avoiding obstacles in front of it. Instead of flying directly to the goal, the drone should take small steps while reassessing the obstacle situation at every step.
The drone is equipped with a distance sensor that returns the distance to the closest obstacle directly in front of the drone. You can access it by calling `get_distance()`.
I want you to make sure to fly only if there is at least 10 meters of distance in front of the drone. If not, you should rotate the drone until there is at least 10 meters of clearance in the front and then take a step in the direction that the drone is currently facing. Don't forget to face the goal again after taking a step. Got it?
Yes, I got it. Here's how I would approach the problem.

```
# Get the drone's current position and set the yaw
    angle towards the goal
current_pos = get_position('drone')
goal_pos = [x_goal, y_goal, z_goal]
yaw = math.atan2(goal_pos[1]-current_pos[1],
    goal_pos[0]-current_pos[0]) * 180 / math.pi
set_yaw(yaw)
# Start a loop that runs until the drone reaches
    the goal position
while np.linalg.norm(np.array(current_pos) - np.
    array(goal_pos)) > 0.5:
# Check if there is at least 10~meters of distance
    in front of the drone
distance = get_distance()
while distance < 10:
# Set the yaw angle to rotate the drone until
    there is at least 10~meters of clearance in
    the front
set_yaw(yaw+30) # rotate 30~degrees at a time
time.sleep(1) # wait for a second to measure the
    new distance
distance = get_distance()
# Take a step towards the goal while maintaining
    the yaw angle
step_size = 5 # take steps of 5~meters
step_vector = [step_size * math.cos(yaw),
    step_size * math.sin(yaw), 0]
new_pos = [current_pos[i]+step_vector[i] for i in
    range(3)]
fly_to(new_pos)
current_pos = get_position('drone')
```

After you take a step, you need to face the goal again.
You're right. After taking a step, we need to recalculate the yaw angle to face the goal again. Here's the modified code that includes this step:

```
'''Same code as before'''
yaw = math.atan2(goal_pos[1]-current_pos[1],
    goal_pos[0]-current_pos[0]) * 180 / math.pi
set_yaw(yaw)
```

This code should ensure that the drone faces the goal after taking a step, and continues moving towards the goal while avoiding obstacles in front of it.
You are not using the current yaw angle when computing the step vector. Remember there is a get_yaw() function.
You're right. After taking a step, we need to recalculate the yaw angle to face the goal again. Here's the modified code that includes this step:

```
# Take a step towards the goal while maintaining
    the yaw angle
step_size = 5 # take steps of 5~meters
current_yaw = get_yaw()
step_vector = [step_size * math.cos(current_yaw),
    step_size * math.sin(current_yaw), 0]
```

1) MANIPULATION WITH CURRICULUM LEARNING (CLICK FOR VIDEO)

As a first example, we set up a real-life block arrangement task using a robot arm, for which we devised a curriculum. ChatGPT was asked to learn the simpler skills of picking up objects, and then placing them. Subsequently, when asked to use these skills to perform a more involved task of block arrangement, ChatGPT chains these learned skills together logically. In addition, the model displays a fascinating example of generalizability when bridging the textual and physical domains when tasked with building the Microsoft logo out of wooden blocks. This task requires memory of how the corporate logo looks like, including its colors, followed by building an abstraction of the logo into physical parts which are constructible by the existing robot actions. The code output by ChatGPT was executed on a computer that controlled the robot arm, and the robot successfully completed the task.

2) AERIAL ROBOTICS: AIRSIM OBSTACLE AVOIDANCE (CLICK FOR VIDEO)

We tasked ChatGPT to write a goal-reaching algorithm with obstacle avoidance for a drone in the AirSim simulator equipped with a forward facing distance sensor. ChatGPT built most of the key building blocks for the avoidance algorithm, but required some human feedback on steps it missed regarding the drone's orientation. Although the feedback was provided entirely in high-level text, ChatGPT improved its solution with localized changes to the code where appropriate. We deployed the final solution on a simulated drone in AirSim, and observed that it was able to successfully navigate to the goal while avoiding obstacles.

C. PERCEPTION-ACTION LOOPS

We also evaluate ChatGPT's ability to reason about perception-action loops. At a first level, we outline the model's ability to make use of the API library to construct perception-action loops in its code output. The model correctly employs perception functions such as image acquisition and object detection to extract the relevant information for robot navigation and controls.

At a second level of complexity, we try to answer the question of whether ChatGPT's dialogue system can serve as a closed feedback perception-action loop in itself. We explore the idea of continuously feeding the model with perception information via textual dialog, where we input in observations (converted into a textual format) to ChatGPT during a conversation. We find that ChatGPT is able to parse this stream of observations and output relevant actions. The latency of the model is a limiting factor in this mode of operation (we have observed 1-2 seconds to parse the observation and return the appropriate action), but we believe that this is a promising direction for future research, especially as inference speeds of LLMs continue to improve.

1) EMBODIED AGENT: CLOSED LOOP OBJECT NAVIGATION WITH API LIBRARY (CLICK FOR VIDEO)

We provided ChatGPT access to a computer vision model as part of its function library, and tasked it to explore an unknown environment and navigate to a user-specified object inside the AirSim simulator. The object detection API (YOLOv8 [24] in the back-end) returned bounding boxes, and ChatGPT generated the code to estimate relative object angles and navigate towards them. When we offered ChatGPT additional information from a depth sensor it produced an improved algorithm with pixel depth masking, expressed as a modular pipeline.

2) EMBODIED AGENT: CLOSED LOOP VISUAL-LANGUAGE NAVIGATION USING CHATGPT'S DIALOGUE (CLICK FOR VIDEO)

In experimental character, we also evaluated ChatGPT's ability to act as a perception-action control feedback loop by leveraging the model's ability to hold a dialogue. In this mode of operation new state observations are fed into the system as dialogue text, and the model's outputs are executed by the robot at each step, generating new observations. We note that

[Initial model prompt found in Appendix B-F]

I need you to help me find an object. Objects might be scattered around the scene, so if you don't find it right away, a good strategy is to turn around a few times to see if it becomes visible. Give me code to explore and find the bottle.

Got it. Here's some code that will help you explore the environment and find the bottle:

```
def search_for_bottle():
num_turns = 4 # Turn around 4~times to look for the bottle
for i in range(num_turns): # Turn left 90~degrees to face
      a new direction
turn_left()
img = get_image()
obj_list, obj_locs = detect_objects(img)
if 'bottle' in obj_list:
# Get the coordinates of the bottle in the image
bottle_idx = obj_list.index('bottle')
bbox = obj_locs[bottle_idx]

# Compute the center of the bounding box
center_x = (bbox[0] + bbox[2]) / 2
center_y = (bbox[1] + bbox[3]) / 2

# Print the center coordinates of the bottle
print(f"Found the bottle at ({center_x}, {center_y})")
```

Great! Now let's say we did find the bottle. Now I want to move towards it. The only information we have is the location of the bottle bounding box in the image. Can you give me code to make this happen?

…(model comments omitted for brevity)…

```
'''prior code omitted for brevity'''
center_x = (bbox[0] + bbox[2]) / 2
center_y = (bbox[1] + bbox[3]) / 2
# Compute the distance and angle to the center of the
      bounding box
img_width, img_height = img.size
angle = math.atan2(center_x - img_width / 2, img_height /
      2 - center_y)
distance = (center_x - img_width / 2) / (img_width / 2)
if abs(distance) > 0.05:
if distance > 0:
# Bottle is to the right, so~turn right and move forward
turn_right()
forward()
else:
# Bottle is to the left, so~turn left and move forward
turn_left()
forward()
'''later code omitted for brevity'''
```

I want you to modularize this code a bit. Why don't you try coming up with a few high level functions and use them?

Good idea! Here's a modularized version of the code that uses some helper functions:

```
def get_object_distance_and_angle(img, depth_img, obj_name
      ):
#Code omitted for brevity
def turn_towards_angle(angle):
#Code omitted for brevity
def search_for_object_and_move_to_it(obj_name):
#Code omitted for brevity
```

Now, let's assume you have access to all these functions you just created. Let's get to work solving some tasks. Find out where I can warm up my lunch.

Sure, to find out where you can warm up your lunch, we can use the `search_for_object_and_move_to_it` function to search for a microwave in the scene. Here's an example code snippet:

```
# Search for a microwave
search_for_object_and_move_to_it(''microwave'')
```

This code will keep exploring the environment until a microwave is found. Once a microwave is found, the agent will move towards it and stop when it gets within 30 cm of it.

[*snapshots from embodied object navigation*]

this type of behavior is heavily dependent on a dialog-based LLM such as ChatGPT, which can naturally take receive a stream of observations. We constructed a scenario using the Habitat simulator [25] and tasked the agent to navigate to an area of interest. At each step we provided ChatGPT with a description of the scene in terms of polar coordinates of the visible objects relative to the agent. We constrained the output of ChatGPT to only return a forward distance of motion and the turning angle. We verify that the model is able to complete simple navigation tasks, but more research is needed in terms of properly describing the scene context in textual or vector form for more complex tasks and environments.

For additional experiments, we refer the reader to Appendix C.

[Initial model prompt found in Appendix B-E]
Task: go to the chairs Objects in the scene: <door, 0.53 m, 22>, <chair, 4.84 m, -21>, <chair, 2.12 m, -24>, <window, 6.74 m, -3>, <window, 7.17 m, 6>, <door, 0.61 m, 18>, <stairs, 1.42 m, 0> Command:
move(1.5) # Move forward by 1.5 meters to explore the environment.
Objects in the scene: <seating, 5.05 m, 21>, <chair, 3.93 m, -25>, <window, 4.55 m, 22>, <window, 5.24 m, -4>, <stairs, 3.49 m, 13>, <window, 5.67 m, 7>, <window, 4.37 m, 23> Command:
turn(-25) # Turn to the left to face the chair that is 3.93 meters away
Objects in the scene: <chair, 4.85 m, -14>, <chair, 4.76 m, -9>, <chair, 1.57 m, -23>, <chair, 2.66 m, -24>, <window, 4.96 m, 10>, <window, 4.19 m, 25> Command:
turn(-10) # Turn to the right to explore the environment further.
Objects in the scene: <door, 5.07 m, -25>, <chair, 5.36 m, -6>, <table, 2.17 m, -24>, <chair, 5.01 m, -1>, <chair, 1.80 m, -16>, <chair, 3.15 m, -16>, <window, 4.30 m, 19> Command:
turn(-6) # Turn to the right to face the chair that is 5.36 meters away.
Objects in the scene: <counter, 1.24 m, -24>, <door, 5.93 m, -17>, <chair, 5.48 m, 1>, <table, 2.60 m, -14>, <chair, 4.92 m, 6>, <chair, 2.02 m, -7>, <chair, 3.53 m, -7>, <chair, 3.16 m, -18> Command:
turn(1) # Turn to the left to explore the environment further.
Objects in the scene: <door, 5.07 m, -25>, <chair, 5.36 m, -6>, <table, 2.17 m, -24>, <chair, 5.01 m, -1>, <chair, 1.80 m, -16>, <chair, 3.15 m, -16>, <window, 4.30 m, 19> Command:
move(5.5) # Move forward towards the chair that is 5.36 meters away.
Objects in the scene: <chair, 0.96 m, -10>, <chair, 0.99 m, 20>, <chair, 1.77 m, -20> Command:
move(1) # Move forward to approach the chairs.

[*snapshots from Habitat navigation*]

D. REASONING AND COMMON-SENSE ROBOTICS TASKS

We tasked ChatGPT with answering simple logical reasoning questions, as well as basic robotics problems, shown in Appendix D. We observe that ChatGPT offers a rich foundation of common-sense and logical reasoning capabilities upon which more advanced robotics abilities can be built. This logics grounding allows the user to interact with the model more naturally, as opposed to having to prompt or define every concept from scratch. Furthermore, ChatGPT's out-of-the-box understanding of basic robotics concepts such as control, camera geometry, and physical form factors makes it an excellent choice to build upon for generalizable and user-friendly robotics pipelines.

E. USAGE OF CHATGPT API IN REAL-TIME

Most of the prior examples we presented in this section demonstrated open loop behavior where ChatGPT generated code to solve a task while conversing with a user, prior to the actual deployment. As an extension, we have also examined the usage of ChatGPT in real-time robotics applications, where the model generates code for robotics problems in real-time. In order to achieve this, we use the AirSim simulator, which allows us to safely test the model's generated code in a controlled environment. We utilize the OpenAI ChatCompletion Python package to interact with the model in real-time, and execute the generated code in a Python kernel. In order to use the API in real time, we modified the system prompt for the ChatGPT model to command it to always return properly formatted Python code that is easily identifiable through the markdown format. In each response returned by the API, we use regex-based matching to extract the Python code if any exists, and then executed that in a Python kernel. In order to ensure safety of the drone even in the simulation, the drone was made to hover in place while a user places a request to the model, and until the model's response was executed.

We performed some experiments within the AirSim simulator, where we observed that the model was able to generate code for a drone to navigate to specified goals such as object goal navigation, inspection, etc. The results of this experiment can be seen at https://www.youtube.com/watch?v=iE5tZ6_ZYE8. We note that the latency of the model was not entirely suitable for high speed real-time applications, but the ability of the drone to hover in place for a few seconds allowed us to use the LLM in a safe manner. Similarly, a simulation is an ideal environment for such an experiment, because even if the output of ChatGPT were wrong or inaccurate, the simulation can be reset to a previous safe state. If there is an intent to perform such an experiment in real life, users would first need to create a safety layer that can evaluate the correctness of the generated code before directly executing it, or have other appropriate safety rules in place for the robot (e.g. hardcoded constraints on speed, direction of motion, etc.). We expect future research to explore how to properly use LLMs in real-time robotics applications, and how to ensure the safety and correctness of the generated code.

IV. PROMPTCRAFT, A COLLABORATIVE TOOL FOR LLM + ROBOTICS RESEARCH

Prompting is a crucial component to generate the desired behaviors in large language models (LLMs). Prompt engineering is particularly challenging at the intersection of LLMs with robotics, where there is a lack of comprehensive and accessible resources that provide examples of positive (and negative) interactions. To address this gap, we introduce *PromptCraft*,[1] a collaborative open-source platform for

[1] https://github.com/microsoft/PromptCraft-Robotics

researchers to share examples of prompting strategies and test their algorithms in sample robotic environments.

PromptCraft is a Github-based platform that allows researchers to share examples of prompt engineering strategies within different robotics categories, such as navigation, grasping, and manipulation. Users can submit their examples and rate others' submissions, which we hope will create a community-driven resource for researchers working with LLMs. Submissions of prompts and dialogues are primarily based on text, but we encourage users to share videos and images depicting the robot's behavior, especially for real-world deployment scenarios.

In addition to providing a platform for sharing prompt examples, PromptCraft also offers an AirSim [23] environment with a ChatGPT wrapper for researchers to prototype prompts and algorithms in a controlled simulated setting. We welcome contributions of new test environments to expand the range of scenarios where researchers can test their algorithms.

With Promptcraft we aim to support the empirical science of prompt engineering and enable researchers to advance the field.

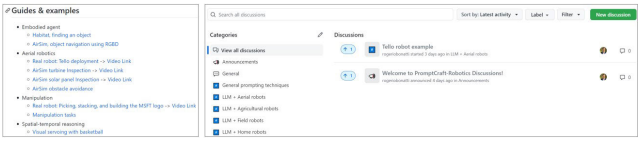


FIGURE 6. Promptcraft open-sourced repository. Researchers can upload and vote on examples of LLM prompts for various robotics categories.

V. RELATED WORK

A. NATURAL LANGUAGE AND ROBOTICS

Natural language processing (NLP) has long been recognized as a crucial component for human-robot interaction. There are many applications where robots can benefit from NLP, including but not limited to task instruction, navigation, and information retrieval. Classically, modeling human-robot interactions using language is challenging because it forces the user to operate within a rigid set of instructions [26], or requires mathematically complex algorithms to keep track of multiple probability distributions over actions and target objects [27], [28]. More recent works explore neural networks to implicitly keep track of the complex mapping between language and actions, but such techniques often require vast amounts of labeled data for training [9], [10], [29], [30]

B. LARGE (VISION AND) LANGUAGE MODELS FOR ROBOTICS

The Transformer architecture, introduced in the paper by [31], has revolutionized NLP and has also shown great promise in robotics. Transformers have been used for robot control and planning [32], [33], [34], object recognition [35], and robot navigation [36]. A more common use of transformers in robotics has been as feature extraction modules for one or more modalities simultaneously. These systems are often coupled with additional features from pretrained large-scale vision and language models models [11], [14], [15], [16], [37], [38], [39].

Models such as SayCan [38] focus on grounding LLMs so that free-form text commands are used to compute a value function to rank the best action types within a robot-specific library. RT-1 [40], on the other hand, takes an end-to-end approach to learn the mapping between language commands low level actions, without the use of intermediate high-level functions. Recent works have also explored the ability of large language models (LLMs) for zero-shot high-level robotics task planning [17], [19], [20]. These models make use of prompting structures with predefined functions, behaviors, and examples to guide the generation of the model's answers. Reference [18] also explore the use of interactivity between user and LLM for table-top manipulation settings. Another interesting approach was outlined in Socratic Models [41], which shows that the individual shortcomings of VLMs and LLMs can be alleviated through a combination of several models, and by allowing them all to communicate to the LLM the common modality of text. In our work, we discuss a possible similarity where an object detection model and the LLM can be integrated for vision based navigation. We find the ChatGPT paradigm to be a potentially more flexible and generalizable approach as it was shown to be effective for a wide range of form factors, can integrate multiple other models through code, and can naturally hold a dialogue with the user for refinement and improvement of the policies.

Conceptually, the main difference of these approaches with respect to our work, which leverages ChatGPT [1], is the conversational ability of our LLM, which allows the user to interactively improve and correct the robot's behavior (as opposed to re-engineering the prompt from scratch and generating another zero-shot answer). In addition, our works aims to provide a generalizable pipeline and set of principles to be used by researchers in different fields of robotics, as opposed to focusing on a single domain such as table-top manipulation or task planning.

C. PROMPTING LLMS WITH APIS, AND ITS CONNECTIONS TO SYMBOLIC AI

When designing LLM prompts for robotics applications, users often make use of high-level library of APIs to represent specific behaviors to be used. We can draw a connection between this approach with classical symbolic AI, which uses logic and rules to represent and reason about knowledge [42]. While the traditional symbolic AI approach presented difficulties in new knowledge acquisition and dealing with out-of-distribution data, we believe that LLMs can overcome these challenges. As we showed in Section II-A and Section III, models such as ChatGPT can compose new primitive functions based on the context and generate code for them automatically.

VI. CONCLUSION AND FUTURE WORK

We presented a framework for using ChatGPT for robotics applications. The framework entails designing and implementing a library of APIs that for robot control which are amenable to prompt engineering for ChatGPT. We discussed design principles for creating such APIs and prompting strategies that can be used to generate code for robotics applications via ChatGPT. The proposed framework allows the generated code to be tested, verified, and validated by a user on the loop via a range of methods including simulation and manual inspection. We demonstrated how the framework can be used for multiple applications ranging from simple common-sense robotics knowledge tasks all the way to deployments in aerial robotics, manipulation and visual navigation.

We believe that this work presents only a small fraction of what is possible within the intersection of large language models operating in the robotics space. In this work, we examine the idea of using ChatGPT during the development process of robotics algorithms, but a future direction is to explore how to use LLMs directly in the deployment setting. We hope to not only inspire other researchers to take these next steps, but to also help them achieve results with the use of the PromptCraft collaborative tool.

Most of the examples we presented in this work demonstrated open perception-action loops where ChatGPT generated code to solve a task, with no feedback was provided to the model afterwards. Given the importance of closed-loop controls in perception-action loops, we expect much of the future research in this space to explore how to properly use ChatGPT's abilities to receive task feedback in the form of textual or special-purpose modalities. This could be achieved either by using cloud-based models such as ChatGPT within a closed loop, or by deploying open source LLMs and leveraging optimization techniques such as quantization for efficiency [43], [44].

We emphasize that these tools should not be given full control of the robotics pipeline, especially for safety-critical applications. Given the propensity of LLMs to eventually generate incorrect responses, it is fairly important to ensure solution quality and safety of the code with human supervision before executing it on the robot. We expect several research works to follow with the proper methodologies to properly design, build and create testing, validation and verification pipelines for LLM operating in the robotics space.

A. USAGE OF LLMS WITHIN DEPLOYMENT PIPELINES

In this paper, we discuss two main ways to use LLMs in robotics: as a tool for generating code for robotics problems apriori, and as a tool for generating code for robotics problems in real-time. As mentioned earlier, we highly recommend the former approach for safety-critical applications, and the latter as an experimental tool for non-critical applications. LLMs can be used to generate code for robotics problems in real-time, but it is important to ensure that the generated code is safe and correct before executing it on the robot. When evaluating the usage of LLMs in real time, robots and the scenarios should be sufficiently equipped such that issues such as latency are not a concern. For instance, if a drone could hover safely and accurately for a few seconds while waiting for a response from the LLM, then the latency of the LLM may not be a safety concern. In such a case, the LLM acts as a high level task or behavioral planner, whereas the low level controls are handled by the drone or the robot itself, which we believe is a more pragmatic approach. Similarly, it would be highly important to have a layer that could check for the feasibility and correctness of the generated code before executing it on the robot.

APPENDIX A
ADDITIONAL EXAMPLES ON DESIGN PRINCIPLES ON PROMPT ENGINEERING

A. EXAMPLE OF PROMPT WITH NO API LIBRARY: UNBOUNDED TEXT-BASED RESPONSE

Consider you are a home assistant robot. You goal is to prepare an omelette for an elderly person. Output python code with the sequence of steps that achieves your objective.

Sure, here's an example Python code that outlines the sequence of steps to prepare an omelette for an elderly person:

```
print(''Good morning! Would you like an omelette for breakfast?'')
time.sleep(5)
print(''Great! Let's get started.'')
print(f"Cracking {eggs} eggs into a bowl\ldots ")
print("Adding a splash of milk, a~pinch of salt, and a pinch of pepper
    \ldots ")
print("Whisking the ingredients together\ldots ")
print(f"Heating {butter} in a \text{non-stick} pan over medium heat\
    ldots ")
print("Pouring the egg mixture into the pan\ldots ")
print("Cooking the omelette for 2-3 minutes, until the bottom is set\
    ldots ")
print("Using a spatula to flip the omelette\ldots ")
print("Cooking the other side of the omelette for an additional minute
    \ldots ")
print(''Serving the omelette hot and ready to eat. Enjoy your meal!'')
```

B. EXAMPLE OF PROMPT WITH UNDER-SPECIFIED APIS: HALLUCINATION OVER FUNCTION PARAMETERS

Consider you are a home assistant robot. You goal is to prepare an omelette for an elderly person. You are equipped with functions such as `locate_object()`, `move_to_location()`, `cook_item()`, `grab_object()`. Output python code with the sequence of steps that achieves your objective.

Sure, here is a Python code with the sequence of steps that a home assistant robot can take to prepare an omelette for an elderly person:

```
print("Starting to prepare an omelette for the elderly person\ldots ")
locate_object(EGGS_LOCATION)
move_to_location(FRIDGE_LOCATION)
grab_object(EGGS_LOCATION)
locate_object(PAN_LOCATION)
move_to_location(STOVE_LOCATION)
grab_object(PAN_LOCATION)
locate_object(OIL_LOCATION)
grab_object(OIL_LOCATION)
print("Pouring the oil into the pan\ldots ")
time.sleep(2)
print("Cracking the eggs into the pan\ldots ")
grab_object(EGGS_LOCATION)
time.sleep(1)
cook_item(PAN_LOCATION, COOKING_TIME)
print("The omelette is ready! Serving it to the elderly person\ldots "
    )
```

C. EXAMPLE OF PROMPT WITH STRUCTURED RESPONSE FORCING THE ANSWER TO BE IN A LIST FORMAT

Consider you are a home assistant robot. You goal is to prepare an omelette for an elderly person. Find the sequence of steps that achieves your objective.

First, gather all the necessary ingredients and equipment for making an omelette. This would typically include eggs, milk, butter, salt, pepper, a mixing bowl, a whisk, a frying pan, and a spatula.
2. Crack the eggs into the mixing bowl and add a splash of milk, a pinch of salt, and some black pepper to taste.
3. Use the whisk to beat the eggs and milk together until they are well combined.
4. Heat the frying pan over medium heat and add a small amount of butter to the pan, allowing it to melt and coat the bottom of the pan.
5. Pour the egg mixture into the frying pan and use the spatula to spread it evenly over the bottom of the pan.

6. Allow the eggs to cook for a minute or two, until the bottom starts to set and the top is still slightly runny.
7. Use the spatula to fold one half of the omelette over the other half, forming a half-moon shape.
8. Cook for another minute or two until the eggs are fully set and the omelette is heated through.
9. Use the spatula to slide the omelette onto a plate and serve immediately.
10. Make sure to clean up any mess that was made during the cooking process and return all equipment and ingredients to their proper places.

APPENDIX B
ORIGINAL INITIAL PROMPTS FOR ROBOTICS TASKS EXAMPLES

A. SPATIAL-TEMPORAL REASONING: CATCHING A BASKETBALL WITH VISUAL SERVOING

Full conversation found at:
https://github.com/microsoft/PromptCraft-Robotics/blob/main/examples/spatial_temporal_reasoning/visual_servoing_basketball.md
Initial ChatGPT prompt:

Imagine you are a planar robot that can move along the XY axes, and you're positioned in the center of a basketball court. A person on the side of the court is going to throw a basketball ball in the air somewhere in the court, and your objective is to be at the exact XY location of the ball when it lands. The robot has a monocular RGB camera that looks up. You can assume that the following functions are available:
`get_image()`: returns an image from the robot's camera looking up;
`get_location()`: returns 2 floats XY with the robot's current location in the court;
`move_to_point(x,y, vx, vy)`: moves the robot towards a specific (x,y) location in the court with velocity (vx,vy). You can assume for this exercise that the robot can accelerate or break instantly to any velocity;
`move_by_velocity(vx, vy)`: moves the robot along the X axis with velocity vx, and Y axis with velocity vy;
Additional points to consider when giving your answer 1) Your reponses should be informative, visual, logical and actionable, 2) Your logics and reasoning should be rigorous, intelligent, and defensible, 3) You can provide additional relevant details to respond thoroughly and comprehensively to cover multiple aspects in depth.
Write a python script that executes a visual servoing approach towards catching a basketball in a court. You can use opencv functions to detect the ball as an orange blob.

B. AERIAL ROBOTICS: REAL-WORLD DRONE FLIGHT

Full conversation found at:
https://github.com/microsoft/PromptCraft-Robotics/blob/main/examples/aerial_robotics/tello_example.md
Initial ChatGPT prompt:

Imagine you are helping me interact with the AirSim simulator for drones. At any given point of time, you have the following abilities, each identified by a unique tag. You are also required to output code for some of the requests.
Question: You can ask me a clarification question, as long as you specifically identify it saying "Question". Code: Output a code command that achieves the desired goal.
Reason: After you output code, you should provide an explanation why you did what you did.
The simulator contains a drone, along with several objects. Apart from the drone, none of the objects are movable. Within the code, we have the following commands available to us. You are not to use any other hypothetical functions.
`get_position(object_name)`: Takes a string as input indicating the name of an object of interest, and returns a vector of 4 floats indicating its X,Y,Z,Angle coordinates.
`self.tello.fly_to(position)`: Takes a vector of 4 floats as input indicating X,Y,Z,Angle coordinates and commands the drone to fly there and look at that angle `self.tello.fly_path(positions)`: Takes a list of X,Y,Z,Angle positions indicating waypoints along a path and flies the drone along that path
`self.tello.look_at(angle)`: Takes an angle as input indicating the yaw angle the drone should look at, and rotates the drone towards that angle
Here is an example scenario that illustrates how you can ask clarification questions. Let us assume a scene contains two spheres?
Me: Fly to the sphere. You: Question - there are two spheres. Which one do you want me to fly to? Me: Sphere 1, please.
You also have access to a Python dictionary whose keys are object names, and values are the X,Y,Z,Angle coordinates for each object:
self.dict_of_objects = {'origin': [0.0, 0.0, 0.0, 0], 'mirror': [1.25, -0.15, 1.2, 0], 'chair 1': [0.9, 1.15, 1.1, np.pi/2], 'orchid': [0.9, 1.65, 1.1, np.pi/2], 'lamp': [1.6, 0.9, 1.2, np.pi/2], 'baby ducks': [0.1, 0.8, 0.8, np.pi/2], 'sanitizer wipes': [-0.3, 1.75, 0.9, 0], 'coconut water': [-0.6, 0.0, 0.8, -np.pi], 'shelf': [0.95, -0.9, 1.2, np.pi/2], 'diet coke can': [1.0, -0.9, 1.55, np.pi/2], 'regular coke can': [1.3, -0.9, 1.55, np.pi/2]}
Are you ready?

C. AERIAL ROBOTICS: AIRSIM INDUSTRIAL INSPECTION

Full conversation found at:
https://github.com/microsoft/PromptCraft-Robotics/blob/main/examples/aerial_robotics/airsim_turbine_inspection.md
Initial ChatGPT prompt:

Imagine you are helping me interact with the AirSim simulator for drones. At any given point of time, you have the following abilities. You are also required to output code for some of the requests.
Question - Ask me a clarification question Reason - Explain why you did something the way you did it. Code - Output a code command that achieves the desired goal.
The simulator contains a drone, along with several objects. Apart from the drone, none of the objects are movable. Within the code, we have the following commands available to us. You are not to use any other hypothetical functions.
`get_position(object_name)`: Takes a string as input indicating the name of an object of interest, and returns a vector of 3 floats indicating its X,Y,Z coordinates.
`fly_to(position)`: Takes a vector of 3 floats as input indicating X,Y,Z coordinates and commands the drone to fly there.
`fly_path(positions)`: Takes a list of X,Y,Z positions indicating waypoints along a path and flies the drone along that path.
Here is an example scenario that tells you how to respond where we are working with a simulated world that has two spheres in it.
Me: Fly the drone to the sphere. You: Question - There are two spheres in the world, which one do you want me to fly the drone to? Me: Let's pick Sphere 1.
There are two turbines, some solar panels and a car in the world.
Are you ready?

D. AERIAL ROBOTICS: AIRSIM OBSTACLE AVOIDANCE

Full conversation found at:
https://github.com/microsoft/PromptCraft-Robotics/blob/main/examples/aerial_robotics/airsim_obstacleavoidance.md
Initial ChatGPT prompt:

Imagine you are helping me interact with the AirSim simulator for drones. At any given point of time, you have the following abilities. You are also required to output code for some of the requests.
Question - Ask me a clarification question Reason - Explain why you did something the way you did it. Code - Output a code command that achieves the desired goal.
The simulator contains a drone, along with several objects. Apart from the drone, none of the objects are movable. Within the code, we have the following commands available to us. You are not to use any other hypothetical functions.
`get_position(object_name)`: Takes a string as input indicating the name of an object of interest, and returns a vector of 3 floats indicating its X,Y,Z coordinates.
`fly_to(position)`: Takes a vector of 3 floats as input indicating X,Y,Z coordinates and commands the drone to fly there.
`fly_path(positions)`: Takes a list of X,Y,Z positions indicating way-points along a path and flies the drone along that path.
`get_yaw()`: Get the current yaw angle for the drone (in degrees)
`set_yaw(angle)`: Set the yaw angle for the drone (in degrees)
Are you ready?

E. EMBODIED AGENT: HABITAT NAVIGATION

Full conversation found at:
https://github.com/microsoft/PromptCraft-Robotics/blob/main/examples/embodied_agents/visual_language_navigation_1.md
Initial ChatGPT prompt:

Imagine I am a robot equipped with a camera and a depth sensor. I am trying to perform a task, and you should help me by sending me commands. You are only allowed to give me the following commands:
`turn(angle)`: turn the robot by a given number of degrees
`move(distance)`: moves the robot straight forward by a given distance in meters.
On each step, I will provide you with the objects in the scene as a list of <object name, distance, angle in degrees>. You should reply with only one command at a time. The distance is in meters, and the direction angle in degrees with respect to the robot's orientation. Negative angles are to the left and positive angles are to the right. If a command is not valid, I will ignore it and ask you for another command. If there is no relevant information in the scene, use the available commands to explore the environment.

F. EMBODIED AGENT: AIRSIM OBJECT NAVIGATION

Full conversation found at:
https://github.com/microsoft/PromptCraft-Robotics/blob/main/examples/embodied_agents/airsim_objectnavigation.md
Initial ChatGPT prompt:

Imagine you are helping me interact with the AirSim simulator. We are controlling an embodied agent. At any given point of time, you have the following abilities. You are also required to output code for some of the requests.
Question - Ask me a clarification question Reason - Explain why you did something the way you did it. Code - Output a code command that achieves the desired goal.
The scene consists of several objects. We have access to the following functions, please use only these functions as much as possible:
Perception:
get_image(): Renders an image from the front facing camera of the agent
detect_objects(img): Runs an object detection model on an image img, and returns two variables - obj_list, which is a list of the names of objects detected in the scene. obj_locs, a list of bounding box coordinates in the image for each object.
Action:
forward(): Move forward by 0.1 meters.
turn_left(): Turn left by 90 degrees.
turn_right(): Turn right by 90 degrees.
You are not to use any other hypothetical functions. You can use functions from Python libraries such as math, numpy etc. Are you ready?

G. MANIPULATION WITH CURRICULUM LEARNING: PICKING, STACKING, AND BUILDING THE MICROSOFT LOGO

Full conversation found at:
https://github.com/microsoft/PromptCraft-Robotics/blob/main/examples/manipulation/pick_stack_msft_logo.md
Initial ChatGPT prompt:

Imagine we are working with a manipulator robot. This is a robotic arm with 6 degrees of freedom that has a suction pump attached to its end effector. I would like you to assist me in sending commands to this robot given a scene and a task. At any point, you have access to the following functions:
`grab()`: Turn on the suction pump to grab an object
`release()`: Turns off the suction pump to release an object
`get_position(object)`: Given a string of an object name, returns the coordinates and orientation of the vacuum pump to touch the top of the object [X, Y, Z, Yaw, Pitch, Roll]
`move_to(position)`: It moves the suction pump to a given position [X, Y, Z, Yaw, Pitch, Roll].
You are allowed to create new functions using these, but you are not allowed to use any other hypothetical functions. Keep the solutions simple and clear. The positions are given in mm and the angles in degrees.
You can also ask clarification questions using the tag "Question - ". Here is an example scenario that illustrates how you can ask clarification questions.
Let's assume a scene contains two spheres. Me: pick up the sphere. You: Question - there are two spheres. Which one do you want me to pick up? Me: Sphere 1, please.
Use python code to express your solution. Are you ready?

APPENDIX C
ADDITIONAL EXPERIMENTS

A. ZERO-SHOT MANIPULATION: PICK AND PLACE, STACKING, AND PUSHING

Full conversation found at:
https://github.com/microsoft/PromptCraft-Robotics/blob/main/examples/manipulation/manipulation_zeroshot.md

B. CONTROLLING MULTIPLE ROBOT FORM FACTORS FROM A SINGLE PROMPT

Full conversation found at:
https://github.com/microsoft/PromptCraft-Robotics/blob/main/examples/multiple_robots/multiple_robots.md

APPENDIX D
OTHER COMMONSENSE/REASONING TASKS

A. COMPUTER VISION PROBLEM

Imagine we are working with an RGBD camera. The pose of this camera can be accessed via a function get_pose(). Now let us say we observe a point P in the image at coordinates (x, y), and the corresponding perspective depth value can be obtained by get_depth(x, y). The intrinsics of the camera can be obtained by the function get_intrinsics(). Can you give me code that will compute the world coordinates corresponding to point P?

Sure, here's some example code in Python to compute the world coordinates corresponding to point P

```
import numpy as np
def get_world_coordinates(x, y):
# Get camera intrinsics
K = get_intrinsics()
# Get camera pose
pose = get_pose()
# Get perspective depth value at observed
point
depth = get_depth(x, y)
# Compute homogeneous image coordinates
p_image = np.array([x, y, 1])
# Compute inverse camera intrinsics
K_inv = np.linalg.inv(K)
# Compute image coordinates in camera frame
p_camera = depth * K_inv.dot(p_image)
# Compute world coordinates
p_world = np.dot(pose[:3, :3], p_camera) +
pose[:3, 3]
return p_world
```

Given an observed point P in the image at coordinates (x, y), and the corresponding perspective depth value obtained by get_depth(x, y), we can compute the 3D world coordinates corresponding to P. To do this, we need to know the camera intrinsics, which describe how the 3D scene is projected onto the 2D image plane. The intrinsics can be obtained using the get_intrinsics() function.

We also need to know the camera pose, which describes the position and orientation of the camera in the world coordinate system. The pose can be obtained using the get_pose() function.

With the intrinsics and pose, we can transform the observed point P from image coordinates to world coordinates. We first compute the inverse of the camera intrinsics matrix, which allows us to convert from image coordinates to camera coordinates. We then multiply the inverse intrinsics matrix with the observed point's pixel coordinates and the corresponding depth value to obtain the 3D point in camera coordinates.

Finally, we transform the camera coordinates to world coordinates by multiplying the camera coordinates by the camera pose matrix, which describes the transformation from camera coordinates to world coordinates. The resulting 3D point is in homogeneous coordinates, so we need to divide by the last coordinate to obtain the 3D Euclidean coordinates.

B. TRANSFORMATION MATRICES

Imagine we are in a world that operates according to its own specific coordinate convention. There is a robot car located in this world at a specific location. The car has its own coordinate convention. And upon this car, there is a camera which, again, has a different coordinate convention. Imagine we have access to a function get_transformation_matrix(a, b) - that can return the transformation matrix between world and vehicle, or vehicle and camera. What is the transformation matrix between the world and the camera?

The transformation matrix between the world and the camera can be obtained by composing the transformation matrices between the world and the vehicle and the vehicle and the camera. To find the transformation matrix between the world and the camera, we first get the transformation matrix between the world and the vehicle using the function get_transformation_matrix(a, b) with a set to world and b set to vehicle. Next, we get the transformation matrix between the vehicle and the camera using the function get_transformation_matrix(a, b) with a set to vehicle and b set to camera. Finally, we multiply the two transformation matrices to get the transformation matrix between the world and the camera. In mathematical terms, if T_world_vehicle is the transformation matrix between the world and the vehicle, and T_vehicle_camera is the transformation matrix between the vehicle and the camera, then T_world_camera = T_world_vehicle T_vehicle_camera.

Let's get into the details more. Imagine the world operates according to the coordinate convention: X is forward, Y is to the left, Z is upwards. The robot car is located in this world at coordinates (5, 0, 1). The car has its own coordinate convention: X is forward, Y is to the right, and Z is downwards. Can you now implement get_transformation_matrix(world, car)?

Yes, to implement the get_transformation_matrix(world, car) function, we need to first convert the car's coordinate convention to match the world's coordinate convention. The transformation matrix between the world and the car can be represented as follows:

```
def get_transformation_matrix(a, b):
if a == 'world' and b == 'car':
T = np.array([[1, 0, 0, 5], [0, -1, 0, 0],
[0, 0, -1, 1], [0, 0, 0, 1]])
return T
```

C. BALANCING A BALL ON A PLATE

Imagine we are trying to balance a ball on the center of a plate. The plate can be controlled with a pitch and a roll angle as set_plate_pitch(), and set_plate_roll(). The range of pitch and roll angles that can be set are [-0.1, 0.1]. There is an intelligent camera above the plate that can capture top-down images of the plate and return the (X, Y) position of the center of the ball in the image. The image is of resolution 320x240. The pixel coordinates of the ball can be accessed by the function get_ball_position(). Can you give me code that continuously controls the plate such that the ball is always at the center?

Here is an example of the code in Python:

```
import time
def balance_ball():
# Continuously check the position of the ball
while True:
# Get the current position of the ball
x, y = get_ball_position()
# Check if the ball is off-center
if x > 160 or x < 160 or y > 120 or y <
120:
# Calculate the pitch and roll
adjustments needed to center the ball
pitch_adjust = (y - 120) * 0.0001
roll_adjust = (x - 160) * 0.0001
# Limit the pitch and roll adjustments
to the maximum range [-0.1, 0.1]
pitch_adjust = max(min(pitch_adjust,
0.1), -0.1)
roll_adjust = max(min(roll_adjust,
0.1), -0.1)
# Set the pitch and roll angles to
center the ball
set_plate_pitch(pitch_adjust)
set_plate_roll(roll_adjust)
# Wait for a short period before checking
the ball's position again
time.sleep(0.01)
# Start the balancing process
balance_ball()
```

This code uses a loop to continuously check the position of the ball using the get_ball_position function. If the ball is off-center, the code calculates the necessary pitch and roll adjustments to center the ball and sets the plate angles using the set_plate_pitch and set_plate_roll functions. The code limits the pitch and roll adjustments to the maximum range of [-0.1, 0.1]. The loop waits for a short period before checking the ball's position again to prevent over-adjusting the plate.

ACKNOWLEDGMENT

This paper was written with the assistance of OpenAI's ChatGPT [1], with prompts provided by the authors. The model's output was thoroughly revised and adapted, they note that the use of LLMs can significantly speed up the writing process and they recommend their use to the interested reader. *(Sai H. Vemprala and Rogerio Bonatti contributed equally to this work.)*

REFERENCES

[1] (2023). *OpenAI*. Accessed: Feb. 8, 2023. [Online]. Available: https://openai.com/blog/chatgpt/

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[3] T. B. Brown et al., "Language models are few-shot learners," in *Proc. NIPS*, 2020, pp. 1877–1901.

[4] OpenAI. (2023). *Gpt-4 Technical Report*. [Online]. Available: https://arxiv.org/abs/2303.08774

[5] M. Chen et al., "Evaluating large language models trained on code," 2021, *arXiv:2107.03374*.

[6] H. Touvron et al., "Llama 2: Open foundation and fine-tuned chat models," 2023, *arXiv:2307.09288*.

[7] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLAMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[8] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. Le Scao, T. Lavril, T. Wang, T. Lacroix, and W. El Sayed, "Mistral 7B," 2023, *arXiv:2310.06825*.

[9] Y. Hong, Q. Wu, Y. Qi, C. Rodriguez-Opazo, and S. Gould, "A recurrent vision-and-language BERT for navigation," 2020, *arXiv:2011.13922*.

[10] S. Stepputtis, J. Campbell, M. Phielipp, S. Lee, C. Baral, and H. B. Amor, "Language-conditioned imitation learning for robot manipulation tasks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 13139–13150.

[11] N. Muhammad Mahi Shafiullah, C. Paxton, L. Pinto, S. Chintala, and A. Szlam, "CLIP-fields: Weakly supervised semantic fields for robotic memory," 2022, *arXiv:2210.05663*.

[12] A. Bucker, L. Figueredo, S. Haddadin, A. Kapoor, S. Ma, S. Vemprala, and R. Bonatti, "LATTE: Language trajectory transformer," 2022, *arXiv:2208.02918*.

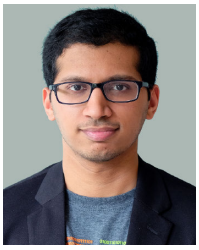
[13] A. Bucker, L. Figueredo, S. Haddadin, A. Kapoor, S. Ma, and R. Bonatti, "Reshaping robot trajectories using natural language commands: A study of multi-modal data alignment using transformers," 2022, *arXiv:2203.13411*.

[14] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," 2022, *arXiv:2209.05451*.

[15] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," in *Proc. Conf. Robot. Learn.*, 2022, pp. 894–906.

[16] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan, "VIMA: General robot manipulation with multimodal prompts," 2022, *arXiv:2210.03094*.

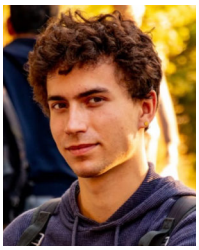
[17] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 9118–9147.
[18] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng,J. Tompson, I. Mordatch, 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," 2022, *arXiv:2207.05608*.
[19] 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," 2022, *arXiv:2209.07753*.
[20] 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," 2022, *arXiv:2209.11302*.
[21] J. Li, D. Li, S. Savarese, and S. Hoi, "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 19730–19742.
[22] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 1–17.
[23] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Proc. Field Service Robot., Results 11th Int. Conf.* Cham, Switzerland: Springer, 2018, pp. 621–635.
[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.
[25] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A platform for embodied AI research," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9338–9346.
[26] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek, "Robots that use language," *Annu. Rev. Control, Robot., Auto. Syst.*, vol. 3, pp. 25–55, Jan. 2020.
[27] J. Arkin, D. Park, S. Roy, M. R. Walter, N. Roy, T. M. Howard, and R. Paul, "Multimodal estimation and communication of latent semantic knowledge for robust execution of robot instructions," *Int. J. Robot. Res.*, vol. 39, nos. 10–11, pp. 1279–1304, Sep. 2020.
[28] M. R. Walter, S. Patki, A. F. Daniele, E. Fahnestock, F. Duvallet, S. Hemachandra, J. Oh, A. Stentz, N. Roy, and T. M. Howard, "Language understanding for field and service robots in a priori unknown environments," 2021, *arXiv:2105.10396*.
[29] J. Fu, A. Korattikara, S. Levine, and S. Guadarrama, "From language to goals: Inverse reinforcement learning for vision-based instruction following," 2019, *arXiv:1902.07742*.
[30] P. Goyal, R. J. Mooney, and S. Niekum, "Zero-shot task adaptation using natural language," 2021, *arXiv:2106.02972*.
[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–12.
[32] F. Giuliari, I. Hasan, M. Cristani, and F. Galasso, "Transformer networks for trajectory forecasting," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, Jan. 2021, pp. 10335–10342.
[33] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 1–19.
[34] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 1273–1286.
[35] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 15979–15988.
[36] R. Bonatti, S. Vemprala, S. Ma, F. Frujeri, S. Chen, and A. Kapoor, "PACT: Perception-action causal transformer for autoregressive robotics pre-training," 2022, *arXiv:2209.11133*.
[37] S. Yitzhak Gadre, M. Wortsman, G. Ilharco, L. Schmidt, and S. Song, "CoWs on pasture: Baselines and benchmarks for language-driven zero-shot object navigation," 2022, *arXiv:2203.10421*.
[38] M. Ahn et al., "Do as I can, not as I say: Grounding language in robotic affordances," 2022, *arXiv:2204.01691*.
[39] P. Sharma, B. Sundaralingam, V. Blukis, C. Paxton, T. Hermans, A. Torralba, J. Andreas, and D. Fox, "Correcting robot plans with natural language feedback," 2022, *arXiv:2204.05186*.
[40] A. Brohan et al., "RT-1: Robotics transformer for real-world control at scale," 2022, *arXiv:2212.06817*.
[41] 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, *arXiv:2204.00598*.
[42] S. J. Russell, *Artificial Intelligence A Modern Approach*. London, U.K.: Pearson Education, Inc., 2010.
[43] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," 2022, *arXiv:2210.17323*.
[44] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 1233–1249.



SAI H. VEMPRALA (Member, IEEE) received the B.Tech. degree in electrical engineering from JNT University, India, the M.S. degree in electrical engineering from Arizona State University in 2013, and the Ph.D. degree in robotics from Texas A&M University, in 2019. From 2019 to 2023, he was a Senior Researcher with Microsoft Research. He is the Co-Founder of Scaled Foundations. He has over 25 peer-reviewed publications. His research interests include perception and planning for robotics, multimodal representation learning, large language models, and simulation. He actively serves as a reviewer/the area chair for several robotics and machine learning conferences.

ROGERIO BONATTI (Member, IEEE) was born in São Paulo, Brazil. He received the B.S. degree in mechatronics engineering from the University of São Paulo and the Ph.D. degree in robotics from the School of Computer Science, Carnegie Mellon University, in 2021.

He is currently a Senior Researcher with the Applied Sciences Group (ASG), Microsoft. His work focuses on multimodal foundational models for decision-making. He creates generative machine-learning models that fuse language, vision, and other features to allow systems to take the best actions over time. Much of his past work before joining ASG was in the robotics space, where he deployed autonomous systems in multiple manipulators, virtual, and real embodied agents all the way to flying robots. His work has been awarded the Best Student Paper Finalist Nomination (IROS 2020), a Microsoft Research Dissertation Grant, a Siebel Scholarship, and a Swartz Entrepreneurship Fellowship.



ARTHUR BUCKER (Student Member, IEEE) received the master's degree from the Technical University of Munich. He is currently pursuing the Ph.D. degree in robotics with Carnegie Mellon University. He is a Roboticist and AI Researcher. His academic journey has been marked by an exploration of autonomous systems, AI, and robotics. His current scholarly endeavors are centered within the domain of robotic learning, with a specific emphasis on leveraging multimodal human–robot interaction to facilitate advanced and efficient cognitive processes in robotic systems.

ASHISH KAPOOR received the Ph.D. degree in computer science from the MIT Media Laboratory. He is currently the CEO and the Co-Founder of Scaled Foundations. Prior to Scaled Foundations, he was the General Manager for autonomous systems research with Microsoft, focusing on building safe AI systems and specifically aerial robotics and its applications in areas that positively influence society. His research interests include machine learning, computer vision, and robotics while touching on various disciplines of computer science that include quantum computation, systems, formal methods, programming languages, and human–computer interaction.