# Reducing Latency in LLM-Based Natural Language Commands Processing for Robot Navigation

Diego Pollini[1,2], Bruna V. Guterres[1], Rodrigo S. Guerra[3] and Ricardo B. Grando[1]

*Abstract*—The integration of Large Language Models (LLMs), such as GPT, in industrial robotics enhances operational efficiency and human-robot collaboration. However, the computational complexity and size of these models often provide latency problems in request and response times. This study explores the integration of the ChatGPT natural language model with the Robot Operating System 2 (ROS 2) to mitigate interaction latency and improve robotic system control within a simulated Gazebo environment. We present an architecture that integrates these technologies without requiring a middleware transport platform, detailing how a simulated mobile robot responds to text and voice commands. Experimental results demonstrate that this integration improves execution speed, usability, and accessibility of the human-robot interaction by decreasing the communication latency by 7.01% on average. Such improvements facilitate smoother, real-time robot operations, which are crucial for industrial automation and precision tasks.

## I. INTRODUCTION

The adoption of robotics in industrial applications has expanded significantly, driven by advances in computing hardware, sensor technology, and artificial intelligence (AI). Frameworks such as the Robot Operating System (ROS) have emerged as a foundational standard for industrial robotic applications, offering flexibility and extensive libraries to accelerate development cycles. ROS 2, the latest iteration, further enhances performance, security, and real-time capabilities, overcoming communication reliability and scalability limitations of its predecessor in distributed systems [1]. Currently, advances in AI have given rise to new solutions for improvement in mobile robotics [2]–[5] and sophisticated language models such as the GPT-3 and GPT-4 architectures having been adopted [6], [7], capable of understanding context and generating coherent text. The integration of these models with ROS-based systems enables industrial environments to leverage intuitive natural language commands for enhanced flexibility and adaptability in robotic operations, broadening robotics' applicability through more effective human-robot interactions [8].

The purpose of this article is to explore and demonstrate the integration of ChatGPT with ROS 2. Specifically, we present an industrial environment that enables mobile robot navigation through natural language commands. The input prompt is sent to the GPT model, which interprets the command and generates properly formatted ROS 2 instructions for robot movement without requiring additional middleware. This integration demonstrates a more efficient approach that reduces latency in the command execution cycle, improving Human-Robot Interaction (HRI) for mobile robot navigation tasks. Figure 1 illustrates our proposed system.

Key contributions of this work include:

- A ROS 2-based solution that interprets natural language voice commands and transforms them into control instructions for a mobile robot, without requiring rigid command syntax.
- A direct integration architecture that enables human-robot interaction without intermediate middleware transport platforms such as JSON or Flask.
- A comparative analysis between two GPT model versions, evaluating their performance metrics specifically for Human-Robot Interaction (HRI) navigation tasks with mobile robots.

This paper is organized as follows: Section II reviews related works. Section III describes our methodology, and Section V presents the experimental results. Finally, Section VI summarizes our contributions and discusses future work.

## II. RELATED WORK

Voice-based robotic control systems have long been an active research area. Many solutions rely on speech recognition technologies to interpret verbal commands and translate them into robot actions within industrial settings. The Google Voice Kit allows developers to build voice control interfaces using Google's speech recognition technology. This kit has been used to easily control robots using voice commands [9]. Furthermore, Amazon Alexa has been integrated into various robotic platforms to enable control using voice commands [10].

AI-driven interaction with robotics has led to intuitive user interfaces, which allow more natural and effective interaction with robotic systems. However, these advanced interfaces typically do not use large language models (LLMs) like ChatGPT, but rather rely on task-specific speech recognition and command processing technologies. In this regard, Rasa is an open-source platform for building conversational assistants, including those that can control robots. It uses custom natural language processing (NLP) models to interpret voice and text commands. Wilcock et al. [11] show an implementation where a Furhat social robot uses Rasa conversational AI to access knowledge graphs in a Neo4j graph database. On the other hand, IBM Watson Assistant has been used in service robots to provide interaction capabilities using voice commands. This assistant uses NLP and machine learning technologies to understand and respond to user queries. For example, Di Nuovo

[1]Bruna V. Guterres, Ricardo B. Grando and Diego Pollini are with the Technological University of Uruguay. E-mail: `ricardo.bedin@utec.edu.uy`

[2]Diego Pollini is with National Technological University, Argentina. E-mail: `diego.pollini@frra.utn.edu.ar`

[3]Rodrigo S. Guerra are with the Federal University of Rio Grande. E-mail: `rodrigo.guerra@furg.br`
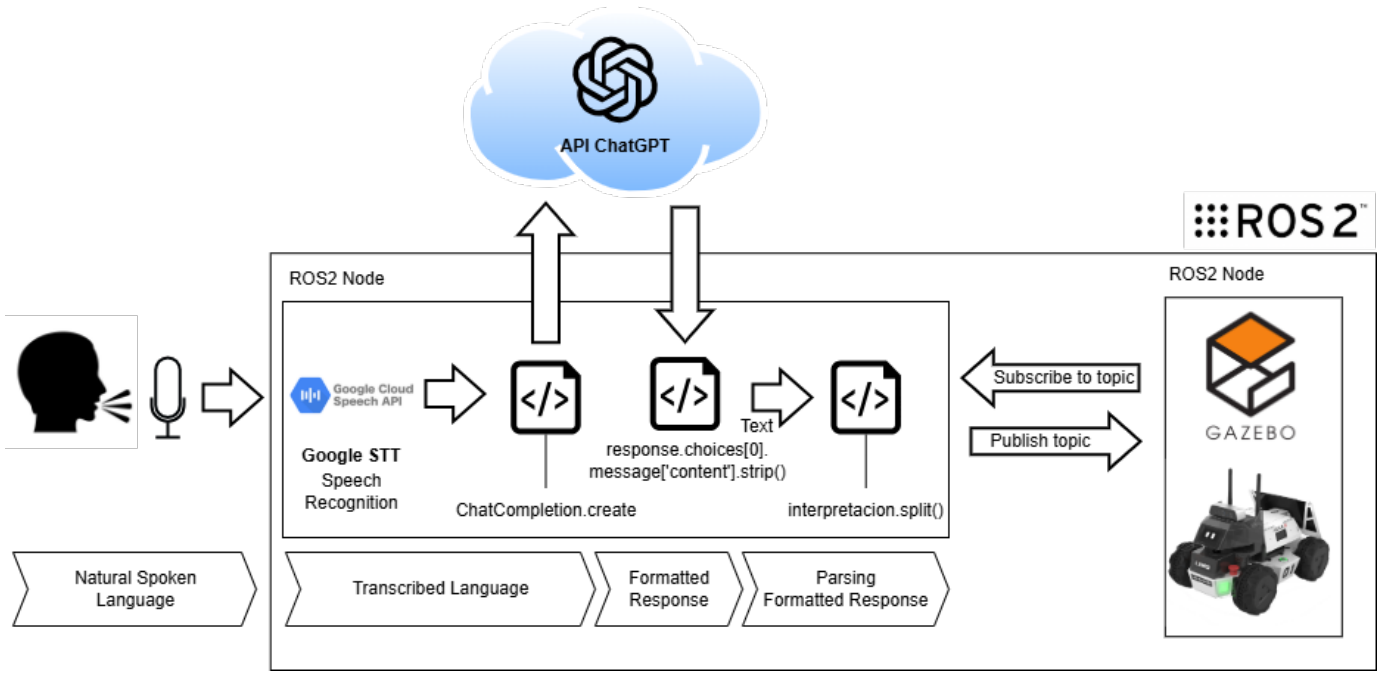
Fig. 1: Our proposed system.

et al. [12] implemented, on a SoftBank Robotics Pepper robot, IBM Watson Assistant to organize the workflow, Watson Text-to-Speech to generate the robot's voice, and Watson Speech-to-Text to perform speech recognition.

Unlike chatbots that only handle text, robotic systems require a deep understanding of real-world physics, environmental context, and the ability to execute physical actions. A generative robot model must possess strong common-sense knowledge and a sophisticated model of the world, as well as be able to interact with the user and the environment to reliably interpret and execute commands. These challenges are beyond the scope of traditional linguistic models, as they must not only understand the meaning of text but also translate intent into a logical sequence of physical actions. He et al. [13] present the concept of RobotGPT as a multimodal approach to implementing the seven types of robotic intelligence that not only seek to incorporate various forms of intelligence, such as logical-mathematical and linguistic but also addresses the challenge of operating in a real-world environment, which is inherently complex and dynamic. By leveraging GPT's natural language processing capabilities, the author presents RobotGPT with the potential to improve human-robot interaction by facilitating a deeper understanding of context and enabling more accurate and adaptive responses to the situations at hand.

Yu et al. [14] introduces an innovative approach that combines LLMs with a low-level model prediction controller, MuJoCo MPC, to define reward functions in robotics. A system was developed that enables robots to perform complex manipulation and locomotion tasks, such as pushing objects and grabbing apples, using natural language instructions. The system translates these instructions into structured descriptions

of robotic movements, improving stability and task-solving capability. It was implemented on a quadruped robot and a dexterous manipulator, achieving 90% success in executing 17 tasks, compared to 50% for a primitive skill-based approach. Furthermore, integration with ROS enables efficient communication between language models and robot controllers, facilitating real-time interaction management and adaptation of robotic movements. A specific regularization term was used to encourage stable movements in real-world environments, demonstrating the system's effectiveness in transferring from simulation to reality.

Along the same lines of integrating large language models (LLM) such as ChatGPT with robotic systems, Koubaa et al. [15] presents ROSGPT, a package developed for ROS2 that integrates ChatGPT with ROS2-based robotic systems. The implemented solution allows robots to interpret and execute natural language commands, significantly improving user experience and robot functionality. The integration of LLM with robotic systems, as demonstrated in Koubaa et al. [15], represents a significant advance by enabling more sophisticated natural language understanding and generation, which improves human-robot interaction and expands the control and adaptability capabilities of robots.

Unlike previous works, our study investigates a direct API request model to enhance latency performance and minimize processing overhead, which is crucial in real-time industrial applications. The proposed system eliminates middleware dependency, processing responses from ChatGPT in structured lists without the need for JSON library, and improves execution speed essential for effective human-robot collaboration in industrial automation. In turn, the effectiveness of the GPT

3.5 and GPT 4.0 models is evaluated when interpreting the same unstructured commands from natural human speech and converting them into structured data that can be analyzed programmatically.

## III. METHODOLOGY

The proposed architecture allows human-robot interaction based on the Agilex Limo mobile robot using natural spoken language. It enables the robot to perform linear movements or turns. Verbal commands are translated into text through the Whisper speech-to-text module and sent to ChatGPT using the OpenAI API. ChapGPT interprets the request and returns a formatted response that is parsed to produce ROS-compatible commands, allowing the robot to perform the action initially requested.

The script that manages the interaction between natural language and the control of the robot is implemented using the Python programming language. This choice is based on the versatility and wide adoption of Python in the robotics community, especially in the context of ROS 2. Python not only provides a clear and accessible syntax, but also has an extensive ecosystem of libraries and tools that facilitate the development and integration of complex systems. Thus, the ROS2 node that was developed facilitates the efficient integration of the communication logic with the OpenAI API and the Whisper speech-to-text module, also allowing the accurate generation of ROS commands for robot control. The hardware supporting the generated ROS2 node is a PC with sufficient capabilities to run the simulation on Gazebo and its microphone is used to receive the user's voice commands.

The system is started by configuring the OpenAI API key, which allows the use of the GPT model to interpret natural language commands. Communications with the robot is done with a publisher to send motion commands to the *cmd_vel* topic and a subscriber to receive data from the robot's IMU (Inertial Measurement Unit), thus updating the robot's yaw angle based on the received quaternion orientation. A **record_audio** function was made to record sound from a microphone and save the audio to a temporary WAV file, starting the recording when sound is detected and ending it after a period of silence. Subsequently, a **transcribe_audio** function uses the Whisper model to convert the audio file into text. Once the transcribed text is obtained, it is sent to the OpenAI API using a function that extracts and formats the commands necessary for controlling the robot. Depending on whether the command is of the move or rotate type, the corresponding messages are sent to the robot to perform the desired action. Finally, the main loop of the system captures voice commands, interprets them, and executes the corresponding actions in the robot. The system is safely closed upon receiving an interrupt signal, ensuring proper termination of the ROS 2 node and resource cleanup. Below is the pseudocode of the implemented script:

```
#OpenAI API Setup
    Read tOPENAI_API_KEY
    Set the API key

#Speech Recognition Setup
    Create a 'recognizer' object of '
        speech_recognition'
    Create a 'mic' object of 'speech_recognition'

#Defining the 'RobotController' class
    '__init__' method
        Initialize the ROS 2 node
        Create a publisher for the 'cmd_vel'
        Create a subscriber for the 'imu/data'
        Initialize the 'yaw' variable to 0.0
    'imu_callback' method
        Update the 'yaw' variable from IMU
    'get_yaw_from_quaternion' method
        Calculate the 'yaw' angle
        Calculate the direction and time
        Publish a message to move the robot
    'rotate' method
        Calculate the target angle and direction of
            rotation
        Publish to rotate the robot in the specified
            direction and angular speed
        Wait until the robot turns to the desired
            angle
        Stop the rotation

#Definition of the 'recognize_vocal_order' function
    Capture Audio
    Transcribe audio to text using Google's service
        If transcribed, return the text
        If error, return 'None'

#Definition of the 'interpret_sentence' function
    Create a message to the model with data to be
        interpreted
    Send the message to the model and receive the
        response
    Parse the response to move or rotate
    Return a tuple with the command and parameters

#Definition of the 'play_message' function
    Randomly choose a message from a list of options
    Use a TTS model to play the chosen message aloud

#Defining the 'main' function
    Initialize the ROS 2 system
    Create an instance of the robot controller
    Run an infinite loop to:
        Call 'recognize_vocal_command' to get a
            command from the user
        If a valid, interpret the command with '
            interpret_phrase'
            Execute command (move or rotate)
        If not valid
            Print an error message
    Handle program interruption
    Shut down the ROS 2 system
```

### A. Human Robot Communication

In the development of voice command robot control systems, the effective integration of speech recognition and synthesis is essential to achieve a natural, intuitive and efficient interaction. In this context, two key functions have been implemented. The first function is responsible for outputting text messages through a text-to-speech (TTS) engine, providing audible feedback to the user. This function not only informs the user about the current state of the system, such as starting to listen for commands, but also adds a layer of dynamism to the interaction

by randomly selecting predefined messages. This approach ensures that communication is varied and less monotonous, improving the user experience. To achieve this, the function randomly selects a message from a predefined list of possible messages using the random module. Once the message is selected, the pyttsx3 TTS engine is used to convert the text into speech.

```
def message_play():
    # Choose a random message
    message = random.choice(possible messages)
    engine.say(message)
    engine.runAndWait()
```

The second function developed plays, captures, and transcribes the audio received through the microphone, using Google's speech recognition service to convert the audio into text. This function is initiated by setting the system microphone as the audio input source. Once the audio has been captured, the function proceeds to transcribe it into text using Google's speech recognition service. The captured audio is sent then to the Google service for processing. This allows the system to understand and process voice commands in the specified language. The robustness of the function is ensured by efficient error handling. If the speech recognition service is unable to understand the audio due to problems such as excessive noise or unintelligible articulation, an exception is thrown. In such cases, the function prints a message indicating that the audio could not be understood and returns None. Also, if there is a problem connecting to the speech recognition service, such as network failures or problems in the Google service, an exception is thrown. The function handles this error by printing a message with the reason for the problem and also returns *None*. When transcription is successful, the function returns the recognized text, which is the textual representation of the captured vocal command.

### B. Voice Recognition

Google speech recognition offers a convenient way to transcribe audio without the need to set up API keys, which is very useful for prototypes and small projects. In addition, it offers very good accuracy in transcribing speech to text, supporting multiple languages and accents. The user's speaking language and the language in which the transcription is intended must be configured. However, the es=ES model perfectly supports the English language. There are other models available in the library used that require an internet connection and an API key and support the Spanish language that were not tested in this research work. The main reason was due to the limitations of computational resources compared to cloud processing, which uses servers with specialized and scalable hardware for handling deep learning models.

### C. Prompt Structure

Once the transcript of the user's verbal command is obtained, a systematic process is initiated to interpret and execute that command using the ChatGPT model. The transcript of the verbal command is converted into a structured query directed to the ChatGPT language model through the OpenAI API. A function was created to organize this query into two distinct roles within the model: *system role* and the *user role*. Properly formulating the prompts for the *system role* and the *user role* in the interaction with ChatGPT is crucial to obtaining accurate and structured responses that the script can efficiently parse and process. The *system role* prompt sets the context and general instructions for the model, defining the role it should play and the rules it should follow. This prompt acts as a guide that directs ChatGPT to interpret and format responses consistently. On the other hand, the *role user* prompt presents the user's specific request, in this case, the verbal instruction transformed into text. The clarity and precision in both prompts ensure that ChatGPT generates a structured response that the script can analyze to generate the ROS 2 language commands that set the robot in motion.

At the *rol system* prompt, the model is set to act as a wizard for interpreting robot commands, providing clear instructions on converting magnitudes to standard units (meters and m/s) and assigning default values when the information is not available. On the other hand, the *rol user* prompt specifies the task of interpreting the given sentence and extracting the values needed to control the robot, with precise instructions on the expected response format, without quotation marks or a full stop. In this way, the model returns responses in a consistent and usable format. If the different ways the GPT model responds to different user questions are not considered, the script will not be able to publish the command to the corresponding ROS 2 topic that sets the mobile robot in motion. Once ChatGPT has processed the query and generated a structured response, the script continues with the processing stage of this response, within the same *interpret_phrase* function, to convert it into executable commands in the ROS 2 ecosystem. The function starts by extracting the relevant content from ChatGPT's response, which is located in the content attribute of the response object. The text is cleaned by removing extra whitespace and ensuring that it is in a processable format.

```
interpretation = response.choices[0].message['
    content'].strip()
```

*response.choices[0]* accesses the first (and in this case, only) item in the choice list in the ChatGPT response. In the context of the OpenAI API, *choices* is a list of responses generated by the model. Each object within *choices* contains several fields. The *message* field is one of these fields, and is a dictionary containing the structure of the generated response. The *content* field of the *message* dictionary stores the text string generated by the model. On the other hand, *.strip()* is a string method that removes any whitespace at the beginning and end of the text string. This ensures that the response is clean and free of unwanted characters before and after the main content. The *interpretation* list is then split word by word using the *.split* command. This step is critical to identify the individual components of the instruction based on the expected structure of the text generated by ChatGPT, such as command type, direction, distance, angle, and velocities.

```
        words = interpretation.split()
```

If the first word in the list is "move", the command is interpreted as moving the robot longitudinally forward or backward. The values for direction, meters, and speed are extracted. The meter's value is adjusted based on whether the direction is "backward", making the value negative if necessary.

```
if words[0] == "move":
    direction = words[1]
    meters = float(words[2])
    speed = float(words[6])
    if direction == "back":
        meters = -meters
    return ('move', meters, speed)
```

Conversely, if the first word is "rotate", the command is interpreted as being to rotate the robot. The values for direction, angle, and angular velocity are extracted. The angle is set to the specified direction (clockwise or counterclockwise).

```
elif words[0] == "rotate":
    direction = words[3]
    angle = float(words[4])
    angular_velocity = float(words[9])
    if direction == "clockwise":
        angle = -angle
    return ('rotate', angle, angular_velocity)
```

The *interpret_phrase* function returns a tuple composed of a string containing the type of action (move or rotate), a float containing the meters or the angle to rotate, and another float containing the value of linear speed or angular speed as appropriate.

## IV. COMMANDS IN THE ROBOT

If the action to be performed is to move, the script calls the move method of the controller object (an instance of the *RobotController* class). The move method receives two arguments: the distance in meters and the speed in meters per second, which are extracted from the tuple. The move method then creates a message of type *Twist* and publishes it to the *cmd_vel* topic to tell the robot to move in a straight line at the specified speed for the time calculated based on the distance. If the order was to move forward, the direction variable gets the value 1, which means the speed is positive, and if, on the contrary, the order was to move backward, the direction variable gets the value -1, which means the linear speed in x will now be negative. If the action to be performed is to rotate, the script calls the rotate method of the controller object. The rotate method receives two arguments: the angle in degrees and the angular velocity in radians per second, which are extracted from the tuple. The rotate method also creates a message of type *Twist*, but this message configures the robot's z component to rotate in the specified direction.

## V. EXPERIMENTAL RESULTS

Using the time library for Python, the duration of the verbal command and the response time of the Google speech recognition model are determined to find the requisition latency. To evaluate the effectiveness of the speech recognition system, a sample of 20 verbal commands was randonmly taken, which

were generated in a silent environment and recorded in real time using the built-in microphone of a laptop. These verbal commands were transcribed and the transcription result is compared with the original verbal command to determine the success rate. To measure the response time of the OpenAI API, the *rosgpt.py* script of the ROSGPT package is modified by including the *time* library. Within the *askGPT()* method of the *ROSGPTProxy()* class, the time just before sending the request to ChatGPT is captured in a variable, then in another variable, the time just after receiving the response from ChatGPT is captured. Finally, in another variable, the difference is stored and the latency value is printed on the screen.

The latency time of the ROSGPT model is compared with the model of the developed solution with GPT3.5 Turbo and GPT4.0. It can be observed that for 16 out of 20 cases, the latency time of the developed solution is lower, as can be seen in Table I. The average latency time is reduced 7.01 %, considering an average latency of 1.2865 s with our solution with GPT3.5 Turbo to 1.1804 s with ROSGPT [16]. With both GPT3.5 Turbo and GPT4.0, 100% of success rate was also achieved, as seen in Table I, while a total of 14 successes was achieved with ROSGPT [16]. In this regard, it is worth remembering that the solution developed in this research work used a direct request methodology to the OpenAI API, unlike the ROSGPT model, which uses a Flask server. On the other hand, the ROSGPT model incorrectly interpreted 13 of the orders, which led to failures in the execution of movements. Table I details these misinterpreted outputs and from them the following observations are made:

1) The ROSGPT model does not recognize speeds or lengths expressed in units other than m/s or m, respectively. Therefore, the user must manually convert before executing the order. Unlike this, in the developed solution, the *rol system* prompt and the *rol user* prompt that hosts the user's request are designed in such a way that ChatGPT performs the unit conversion if necessary.
2) The ROSGPT model does not recognize right turn or left turn indications as the developed solution model does.

## VI. CONCLUSIONS

The present work investigated the integration of ChatGPT with ROS2 for enhanced human-robot interaction in industrial settings by reducing latency and improving command interpretation. The implemented solution demonstrates an effective ability to transcribe, interpret, and execute voice commands in a robotic environment, providing a solid foundation for controlling a mobile robot within industrial settings using natural language, with a high success rate in correctly executing commands. An STT model has been successfully integrated with a GPT language model to interpret commands and generate instructions in the ROS 2 environment. The research has provided a detailed evaluation of the efficiency of Google's STT model and the GPT-3.5 Turbo and GPT-4.0 models, measuring response times and effectiveness in converting verbal commands to executable commands for ROS 2. The GPT-3.5 model outperformed the other investigated solutions in response

| N° | Latency GPT3.5 Turbo [s] | Success GPT3.5 Turbo | Latency GPT4.0 [s] | Success GPT4.0 | Latency [16] [s] | Success [16] |
|---|---|---|---|---|---|---|
| 1 | 1.05 | OK | 1.83 | OK | 1.15 | OK |
| 2 | 0.97 | OK | 1.71 | OK | 1.66 | OK |
| 3 | 0.91 | OK | 1.69 | OK | 1.19 | OK |
| 4 | 1.08 | OK | 1.26 | OK | 1.42 | OK |
| 5 | 0.77 | OK | 1.81 | OK | 1.45 | OK |
| 6 | 0.88 | OK | 1.75 | OK | 1.32 | OK |
| 7 | 0.61 | OK | 1.25 | OK | 1.27 | FAIL |
| 8 | 3.62 | OK | 1.98 | OK | 1.01 | FAIL |
| 9 | 1.03 | OK | 1.69 | OK | 1.31 | OK |
| 10 | 1.09 | OK | 1.99 | OK | 1.13 | FAIL |
| 11 | 1.46 | OK | 1.93 | OK | 1.46 | OK |
| 12 | 0.72 | OK | 1.37 | OK | 1.37 | OK |
| 13 | 0.97 | OK | 1.67 | OK | 1.61 | OK |
| 14 | 0.72 | OK | 2.06 | OK | 1.37 | OK |
| 15 | 0.92 | OK | 1.49 | OK | 1.29 | OK |
| 16 | 0.83 | OK | 1.99 | OK | 1.37 | OK |
| 17 | 1.36 | OK | 2.02 | OK | 1.26 | FAIL |
| 18 | 0.88 | OK | 2.53 | OK | 0.99 | FAIL |
| 19 | 0.79 | OK | 1.57 | OK | 1.03 | OK |
| 20 | 1.27 | OK | 1.67 | OK | 1.07 | FAIL |
| **Average Latency Time Ours with GPT3.5 Turbo:** 1.18 | | | | | | |
| **Average Latency Time Ours with GPT4.0:** 1.76 | | | | | | |
| **Average Latency Time [16]:** 1.28 | | | | | | |
| **Total number of success cases Ours with GPT3.5 Turbo:** 20 | | | | | | |
| **Total number of success cases Ours with GPT4.0:** 20 | | | | | | |
| **Total number of success cases [16]:** 14 | | | | | | |

TABLE I: Performance Comparision between approaches.

latency and command interpretation accuracy, enabling more responsive and adaptable industrial robotic systems. This study supports automated workflows by enabling natural language interactions with industrial robots, minimizing manual interventions, and improving intuitive control. Future works include exploring faster speech-to-text models and leveraging vision-based systems for better situational awareness.

Alternatives could be explored to improve the speed of the STT model and the GPT model in order to achieve greater fluidity in the execution of the user's verbal orders. The use of other STT models in the cloud could be evaluated, which, a priori, could have lower latency than Google's STT, such as Deepgram or Microsoft Azure Speech-to-Text. Another option could be to implement a local model of OpenAI's Whisper. Although this would require having adequate computing infrastructure to support it and avoid processing delays (GPU, TPU), it would allow the simultaneous transcription and translation of the user's verbal command, which would enhance the performance

of the system. Undoubtedly, the availability of better computing infrastructure would also improve the GPT model's response time. However, the use of Prompt Engineering techniques and tuning OpenAI API parameters such as temperature or the maximum number of tokens can help further improve latency.

## REFERENCES

[1] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *Proceedings of the 13th international conference on embedded software*, 2016, pp. 1–10.

[2] M. Soori, B. Arezoo, and R. Dastres, "Artificial intelligence, machine learning and deep learning in advanced robotics, a review," *Cognitive Robotics*, vol. 3, pp. 54–70, 2023.

[3] J. C. de Jesus, V. A. Kich, A. H. Kolling, R. B. Grando, M. A. d. S. L. Cuadros, and D. F. T. Gamarra, "Soft actor-critic for navigation of mobile robots," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 2, p. 31, 2021.

[4] V. A. Kich, J. A. Bottega, R. Steinmetz, R. B. Grando, A. Yorozu, and A. Ohya, "Advancing behavior generation in mobile robotics through high-fidelity procedural simulations," in *2024 33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN)*. IEEE, 2024, pp. 43–48.

[5] J. A. Bottega, V. A. Kich, J. C. d. Jesus, R. Steinmetz, A. H. Kolling, R. B. Grando, R. d. S. Guerra, and D. F. T. Gamarra, "Jubileo: an immersive simulation framework for social robot design," *Journal of Intelligent & Robotic Systems*, vol. 109, no. 4, p. 91, 2023.

[6] N. Wake, A. Kanehira, K. Sasabuchi, J. Takamatsu, and K. Ikeuchi, "Chatgpt empowered long-step robot control in various environments: A case application," *ArXiv*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2204.01691

[7] 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," *ArXiv*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2209.07753

[8] R. Mao, G. Chen, X. Zhang, F. Guerin, and E. Cambria, "Gpteval: A survey on assessments of chatgpt and gpt-4," *arXiv preprint arXiv:2308.12488*, 2023.

[9] N. Tiwari, S. Shinde, S. P. Koli, Akash, and R. Nikhare, "Voice based smart robotic arm," *The PCE Journal of Computer Engineering*, vol. 9, no. 1, pp. 88–90, 2021.

[10] J. M. Vicente-Samper, I. Campos, B. Sanz Worrel, Akash, A. Rodríguez, J. M. Oñate Tevar, and J. M. Sabater-Navarro, "Ejemplo de integración de alexa con un robot ur," *XL Jornadas de Automática: libro de actas, Ferrol, 4-6 de septiembre de 2019*, pp. 360–365, 2019. [Online]. Available: https://doi.org/10.17979/spudc.9788497497169.360

[11] G. Wilcock and K. Jokinen, "Conversational ai and knowledge graphs for social robot interaction," *17th ACM/IEEE International Conference on Human-Robot Interaction (HRI 2022) , Sapporo , Japan*, pp. 1090–1094, 2022. [Online]. Available: https://doi.org/10.1109/HRI53351.2022.9889583

[12] A. Di Nuovo, S. Varrasi, D. Conti, J. Bamsforth, A. Lucas, A. Soranzo, and J. McNamara, "Usability evaluation of a robotic system for cognitive testing," *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI), Daegu, Korea (South)*, pp. 588–589, 2022.

[13] H. He, "Robotgpt: From chatgpt to robot intelligence," *TechRxiv*, 2023.

[14] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. Gonzalez Arenas, H.-T. Lewis Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia, "Language to rewards for robotic skill synthesis," *ArXiv*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.08647

[15] A. Koubaa, "Rosgpt: Next-generation human-robot interaction with chatgpt and ros," *Preprints*, 2023. [Online]. Available: https://doi.org/10.20944/preprints202304.0827.v2

[16] ——, "Rosgpt: Chatgpt interface for ros2 for human-robot interaction," 2023, repositorio en GitHub. [Online]. Available: https://github.com/aniskoubaa/rosgpt