

GROOT: GPT-based Human-RObOT Interface

Shobhit Maniar
Centre for Robotics and Assembly
Cranfield University
Bedford, UK
0009-0001-4340-801X

Dr. Gilbert Tang
Centre for Robotics and Assembly
Cranfield University
Bedford, UK
0000-0002-5497-7269
*Corresponding author

Dr. Marco Chacin
Centre for Robotics and Assembly
Cranfield University
Bedford, UK
marco.chacin@airbus.com

Abstract—Large Language Models (LLMs) have revolutionized the realm of Natural Language Processing (NLP). Their proficiency in planning and reasoning, combined with code generation capabilities, presents a novel avenue for robotics applications. This work introduces GROOT, a novel speech-based language-agnostic middleware that uses instructions and code examples as grounding principle to leverage Generative Pre-Trained Transformer’s ability to produce code for new and unseen tasks. Unlike methods based on language-conditioned robot policies, GROOT capitalises on auto-regressive code generation inspired by Code-as-Policy (CaP) and ProgPrompt. The aim is to create a human-robot interface using GROOT that enables the embodiment of an LLM to take user instructions like ‘move in a square’, ‘move 20 cm in front’, ‘go to position ‘X’ on the grid’ and return policy code based on robot API. In this paper, GROOT was used in a number of experiments to assess its performance to few-shot learning against spatial reasoning, logical reasoning and compound simulated tasks. This work reflects the potential of prompting code-based examples and API-based instructions as a grounding method to integrate large-language models with robotic platforms, envisioning seamless and intuitive human-robot interactions.

Index Terms—Natural Language Processing, Human-Robot Interaction, Embodied Intelligent Agent, Few-shot Planners, Policy Code Generation, Prompt Engineering

I. INTRODUCTION

In November 2022, OpenAI introduced ChatGPT, based on GPT-3.5 Large Language Models (LLMs). This chatbot-style platform engages in conversations and performs various Natural Language Processing (NLP) tasks, sparking interest in its application within robotics. More recently, [1] outlined prompt engineering principles to enhance ChatGPT’s adaptability for high-level robotics tasks.

GPT models excel in natural language generation (NLG), code generation, summarization, arithmetic reasoning, problem-solving, and common sense reasoning [2]–[4], making them valuable for robotic task generalisation. Robots incorporating GPT, function as an embodied intelligent agents (EIAs), leveraging prompts for zero-shot learning in NLP tasks by utilising suitable prompts to condition the model and enable zero-shot learning for the desired downstream task [5].

This article explores the role of ChatGPT-like LLMs in robotics, highlighting their potential as zero-shot planners and their ability to directly interact with robotic platforms by generating executable code for task execution.

II. RELATED WORK

Integrating language into all aspects of a robotic system, perception, planning, and action is essential for shaping the behaviour of language-driven robots [6]. The emergence of transformer-based architectures [7] has enabled modern Large Language Models (LLMs) to excel in zero-shot and few-shot reasoning for robotic planning [5], [8]–[11].

However, a key challenge in LLM-based planning is the lack of direct grounding in the physical world. Although LLMs can generate high-level task plans, they do not inherently understand a robot’s capabilities to perform a task. The SayCan [12] framework addresses this by weighting LLM-generated task plans against a value function, selecting the best plan based on affordance scores for the actions the robot can physically perform. This approach has inspired research on grounding robotic planners with closed-loop feedback for long-horizon tasks [13]. Multi-modal end-to-end frameworks that combine language and visual information either through language-conditioned imitation learning [14], [15] or structured dialogues between Vision-Language Models (VLMs) and LLMs [16] have further advanced robotic planning for complex, long-horizon tasks.

Although language models excel in zero-shot robotic planning, they depend on predefined action primitives, assuming the robot possesses the required skills. Translating action plans into executable policies often requires behavioural cloning and imitation learning. However, using code offers finer control and richer representation rather than mapping text instructions directly to actuator commands, thus, they can be converted into Python functions for execution. This translation-based method outperforms models that directly predict robot actions from natural language instructions [17]. Since Large Language Models (LLMs) are trained on extensive coding datasets [18], they can directly generate policy code for robots based on instructional prompts. API-based instructions and code examples serve as effective prompt engineering strategies for robotic control [19], [20]. Rather than predicting natural language task plans, LLMs generate executable code aligned with robot APIs, removing the need for explicit skill mapping. Additionally, by leveraging prior examples, LLMs can infer structured robotic policies, enhancing adaptability through autonomous construction and refinement of control logic.

OpenAI’s ChatGPT, a dialogue-optimised LLM, offers significant potential for human-robot interaction. While it excels in language inference and comprehension, its common sense reasoning and logical deduction remain limited [21]. However, few-shot prompting in a dialogue format can help ground its responses, enabling the generation of executable code for robotic tasks [1]. By leveraging multi-layered prompting, defining specific roles, providing detailed robot API descriptions, and incorporating example-based prompts, ChatGPT can generate structured plans for diverse tasks [22]. Building on this multi-layer few-shot prompting technique, this work explores ChatGPT’s code-generation capabilities combined with dialogue to develop a speech-based middleware that translates natural language commands into real-time executable code for a highly adaptable human-robot interface.

III. THE GROOT APPROACH

GROOT - GPT-based Human-Robot Interaction, is a speech-based middleware which takes user commands and translates it into Python code which is executed on the robot like NAO and Tello as in Fig. 1 all within runtime.

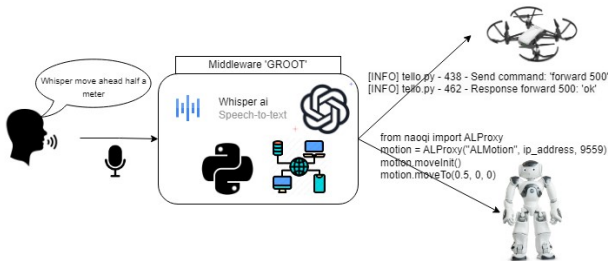


Fig. 1. The Groot Approach

A. Prompt Engineering

To begin with, the behaviour for ChatGPT, a LLM and the central cognitive element, executing procedural actions, devising plans, and generating code, needs to be tuned before deploying it to a robot. This is crucial because ChatGPT lacks inherent context about the environment and user intention. The GROOT approach thus uses multilayer prompt engineering effort to ground ChatGPT in enabling few-shot generalisation, enhancing the LLM’s versatility and ease of adaptation. The multilayer grounding components are:

- 1) *Model role grounding*: The first layer of prompting defines the specific role that GPT assumes within the interaction with the user later.
- 2) *Example-based and API context grounding*: The second layer exploits the dialogue capacity, to strategically provide contexts for generating subsequent programs specific to each robot. This is accomplished through example interactions, that outline the user input and the assistant’s corresponding response prompting API reference of the robots’ library.

B. System Design

The primary mode of interaction for the middleware is through chatbot dialogue, resembling the web-based ChatGPT interaction. Users input the instructions as prompts and receive corresponding responses. These instructions are provided via speech, which is converted into text for dialogue-based interactions with the specific ChatGPT engine. This is achieved using a speech-to-text engine, such as OpenAI's Whisper model. Audio is recorded in five-second chunks, saved as WAV files, and converted into BytesIO objects for streaming to Whisper. The transcriptions are then stored as text files for further processing.

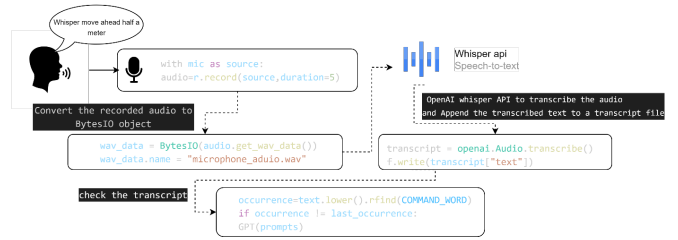


Fig. 2. From Speech to Instruction

A crucial aspect of speech-based HRI is wake word detection. The wake word “Whisper” must precede any instruction sent to ChatGPT. Each detected wake word is assigned an occurrence ID, and the transcript is scanned for the highest-index wake word, ensuring only the latest command is processed by ChatGPT.

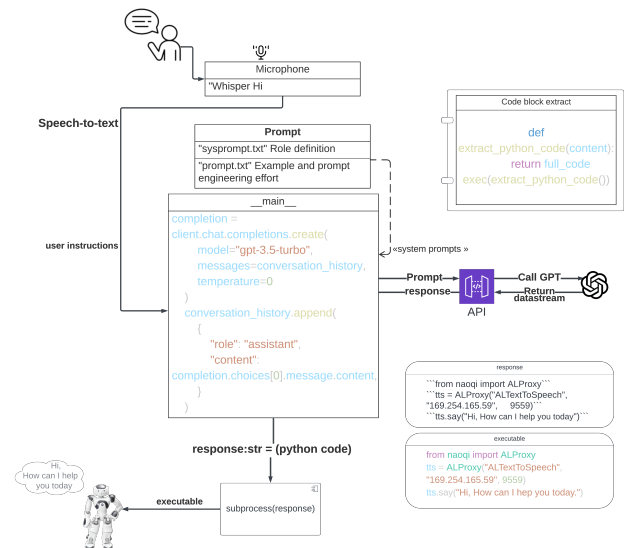


Fig. 3. The System Architecture

The integration of ChatGPT is achieved using the Chat-Completion API, facilitated by the official Python binding. To enable a natural and coherent dialogue flow between the user and the GROOT system, it is crucial to maintain an

active conversation history. Therefore, the messages object is populated with the conversation history stored as a list. Ensuring consistent and deterministic model behaviour in its output is essential. This is achieved by setting the temperature field to 0, resulting in reduced randomness in the generated responses.

Using the grounding method described in Section 3, the prompt strategically provides context for ChatGPT to generate a response containing the specific program for the NAO robot within triple quotes, ensuring easy extraction and real-time execution. When a prompt is sent to the GPT engine, it generates Python code enclosed within triple quotes `"""code"""`. To execute this on the robot, a regular expression compiler extracts the code blocks, which are then concatenated. The final code is executed using the `'exec()'` method within a Python subprocess. Since robots may use different Python versions (e.g., PythonSDK with NAOqi APIs on 2.7 vs. GPT on 3.9), the subprocess ensures compatibility by running the code in the appropriate interpreter.

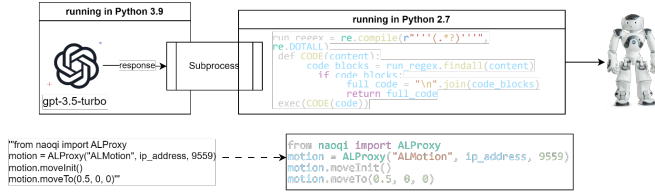


Fig. 4. GPT Response Processing

IV. VALIDATION AND ANALYSIS

A. Groot as Few-shot Planner

To validate GROOT as an effective middleware for robot planing and assess its adaptability beyond the provided prompts, structured interactions are conducted in a controlled environment using the NAO robot. The system is assessed based on its response to a user instruction by executing the output generated by ChatGPT, which is code, and evaluating the successful action of a robot. The validation process involves testing the system's zero-shot learning capabilities with instructions involving various API function calls.

1) *Single Instruction Single Output*: During these tests, a single instruction is provided, and the generated code gets executed in real time. This approach evaluates the system's ability to adapt to novel and previously unseen commands that were not explicitly included in the example prompts.

2) *Compound Instruction Compound Output*: Although single-instruction testing evaluates adaptability, it is equally important to assess performance with complex, compound instructions. Therefore, to test performance and generalisation, the system is challenged with intricate and compound instructions that involve multiple actions requiring multiple API function calls together, providing deeper insight into ChatGPT's contextual understanding and effectiveness as an embodied agent that handle sequential commands.

3) *Results*: These tests in Fig. 5 and 6, successfully demonstrate that the multilayered single-shot approach grounds the LLM effectively, enabling it to generalise across new and unfamiliar tasks. ChatGPT effectively understands the context of user queries, the role of the embodied robot, and the appropriate API functions to utilise. Particularly, in an instruction to tilt its head, despite the absence of explicit API context within the example prompts, for head motion, ChatGPT's extensive pre-training on an internet-scale codebase enables it to successfully generate the required code for the NAO robot as seen in Fig. 5.

<p>say something using command word Whisper</p> <p>Listening</p> <p>transcribing.....</p> <p>Whisper, stand up.</p> <p>ready</p> <p>"""from naoqi import ALProxy</p> <p>motion = ALProxy("ALRobotPosture", ip_address, 9559)</p> <p>motion.goToPosture("Stand", 1.0)"""</p> <p>Please wait while I run the code</p> <p>Single Command: Stand Up</p>	<p>say something using command word Whisper</p> <p>Listening</p> <p>transcribing.....</p> <p>Whisper, can you walk two steps ahead?</p> <p>ready</p> <p>"""from naoqi import ALProxy</p> <p>motion = ALProxy("ALMotion", ip_address, 9559)</p> <p>motion.moveInit()</p> <p>motion.moveTo(0.2, 0, 0)"""</p> <p>Please wait while I run the code</p> <p>Single Command: Walk 2 Steps</p>
<p>say something using command word Whisper</p> <p>Listening</p> <p>transcribing.....</p> <p>Whisper, tilt your head by 180 degrees left.</p> <p>ready</p> <p>"""from naoqi import ALProxy</p> <p>motion = ALProxy("ALMotion", ip_address, 9559)</p> <p>motion.setAngles("HeadYaw", -1.57, 0.5)"""</p> <p>Please wait while I run the code</p> <p>Single Command: Tilt Head</p>	<p>say something using command word Whisper</p> <p>Listening</p> <p>transcribing.....</p> <p>Whisper, can you turn 180 degrees?</p> <p>ready</p> <p>"""from naoqi import ALProxy</p> <p>motion = ALProxy("ALMotion", ip_address, 9559)</p> <p>motion.moveInit()</p> <p>motion.moveTo(0, 0, 3.14)"""</p> <p>Please wait while I run the code</p> <p>Single Command: Rotate by 180 degrees</p>
<p>say something using command word Whisper</p> <p>Listening</p> <p>transcribing.....</p> <p>Whisper, introduce yourself.</p> <p>ready</p> <p>"""from naoqi import ALProxy</p> <p>tts = ALProxy("ALTextToSpeech", ip_address, 9559)</p> <p>tts.say("Hello, my name is NAO. I am a humanoid robot designed to assist and interact with humans. I can perform various tasks such as speaking, listening, and moving. I am here to help you with any questions or tasks you may have. How can I assist you today?")"""</p> <p>Please wait while I run the code</p> <p>Single Command: Introduce itself</p>	<p>say something using command word Whisper</p> <p>Listening</p> <p>transcribing.....</p> <p>Whisper, tell me something about Milky Way galaxy.</p> <p>ready</p> <p>"""from naoqi import ALProxy</p> <p>tts = ALProxy("ALTextToSpeech", ip_address, 9559)</p> <p>tts.say("The Milky Way is a barred spiral galaxy that is part of the Local Group of galaxies. It is estimated to contain between 100 and 400 billion stars, including our own Sun. The Milky Way is about 100,000 light-years in diameter and is home to various astronomical objects such as nebulae, star clusters, and black holes. It is named after the milky appearance of its band of light when seen from Earth.")"""</p> <p>Please wait while I run the code</p> <p>Single Command: General knowledge question</p>

Fig. 5. Single Instruction Single Output

<p>say something using command word Whisper</p> <p>Listening</p> <p>transcribing.....</p> <p>Whisper, introduce yourself and then walk ahead half a meter and then later sit down and explain about Andromeda galaxy and then stand up and go in a square and then tilt your head left by 180 degrees and then by right 180 degrees and then say goodbye.</p> <p>ready</p> <p>"""from naoqi import ALProxy</p> <p># Introduce yourself</p> <p>tts = ALProxy("ALTextToSpeech", ip_address, 9559)</p> <p>tts.say("Hello, I am NAO, a humanoid robot.")</p> <p># Walk ahead half a meter</p> <p>motion = ALProxy("ALMotion", ip_address, 9559)</p> <p>motion.moveInit()</p> <p>motion.moveTo(0.5, 0, 0)</p> <p># Sit down and explain about Andromeda galaxy</p> <p>posture = ALProxy("ALRobotPosture", ip_address, 9559)</p> <p>posture.goToPosture("Sit", 1.0)</p> <p>tts.say("The Andromeda galaxy is a spiral galaxy approximately 2.537 million light-years away from Earth.")</p> <p># Stand up and go in a square</p> <p>posture.goToPosture("Stand", 1.0)</p> <p>motion.moveTo(0.5, 0, 0)</p> <p>motion.moveTo(0, 0.5, 0)</p> <p>motion.moveTo(-0.5, 0, 0)</p> <p>motion.moveTo(0, -0.5, 0)</p> <p># Tilt head left by 180 degrees</p> <p>motion.setAngles("HeadYaw", 1.0, 0.5)</p> <p># Tilt head right by 180 degrees</p> <p>motion.setAngles("HeadYaw", -1.0, 0.5)</p> <p># Say goodbye</p> <p>tts.say("Goodbye!")</p> <p>Please wait while I run the code</p>

Fig. 6. Complex Compounded Instruction Single Output

B. Spatial Reasoning

Spatial reasoning is the ability to understand, visualise, and manipulate objects and their relationships in space. The effectiveness of a robust robot system is based on its ability to perform tasks within a spatial context. Typically, a multi-modal VLM architecture is preferred to make the robot move. Although such systems have proven to solve complex tasks with good probability, it would be beneficial to understand how a single LLM would work, which is tasked with cognitive reasoning on spatial information. Having performed adequate tests for single shot long horizon task, further capabilities of ChatGPT is measured for spatial reasoning tasks.

1) *Test Environment*: An experiment with a 4x4 grid as illustrated in Fig. 7, is set up. Each grid is a square cell of 30 by 30 cm, numbered sequentially from 1 to 16 arranged in a lawnmower pattern. In order to effectively ground the GPT engine, prompt engineering techniques are applied.

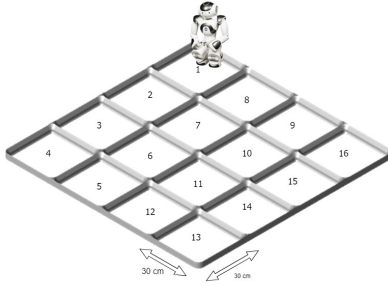


Fig. 7. Grid Experiment setup

2) *Prompt Engineering*: ChatGPT is provided an experiment in the context of a game along with a position dictionary of the game coordinates. In addition to these, an example interaction specific to the game is also provided to help understand how to move from one position to another which it would later adapt to more complex commands.

3) *Results*: The Nao Robot is placed on position 1 of the grid and instructions are given to traverse along different positions. From the experiment result in Fig. 8, GPT shows an exceptional understanding of spatial context purely from language descriptions and responds successfully to complex instructions. Remarkably, when instructed to traverse different positions, it situates the robot within the grid and dynamically updates its current position accordingly. This highlights its ability to infer and maintain spatial awareness based solely on linguistic input, reinforcing its potential for autonomous robotic navigation.

C. Prompt Engineering based Scalability to different robots

As a means to test the generalisability of the architecture, the Tello drone is introduced to be embodied by ChatGPT. Since the system is robust, a changed prompt context is provided, and in addition to giving examples, API usage of the DJITelloPy Library is provided, which enables tele-operation of the drone.

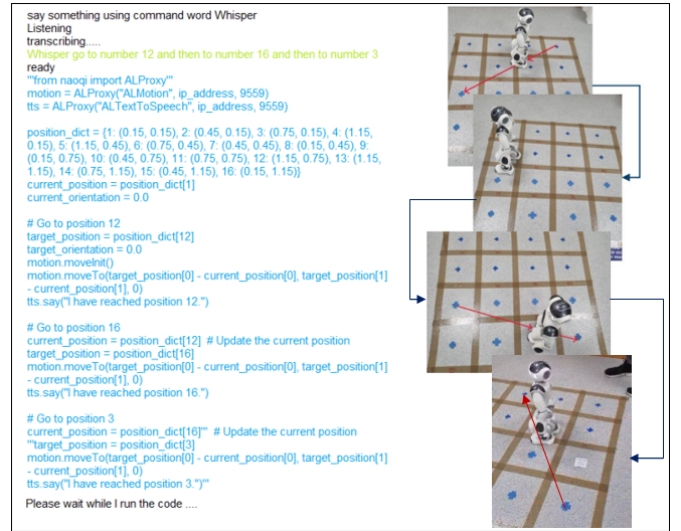


Fig. 8. Successful spatial reasoning task adaptability

1) *Results*: Similar to NAO robot testing, ChatGPT is evaluated on complex tasks beyond simple teleoperation. For example, when instructed to move in a square, it generates and executes a motion policy for the trajectory. This enables Tello to follow desired paths, showcasing GROOT's adaptability as a middleware to embody robots seamlessly.

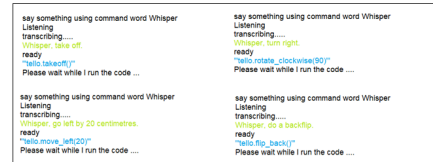


Fig. 9. Grounded Policy code generation for Tello teleoperation

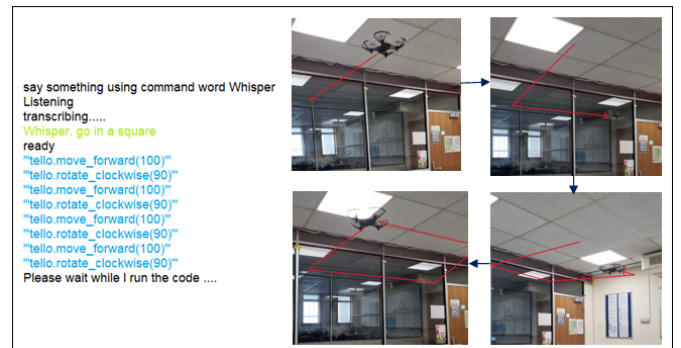


Fig. 10. Grounded Policy code generation for Tello Flight Path

V. DISCUSSION

GROOT unequivocally demonstrates its suitability for Human-Robot Interaction (HRI) as a robust middleware. Its remarkable flexibility and ability to handle Human-Robot Collaboration (HRC) without requiring complex architectural

re-configurations provide a distinct advantage over other language-based robotic control approaches.

In particular, compared to [23], who utilised a robot ontology to map action sequences to the corresponding action words and body parts for the NAO robots, the proposed GPT-based human-robot interaction (GROOT) architecture excels in both task execution efficiency and adaptability. Beyond outperforming their method, GROOT also exhibits the potential to handle more intricate and previously unseen tasks, further solidifying its value in robotic applications.

However, it is important to acknowledge the limitations of the proposed architecture. The primary failure of GROOT arises from the challenges in the auto-regressive nature of ChatGPT. Frequently, ChatGPT exhibit a vulnerability known as hallucinations, a characteristic shortcoming of auto-regressive transformer models. Often, vague prompting can lead ChatGPT to generate suboptimal, non-physical task policies. Additionally, its auto-regressive nature can propagate errors, requiring a reset to correct mistakes. As a generative model, ChatGPT may also rewrite code differently for the same task, increasing the risk of failure.

Say something using command word Whisper
Listening
transcribing
Whisper walk 2 steps to the right
ready
python

```
from naoci import ALProxy
motion = ALProxy("ALMotion", ip_address, 9559)
# Walk 2 steps to the right
motion.moveInit()
motion.moveTo(0, 0.2, 0)
...
```

Please wait while I run the code

Incorrect response structure (NAO)

Non-optimal and unrealistic code for moving in circle (Tello)

```
Whisper say to number 12 and then to number 16 and then to number 3
ready
""" complete code hidden for brevity
def move_to_node(node):
    target_position = position_dict[node]
    target_orientation = 0.0
    motion.moveInit()
    motion.moveTo(target_position[0] - current_position[0], target_position[1] - current_position[1], 0)
    tts.say('I have reached node {} + str(node))
    current_node = node
    current_position = target_position

move_to_node(14)
move_to_node(12)
move_to_node(3)
Please wait while I run the code ....
```

Incorrect GPT code generation for instruction (NAO)

Fig. 11. Hallucination in ChatGPT output

At the same time it should be noted that there are newer, faster and improved GPT models which could potentially solve some of the limitations and hallucinations detected during these experiments. Newer advanced models such as reasoning o1-min or OpenAI’s flagship GPT-4o and the cost optimized GPT-4o mini can be used to replace the legacy GPT model incorporated. The choice of using GPT-3.5-turbo model was justified as it was the only model available at the time of this research due to its closeness to ChatGPT and affordability.

Despite these limitations, the novelty of GROOT arise from its architecture which allows user to select their own GPT model and robot to embody. GROOT establishes prompt engineering efforts to effectively ground a LLM and enhance

human-robot interaction for in-context long-horizon task generalisation. These efforts remain open for experimentation and require iterative process before a final deployment can be carried for large scale industrial application. These meticulously crafted prompt engineering endeavours remain a crucial tool for realising the potential of GPT-based Human-Robot Interactions. Future advancements could integrate hardware-in-the-loop mechanisms, allowing robots to provide feedback for self-correcting GPT-generated code. Multimodal approaches like Inner Monologue [13] could enhance applicability, while, combining Chain-of-Thought prompting with visual inference [24], [25] may further scale the GROOT architecture.

Additionally, the growing concerns surrounding data protection and cybersecurity raise questions about the viability of scaling GPT API for industrial-grade applications. The emergence of open-sourced LLM models offers an alternative approach, enabling the development of robotic applications locally without the need for internet connectivity or API dependency. This shift not only enhances security but also reduces processing delays in executing instructions. Therefore, it would be advantageous to explore the potential of locally deployed open-source LLMs to further broaden the capabilities of the system.

VI. CONCLUSION

Limited programming experience and robotic background hinder users from effectively utilising robots. This article explores integrating LLMs like GPT with robots through GROOT, a speech-based middleware, enabling long- and short-horizon manipulation. It argues that LLMs, such as ChatGPT, can be adapted for behavioral responses via dialogue while leveraging their logical task planning and policy code generation to advance robotics. Auto-regressive code generation could make deployment and execution of GPT based robot faster and give rich context since code provide more low-level control than using imitation learning. GROOT thus uses example-based prompts with documentation of APIs and constraints to ground GPT and allow direct embodiment of a robotic platform for much intuitive and personal human-robot interaction.

REFERENCES

- [1] S. H. Vemprala, R. Bonatti, A. F. C. Buckner, and A. Kapoor, “Chatgpt for robotics: Design principles and model abilities,” *IEEE Access*, vol. 12, pp. 55682–55696, 2023.
- [2] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, and T. Gutman-Solo, “Solving quantitative reasoning problems with language models,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 3843–3857, 2022.
- [3] A. Roberts, C. Raffel, and N. Shazeer, “How much knowledge can you pack into the parameters of a language model?,” *arXiv preprint arXiv:2002.08910*, 2020.
- [4] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, “Harnessing the power of llms in practice: A survey on chatgpt and beyond,” *ACM Trans. Knowl. Discov. Data*, vol. 18, 4 2024.
- [5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, and et al, “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

- [6] S. Tellex, N. Gopalan, H. Kress-Gazit, and C. Matuszek, “Robots that use language,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 25–55, 7 2020. doi: 10.1146/annurev-control-101119-071628.
- [7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [8] P. Goyal, R. J. Mooney, and S. Niekum, “Zero-shot task adaptation using natural language,” *arXiv preprint arXiv:2106.02972*, 2021.
- [9] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, “Language models as zero-shot planners: Extracting actionable knowledge for embodied agents,” in *International Conference on Machine Learning*, pp. 9118–9147, PMLR, 2022.
- [10] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, “Large language models are zero-shot reasoners,” *Advances in neural information processing systems*, vol. 35, pp. 22199–22213, 2022.
- [11] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, 2022.
- [12] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, and et al, “Do as i can, not as i say: Grounding language in robotic affordances,” in *Proceedings of The 6th Conference on Robot Learning*, vol. 205, pp. 287–318, PMLR, 7 2023.
- [13] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, and Y. Chebotar, “Inner monologue: Embodied reasoning through planning with language models,” *arXiv preprint arXiv:2207.05608*, 2022.
- [14] M. Shridhar, L. Manuelli, and D. Fox, “Cliport: What and where pathways for robotic manipulation,” in *Conference on Robot Learning*, pp. 894–906, PMLR, 2022.
- [15] M. Shridhar, L. Manuelli, and D. Fox, “Perceiver-actor: A multi-task transformer for robotic manipulation,” in *Conference on Robot Learning*, pp. 785–799, PMLR, 2023.
- [16] A. Zeng, M. Attarian, B. Ichter, K. Choromanski, A. Wong, S. Welker, F. Tombari, A. Purohit, M. Ryoo, and V. Sindhwani, “Socratic models: Composing zero-shot multimodal reasoning with language,” *arXiv preprint arXiv:2204.00598*, 2022.
- [17] S. V. Gubbi, R. Upadrashta, and B. S. Amrutur, “Translating natural language instructions to computer programs for robot manipulation,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1919–1926, 2021.
- [18] Y. Chen, R. Xu, Y. Lin, and P. A. Vela, “A joint network for grasp detection conditioned on natural language commands,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4576–4582, 2021.
- [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,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, IEEE, 2023.
- [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,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530, 2023.
- [21] C. Qin, A. Zhang, Z. Zhang, J. Chen, M. Yasunaga, and D. Yang, “Is chatgpt a general-purpose natural language processing task solver?,” in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.
- [22] N. Wake, A. Kanehira, K. Sasabuchi, J. Takamatsu, and K. Ikeuchi, “Chatgpt empowered long-step robot control in various environments: A case application,” *IEEE Access*, vol. 11, pp. 95060–95078, 2023.
- [23] G. Bardaro, D. Dessí, E. Motta, F. Osborne, and D. R. Recupero, “Parsing natural language sentences into robot actions,” in *CEUR workshop proceedings*, vol. 2456, pp. 93–96, 2019.
- [24] M. G. Arenas, T. Xiao, S. Singh, V. Jain, A. Z. Ren, Q. Vuong, and et al, “How to prompt your robot: A promptbook for manipulation skills with code as policies,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4340–4348, 2024.
- [25] M. Zawalski, W. Chen, K. Pertsch, O. Mees, C. Finn, and S. Levine, “Robotic control via embodied chain-of-thought reasoning,” in *8th Annual Conference on Robot Learning*, 2024.



DSPACE

<https://dspace.org/>

GROOT: GPT-based human-RObOT interface

Maniar, Shobhit; Tang, Gilbert; Chacin, Marco

2025-12-04

Institute of Electrical and Electronics Engineers (IEEE)

<https://doi.org/10.1109/iccr67607.2025.11372068>

Maniar S, Tang G, Chacin M. (2025) GROOT: GPT-based human-RObOT interface. In: Proceedings of the 2025 7th International Conference on Control and Robotics (ICCR), 4-6 Dec 2025, Kyoto, Japan, pp. 7-12