Enhancing Autonomous Task Execution in Social Robots with Large Language Models

1st Luccas Rojas Becerra
Department of Systems and Computing Engineering
Universidad de los Andes, Colombia
Bogotá, Colombia
l.rojasb@uniandes.edu.co
*Corresponding author

2nd Juan Romero Colmenares
Department of Systems and Computing Engineering
Universidad de los Andes, Colombia
Bogotá, Colombia
j.romero11@uniandes.edu.co

3rd Ruben Manrique
Department of Systems and Computing Engineering
Universidad de los Andes, Colombia
Bogotá, Colombia
rf.manrique@uniandes.edu.co

***Abstract*—The field of Social Robotics focuses on developing autonomous robots that can interact socially and assist with various tasks. However, the design and execution of such robots are complex they need to understand their environment and accurately interpret human-level instructions, and a common issue is the mismatch between the instructions inputted and the robot's actual performance in specific contexts. This study therefore proposes to enhance a Pepper robot's autonomy and natural interaction by enhanced instructions via Large Language Models (LLMs). Our study involves evaluating commercial and open-source LLMs' performance and comparing different prompting strategies. Both automatic and human evaluations were carried out and showed the potential and limitations of an approach guided by LLMs. Results showed that 55 percent of tasks generated passed the automatic runtime execution assessment, and GPT-4 achieved the highest success rate.**

Keywords—Social Robotics, Natural Language Processing, Autonomous Robots, Human-Robot Interaction, Large Language Models, Prompting Strategies, Pepper Robot, Code Generation

I. Introduction

The field of Social Robotics is dedicated to the development of robots as interactive companions and assistants. The primary objective is to create multifaceted robots capable of performing a variety of tasks autonomously to aid human beings. However, it remains a complex endeavor to develop robots that can accurately understand their environment and interpret human instructions. A major challenge is the discrepancy between the commands given and the actual behavior of the robot in specific contexts. Current robotics often require manual task-specific programming, resulting in a limitation when subjected to different activities. This deficit arises due to limited cognition in pre-programmed deterministic tasks and difficulty in comprehending the rationale behind their actions. Cognition, defined by Luca Iocci et al., involves understanding the environment and user needs to reasonably execute actions demonstrating comprehension of user requirements [1].

This study utilizes the Pepper robot, developed by SoftBank, globally recognized for its claimed ability to recognize faces and basic human emotions. Selected for its advanced features, such as dual cameras, 3D eye camera, multiple microphones, as well as touch and collision sensors housed in a gyroscope-equipped torso, the Pepper robot provides a lifelike quality surpassing mere doll-like characteristics to represent a more genuine human-like presence.

In light of the advent of large language models, opportunities arise for producing robots competent at performing various tasks. The broad range of applications provided by these models can significantly benefit society in various areas including code and writing auto-completion, grammar improvement, narrative generation for games, refining search engine results, and answering queries [4]. Employing these language models may enhance collaboration between humans and robots, advancing us toward the goal of highly capable robots performing diverse general tasks.

This research paper presents the development and assessment of a system for a Pepper-like robot designed to autonomously follow instructions, communicated through natural language, to perform general tasks. This process encompasses several subtasks. Initially, we formulate an interface module designed to bridge the gap between primitive robot commands and advanced behaviors. Following this, we employ both commercial and open-source Language-to-Language Modules (LLMs) to create a code specific to Pepper's mediator. By establishing multi-tiered abstraction within the robot's coding base, we are able to adapt the potential outputs of the LLMs through directives. The final stage of the research entails a comprehensive evaluation of the code produced by LLMs and the robot's response under five possible outcomes: executed but failed, lack of robot capabilities, not completed by the model, partially completed, and successfully completed.

II. Related Work

The current emphasis in social robotics revolves around enhancing effective interaction between humans and robots, with natural language taking center stage as a method of communication [5]. This push to establish simpler interaction avenues is driven by the fact that most robot users do not have expertise in robotics. The factors impeding effective human-

robot interactions include environmental noise and operator bias, specifically from those who are experts in the domain [6].

Moreover, the integration of natural languages into robot planning has traversed beyond interaction into code generation, a fundamental aspect of self-planning code construction. However, the established approaches, such as manual prompt construction and the generation of sub-tasks using hierarchical structures, have some inherent limitations [7]. Specifically, Large Language Models (LLMs), which are widely used, have exhibited challenges with the stability of the generated codes, resulting in a high degree of unpredictability in the output. This lack of determinism has been manifested across various code generation benchmarks [8].

Despite these hurdles, LLMs are being increasingly employed in the realm of robotics, given their efficacy in enhancing the quality of human-robot interactions and executing complex tasks. Microsoft, for instance, has reported the effectiveness of ChatGPT, an LLM, in transforming natural language into executable tasks. The process involves using a robot function library to create a specific prompt for ChatGPT, which is then assessed before deployment onto the physical robot. Appropriately harnessing the potential of LLMs is, however, not without challenges, necessitating intensified human supervision to guarantee the quality of the solutions and safety [9].

Further studies have highlighted the role of LLMs in helping robots comprehend their surroundings for effective task execution. Singh et al. (2022) elucidate how LLMs can arm robots with an understanding of common-sense knowledge regarding objects and locations, which subsequently facilitates successful deployment. The Codex model, for example, demonstrated a success rate over 40% in executing intricate tasks without errors [10].

III. Methodology

To advance a robot towards achieving general-purpose functionality, it is vital to equip the robot with a well-rounded suite of tools which facilitate the perception of its environment and the interpretation of human verbal commands. Our primary objective is to enable the robot to independently generate the procedural steps required to implement tasks as assigned by humans. This objective was realized through the systematic implementation of three core steps:

1. The initial phase is characterized by the transcription of verbal instructions into a text format, thus enabling subsequent analysis by the robot (carried out via a Text to Speech process).

2. Following this, the transcribed textual instructions are processed by a broad language model, tasked with the generation of precise code instructions tailored for robot execution.

3. After receiving these code instructions, the robot then executes the commanded actions, thereby accomplishing the assigned task.

A significant portion of the focus in this study is concentrated on the optimization of the second step. Said optimization entails the assessment of different large language models, distinct prompts, and varying degrees of abstraction within the generated code. The overarching goal of large language models is to intelligently strategize the essential steps required to fulfill each task and then generate code that ensures the successful execution of the given instructions. To illustrate the operation of the expected input and output, Example 1 is presented.

Example 1.

Input: Robot please look for Angel in the corridor, follow her, and tell what day is today.
Expected Output:

```
from datetime import datetime
# Step 1: Navigate to the corridor
self.tm.go_to_place("entrance")
# Step 2: Look for Angel in the corridor
and follow him
found_angel =
self.tm.find_object("person",timeout=25)
if found_angel:
    self.tm.talk("I found Angel in the
corridor! I will now follow Angel")
    self.tm.follow_you()
else:
    self.tm.talk("I couldn't find Angel in
the corridor. I cannot do that task.")
# Step 3: Answer what day is today
day = datetime.today().strftime("%A")
self.tm.talk("Today is " + day)
```

Command Input Generation: In the context of our research, the "Command Input Generation" phase involves generating distinct input scenarios for the pepper robot. These scenarios cover a range of task cases utilized within the RoboCup competition, including simple, complex, and even tasks that may be considered impossible due to contextual constraints (i.e "Go pick up an apple from the kitchen", but there is no apple to be found in the kitchen). To facilitate this process, we have chosen to utilize two specific scripts, both of which have been developed by the RoboCup@Home team. The initial script concerns an earlier generation of tasks, dating back to 2013 [11], whereas the subsequent script encapsulates tasks of a more recent nature, reflective of the year 2023 [12]. Through the integration of these scripts, our objective is to generate as many possible valid tasks that the robot could face in a real case scenario.

Large Language Models: In our research, the primary focus of our investigation lies in the evaluation of three prominent language models: Llama 2 from Meta, GPT-3.5 Turbo and GPT-4 from OpenAI. These models have been selected as the central subjects of our study due to their significance in the field.

Prompting: Regarding the assessment of prompts, our analysis encompasses a comparison between two distinct approaches: a comprehensive single prompt, encompassing the robot's codebase interface, verbal instructions, and general constraints, versus a chained-prompt methodology. In the latter approach, verbal instructions are deconstructed into simpler steps by the model beforehand. These individual steps are then incorporated into a subsequent prompt adhering to the same

structure as the first approach, ultimately generating the required code to accomplish the given task.

A. General Evaluation

The evaluation process is facilitated by generating an array of tasks using the task generator. The results of these tasks are categorized into distinct outcomes: ”executed but failed”, ”not completed by model”, ”partially completed”, and ”successfully completed”. Furthermore, we also included an additional outcome for when the task is too complex and the robot has a lack of capabilities in order to successfully perform it, this alternative outcome was labeled ”lack of robot capabilities”. This comprehensive assessment framework aims to refine the approach and lead to enhanced robot performance. To effectively conduct this evaluation, it was performed and categorized into two sections: automatic evaluation and human evaluation.

i. *Automatic Evaluation:* For the automated assessment, an automatic workflow was established where tasks are generated, processed by the LLMs, and the output is checked for both invalid parameters within the task’s context and correct code execution (i.e. code being interpreted by Python without exceptions). Subsequently, these results were then recorded in a database, setting the stage for the subsequent evaluation phase. This comprehensive approach addresses both the execution and compilation aspects of the framework.

ii. *Human Evaluation:* Next, the outcomes that successfully pass the automatic evaluation undergo manual execution by the Pepper robot. These results were then reassessed with regard to the remaining outcome categories, namely ”not completed by the model”, ”lack of robot capabilities”, ”partially completed the task” and ”successfully completed the task.” This hands-on evaluation was conducted by the authors.

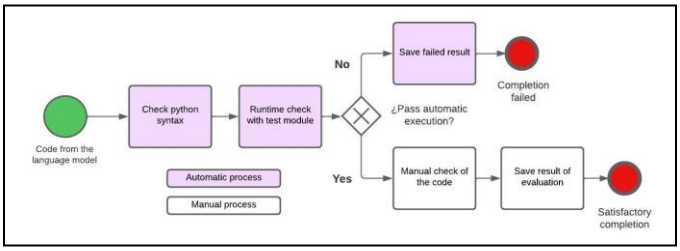

Fig. 1. General Evaluation Process

Figure 1 illustrates the sequential process outlined in the previously described methodology. This process remains in dependent of the specific large language model or prompting strategy employed.

B. LLMs Processing Details

The focal point of this project, as illustrated in Figure 1, is the application of natural language processing to generate executable robot code. This multifaceted process first requires the creation of a coding interface comprising high-level functions for robotic use. These functions, including actions such as speaking, counting, moving, questioning, object-locating, following, and others, should be designed for ease of comprehension by the language model. Our testing indicates that the granularity of these functions is critical: higher abstraction levels tend to yield fewer model errors. Hence, it is more effective to design high-level functions that coincide with the language model’s world conception. The parameters of this interface should be intuitive and well-defined.

This project considers two types of prompts for the coding interface: ”Long String” and ”Chain” prompts. The former includes all necessary instructions in a single lengthy string, while the latter divides tasks into simpler, sequential steps. In the ”Long String Prompting” approach, the prompt comprises a structured set of essential task components. These elements include: the model’s objective; available task module functions with syntax, parameters, and outputs; predefined entities and tags; the output format, which must be a continuous Python code snippet; a real-world example of the desired output; limitations of the robot, which the language model must process to avoid attempting infeasible tasks; and the input task, which the model should solve using its distilled knowledge. A detailed example of a Long String Prompting can be found in the project repository: https://github.com/PepperGPSR/PepperGPSR.

Contrarily, the ”Chain Prompting” approach aims to boost model performance by parsing tasks into manageable steps. This methodology uses a sequenced interaction with the language model to optimize code generation. A detailed description of each step in the ”chain” can be found also in our repository.

IV. RESULTS

We created 720 tasks using GSPR Task Generators developed for the RoboCup Competition [12]. For the automatic evaluation, among the three LLMs evaluated, GPT-4 notably achieved the highest passing rate at 81.1%, GPT-3.5 obtained a 50.8%, and Llama2 had the lowest passing rate, standing at only 28.8% in the context of automatic assessment. The 400 tasks that successfully passed underwent a thorough human review.

For this human evaluation process, we employed a framework that encompassed four categories: "NOT COMPLETED" "PARTIALLY COMPLETED," "SUCCESSFULLY COMPLETED," and "LACK OF CAPABILITIES." In tasks labeled as "Not Completed", the generated code involves models executing actions completely unrelated to the proposed task or, in certain instances, failing to achieve anything useful in addressing the task at hand. Within the "Partially Completed" category, we analyze task outcomes where the sequence of steps mimics successful completion, but key elements crucial to the task fail to be accomplished. For example, overlooking a location name to omitting any step in the process, all while maintaining the logical continuity of the step sequence. In the context of the "Successfully Completed" category, as the name implies, it encompasses all outcomes featuring generated code that successfully accomplishes 100% of the given task.

Finally, the "Lack of Capabilities" category, in essence, comprises outcomes where the model recognizes that the robot lacks the essential capabilities to successfully execute the given task and responds accordingly. This is a tricky category, however, since there are situations where the large language models are creative enough to circumvent this absence of capabilities in some way or another. For instance, the robot does not currently have the capability to distinguish people with black hats or differentiate between young and old people. However, the model can do multiple things to go around this limitation. Such as, asking the person if they have a black hat or asking their age to corroborate if they are young or not. When the model successfully bypasses these limitations, the task may be categorized as "Partially Completed" or "Successfully Completed" based on the overall result. However, if the model accurately acknowledges its inability to perform certain actions, the task falls into the "Lack of Capabilities" category. Additionally, when the model falsely claims incapability despite possessing the necessary capabilities, the task is classified as "Not Completed."

A. *Overall Model Performance*

The evaluation process was conducted on a total of 720 tasks. Out of these, 400 tasks successfully underwent automated evaluation, indicating the absence of Python errors, including syntax and runtime errors. Subsequently to this, the 400 tasks underwent human evaluation based on the constraints specified previously. The results after the complete manual assessment are shown in Figure 2.

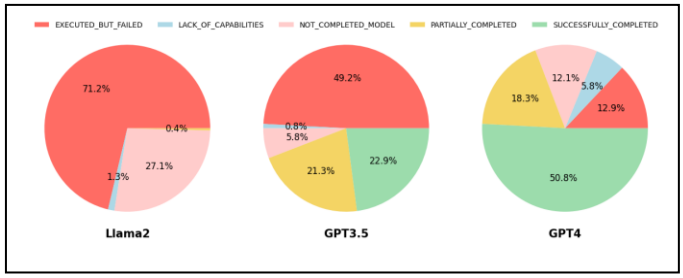

Fig. 2. Overall Task Results

The results obtained for Llama 2 (7B parameters) depicted in the figure were remarkably disappointing, as it failed to record any successful attempts. All results fell short in terms of task fulfillment, with only one task showing partial completion. It is worth noting that there are larger Llama models, but they could not be used due to limitations in the hardware required to run them. In comparison, GPT 3.5's outcomes were somewhat better, though approximately half of them did not pass the automatic evaluation. However, there was discernible progress, as approximately 40% of the tests either partially or successfully concluded, with at least half of this cohort being tasks the model managed completed successfully. Moreover, there was a further improvement in performance with GPT 4. About 90% of the evaluations succeeded, and half of these were considered as completed. It is key to note that, while almost a fifth of tasks were only partially done, about 30\% of them utterly failed. It bears mentioning that these are just aggregated outcomes from the overall evaluation, more precise and detailed analysis can be obtained in the following section.

B. *Prompting Type Effectiveness*

The results for the 'Chain Prompting' strategy are displayed in Figure 3, where each bar signifies the number of tasks that fall into each category post-evaluation. Notably, the application of this strategy didn't achieve optimal results across all models. However, it was advantageous for reducing syntactical errors when invoking functions from the task module and adept at identifying tasks that were unexecutable due to the robot's limitations. However, it must be acknowledged that the overall success rate for this strategy didn't exceed 50% for any model.

The 'Long-String Prompting' strategy, on the other hand, delivered surprisingly superior results compared to the 'Chaining Strategy' (see Figure 4). This is evidenced in the subsequent graph, which outlines the results achieved through the implementation of this strategy. Consistently, the Long-String strategy presented markedly enhanced results with GPT 3.5 and GPT 4 models. Conversely, the Llama 2 model results were found to be less favorable. For the GPT 3.5 model, over half of the tasks were at least partially or fully completed, where approximately 35% of tasks were fully successful. Remarkably, GPT 4 demonstrated superior performance, with an impressive 74% of tasks flawlessly executed and a further 14% partially completed, only totaling 26% less than full completion. Therefore, by employing the 'Long-String' strategy alongside the GPT 4 model, the most effective prompting strategy with the best overall results in this study could be achieved. We hypothesize that since the Chaining strategy involves multiple sequential requests, an early failure in any of them can result in a domino effect that affects the overall result of the task.

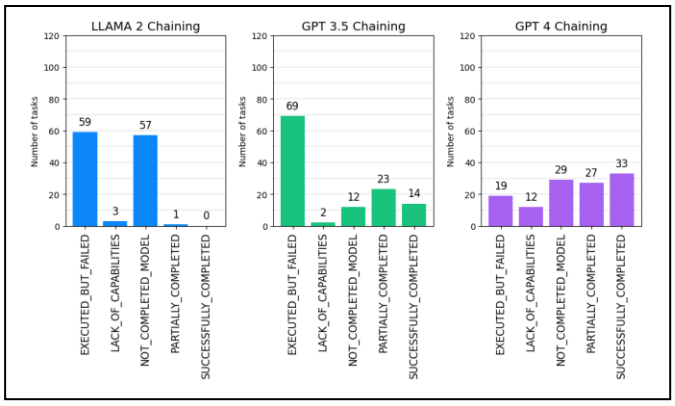

Fig. 3. Chaining Results

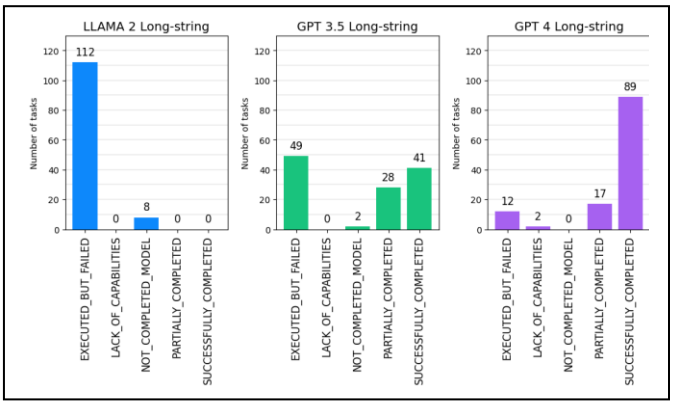

Fig. 4. Long String Results

V. Conclusion

This research is significant in the field of social robotics, demonstrating an approach that has wide applicability for a range of commercial robots designed to interact with humans, interpret commands, and carry out intricate tasks. Central to this approach is the use of natural language, which is the most intuitive way to communicate objectives to a robot. The application of large language models offers considerable advantage as they possess a degree of world understanding allowing them to anticipate actions. Models like GPT 3.5 and GPT 4, trained on a diverse range of code data, are able to generate code consistent with their world understanding, hence facilitating complex tasks execution.

The primary aim of this research was to construct and examine a sophisticated system on a Pepper-like robot, enabling it to autonomously understand and execute instructions delivered through natural language for a multitude of general-purpose tasks. We successfully accomplished the creation of a task processing system and a program that connects natural language instructions to the robot's operations. Additionally, we efficiently constructed an interface functioning as a liaison among the robot commands and high-level behavior. This proved effective, paving the way for large language models, both commercial and open-source, to understand the necessary code to execute the given tasks. Concerning code generation results, our analysis showed that exactly 400 out of 720 tasks generated passed the automatic runtime execution assessment. Among the evaluated LLMs, GPT-4 achieved the highest success rate, followed by GPT-3.5 and Llama2. The success rate for task completion was significantly higher with the Long-String approach compared to the Chaining strategy.

References

[1] Iocchi, L. et al. (2015) 'RoboCup@Home: Analysis and results of evolving competitions for domestic and Service Robots', Artificial Intelligence, 229, pp. 258–281. doi:10.1016/j.artint.2015.08.002.

[2] "Pepper the humanoid and programmable robot," Aldebaran, https://www.aldebaran.com/en/pepper (accessed Aug. 28, 20223).

[3] "Aldebaran documentation what's new in naoqi 2.4.3," Pepper - Developer Guide - Aldebaran 2.4.3.28-r2 documentation, http://doc.aldebaran.com/2-4/family/pepper technical/index dev pepper.html (accessed Aug. 28, 2023)

[4] Brown, Tom, et al. (2020) Language models are few-shot learners. Advances in neural information processing systems, vol. 33, pp. 1877-1901.

[5] Tellex, S., Gopalan, N., Kress-Gazit, H. and Matuszek, C. (2020). Robots that use language. Annual Review of Control, Robotics, and Autonomous Systems, 3, pp. 25-55.

[6] Matamoros, M., Harbusch, K. and Paulus, D. (2019). From commands to goal-based dialogs: a roadmap to achieve natural language interaction in robocup@ home. In RoboCup 2018: Robot World Cup XXII 22 (pp. 217-229). Springer International Publishing.

[7] Jiang, X., Dong, Y., Wang, L., Shang, Q., & Li, G. (2023). Self-planning code generation with large language model. arXiv preprint arXiv:2303.06689.

[8] Ouyang, S., Zhang, J. M., Harman, M, & Wang, M. (2023). LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation. arXiv preprint arXiv:2308.02828.

[9] Vemprala, S., Bonatti, R., Bucker, A., & Kapoor, A. (2023). Chatgpt for robotics: Design principles and model abilities. Microsoft Auton. Syst. Robot. Res, 2, 20.

[10] Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., ... & Garg, A. (2023, May). Progprompt: Generating situated robot task plans using large language models. In 2023 IEEE International Conference on Robotics and Automation (ICRA) (pp. 11523-11530). IEEE.

[11] RoboCupAtHome, "RoboCupAtHome/gpsr command generator: Command Generator for the robocup@home competition/test: General Purpose Service Robot," GitHub, https://github.com/RoboCupAtHome/gpsr command generator (accessed Aug. 2023).

[12] Kyordhel, "Kyordhel/GPSRCmdGen: Command Generators for robocup @home (multiplatform)," GitHub, https://github.com/kyordhel/GPSRCmdGen (accessed Aug. 2023).

[13] Miller, R. B. (1968, December). Response time in man-computer conversational transactions. In Proceedings of the December 9-11, 1968, fall joint computer conference, part I (pp. 267-277).

[14] Shiwa, T., Kanda, T., Imai, M., Ishiguro, H., & Hagita, N. (2009). How quickly should a communication robot respond? Delaying strategies and habituation effects. International Journal of Social Robotics, 1, 141-155.

[15] OpenAI, New models and developer products announced at DevDay, https://openai.com/blog/new-models-and-developer-products-announced-at-devday (accessed Nov. 6, 2023).