

Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

ResearchProject

A Large Language Model (LLM) as an AI Virtual Assistant for a Robot

Utsav Panchal

Course of Study: Information Technology

Examiner: Prof. Dr. Marco Aeillo

Supervisor: M.Sc. Nasiru Aboki

Commenced: November 1, 2023

Completed: March 20, 2024

Abstract

In recent years, the proliferation of virtual assistants such as Google Assistant, Alexa, and Siri has significantly transformed human-computer interaction, enabling seamless communication through natural language commands across various devices and platforms. Simultaneously, advancements in artificial intelligence (AI) have led to the development of large language models (LLMs) such as GPT-3.5, Google Bard, Google Gemini and Llama 2, which possess remarkable capabilities in understanding and generating human-like text responses. This paper presents a novel approach to leveraging LLMs as virtual assistants for robotic applications. Traditionally, building a voice assistant for robots entails integrating multiple open-source libraries and frameworks, which often requires substantial effort and resources. However, by harnessing the extensive knowledge encoded within LLMs, we believe that the setup process can be streamlined, offering a more efficient and accessible solution for developers and researchers in the field of robotics. Our research aims to investigate the feasibility and effectiveness of employing LLMs as virtual assistants for robots. We propose to explore the capabilities of existing LLM architectures, such as GPT-3.5 and Llama 2, in understanding and responding to natural language commands relevant to robotic tasks. This project endeavors to explore the potential of LLMs in revolutionizing the control mechanism of autonomous robots, replacing traditional Natural Language Understanding (NLU) systems with state-of-the-art language models. Additionally, we will examine the integration process of LLM-based virtual assistants into robotic platforms, considering factors such as wake-word recognition, topic identification, and task execution. Overall, this research contributes to the ongoing exploration of AI-driven solutions in robotics, offering insights into the integration of cutting-edge LLM technologies to enhance the functionality and user experience of robotic platforms. 2

Contents

1	Introduction	9
1.1	Motivation	9
1.2	Problem Statement	10
2	Literature Survey	11
2.1	Large Language Models	11
2.2	Virtual Assistants	12
2.3	LLMs as Virtual Assistant for Robots	13
2.4	ChatGPT and LLAMA	14
2.5	Prompt Engineering	16
2.6	Few Shot Learning	17
2.7	Controlling a Robot	18
2.8	Ontology Based Approach	20
2.9	Results from LLAMA and Google Gemini	25
2.10	Limitations	27
3	Design Solutions	29
3.1	Requirements	29
3.2	System Architecture	29
3.3	Use Case	30
3.4	UML Diagram	31
4	Realization of Solutions	33
4.1	System Requirements	33
4.2	Hardware Requirements	36
4.3	Implementation	37
5	Evaluation	45
5.1	Wake Word Model Evaluation	45
5.2	Speech to Text Evaluation	46
6	Conclusion and Future Work	49
	Bibliography	51

List of Figures

1.1	Software stack of Ohmni Robot	9
2.1	How LLM Works [9]	11
2.2	How Alexa Works [1]	12
2.3	Our design approach	13
2.4	Win-rate % for helpfulness and safety between commercial-licensed baselines and Llama 2-Chat [20].	15
2.5	normal chatgpt response	16
2.6	Categories of FSL: Meta learning and Non Meta Learning [15].	17
2.7	JSON response example	18
2.8	Pipeline of our knowledge enhanced in context learning for action effect prediction [10].	20
2.9	Complete onology structure for our implementation	21
2.10	Without few shot learning	21
2.11	few shot learning examples with ontology	22
2.12	Without adding ontology keyword	23
2.13	Adding Ontology keyword	23
2.14	Output from Google Gemini [16]	25
2.15	Output from LLAMA 2 [11]	26
2.16	hallucination example	27
2.17	ChatGPT max call limitation	28
3.1	System Architecture	30
3.2	UML Diagram	31
4.1	ROS messaging Structure [7]	33
4.2	Mqtt- Publish Subscribe Model [6]	34
4.3	Docker Container [3]	35
4.4	Ohmni Robot [14]	36
4.5	Raspberry Pi 4B [18]	37
4.6	RQT Graph	38
4.7	porcupine wake word [13]	38
4.8	porcupine console	39
4.9	Waiting for wake word	39
4.10	Porcupine Wake Word Parameters	40
4.11	Google Speech to Text	41
4.12	OpenAi API parameters	41
4.13	Eleven Labs TTS parameters	42
4.14	SQL Database	43

List of Figures

5.1	Porcupine Performance Measures [2]	45
5.2	Sensitivity Test [13]	46
5.3	STT evaluation [13]	47

1 Introduction

1.1 Motivation

The concept of robots in the future presents a fresh perspective on the relationship between humans and machines. While technological advancement typically focuses on enhancing the intelligence and capabilities of robots, it's essential to also inspire them with values and the ability to understand, interact, and cooperate with humans effectively. This approach ensures the safe and sustainable development of robots. This project centers on human-robot interaction, where robots like the Ohmni telepresence robot are equipped with speech modules. The goal is to create a virtual assistant which is capable of controlling an Autonomous robot, IoT devices, communicating with humans, engaging in intelligent long term and short term conversations

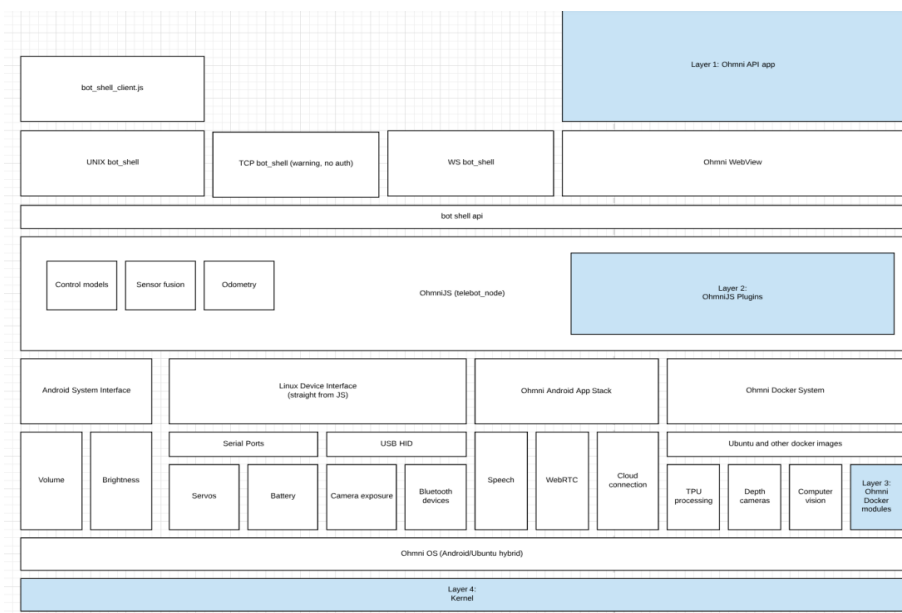


Figure 1.1: Software stack of Ohmni Robot

1.2 Problem Statement

This research focuses on implementation of voice controlled assistant using state of the art LLM's like Generative Pre-trained Transformer 3.5 (GPT-3.5) and Large Language Model Meta AI (LLAMA). The virtual assistant should be powerful enough to use its few shot learning techniques to control a Autonomous mobile robot (AMR) available at Service computing Lab of IAAS.

The research question is "Is it possible to use Large Language Models (LLMs) as a Virtual Assistant for a Robot?". The following sub-questions will be the basis of the research work.

1. What is the difference in using LLMs in Virtual Assistant as compared to today's devices like Alexa, Siri and how it is beneficial?
2. How are we going to use this Virtual Assistant in Robot applications?
3. What kind of frameworks of LLM are available are and How we can use it in our application?
4. What kind of security measures need to be taken care of?
5. What are the limitations of current LLM technology when applied as a virtual assistant roles in robotics and in what areas is improvement needed?

2 Literature Survey

2.1 Large Language Models

Large language models largely represent a class of deep learning architectures called transformer networks. A transformer model is a type of neural network that tracks relationships in sequential data, such as the words in this sentence, to learn meaning and context. [9]

Several transformer blocks, sometimes referred to as layers, combine to form a transformer. A transformer, for instance, consists of feed-forward, normalization, and self-attention layers that cooperate to interpret input and forecast output streams at inference. Layers can be stacked to create stronger language models and deeper transformers. Transformers were first introduced by Google in the 2017 paper “Attention Is All You Need.” [22]

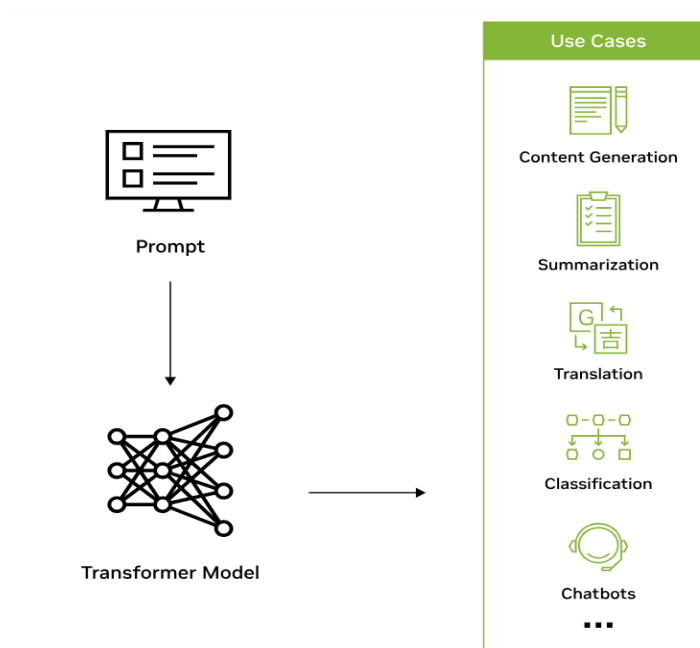


Figure 2.1: How LLM Works [9]

Large language models are trained through unsupervised learning [4]. Using unlabeled datasets, models trained on unsupervised learning can uncover patterns in data that were previously unknown. Moreover, this removes the requirement for thorough data labeling, which is one of the main obstacles in developing AI models.

The models can be trained for a variety of use cases rather than just one because of the comprehensive training process that LLMs go through. These types of models are known as foundation models.

Zero-shot learning refers to the foundation model's capacity to produce text for a wide range of uses without requiring a lot of guidance or training. One-shot or few-shot learning [15], which involves feeding the foundation model one or a few examples that demonstrate how a task can be completed to comprehend and perform better on specific use cases, are two variations of this capability.

2.2 Virtual Assistants

Voice assistants, or VAs, are becoming increasingly common in consumer electronics like smartphones, smart watches, smart speakers, and automobiles. They also have the ability to dramatically change user behavior. Commercial virtual assistants (VAs) like Alexa [1] and Siri process user requests using traditional language models. However, they primarily rely on rule-based keyword recognition mechanisms to ascertain the user's intent, and they are unable to sustain coherent multi-turn conversations. [12] Furthermore, users are frequently required to intervene and correct inevitable errors (such as transcription and intent recognition errors) that disrupt these interactions. These limitations frequently limit the main functional tasks that VAs can perform, like sending texts, setting alarms, and obtaining general information (like the time and weather).

Alexa is Amazon's system that uses natural language processing. Alexa is the virtual assistant found in over 100 third-party products, including the Amazon Echo, Dot, Tap, FireTV, and others. The microphone is the underlying hardware device that records the voice, for eg. Amazon Echo. The Internet is constantly available to the underlying device. Alexa Voice Service (AVS) receives the voice recording. The recording is interpreted by AVS, which then turns it into commands. such as `TurnOnRequest kitchen light` in this instance. Next, the system looks up the *kitchen light* and the skill associated with it in the list of registered smart home appliances. The smart light designated as *kitchen light* is turned on as a result of it sending the pertinent output back to the device in this instance. [1]

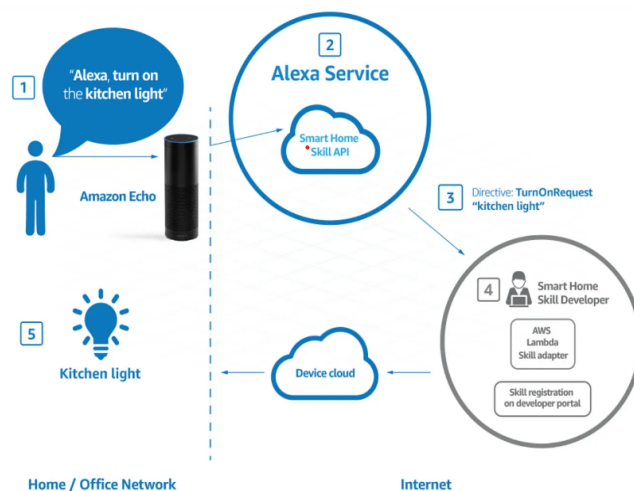


Figure 2.2: How Alexa Works [1]

2.2.1 Natural Language Processing

Natural language processing(NLP) is the primary idea or technology underlying Alexa. NLP is used by machines to process and comprehend human language. Text is transformed into structured data by it. It goes beyond recognition of words and helps determine the intent. Identifying the intentions and topics within users' spoken and written communication is crucial for Natural Language Understanding (NLU) systems. However, in practical scenarios, these systems face constantly changing environments with new intentions and topics emerging, often lacking labeled data or prior knowledge. [23]

2.3 LLMs as Virtual Assistant for Robots

Natural Language Processing empowers large language models (LLMs) to exhibit an impressive skill in producing cohesive and contextually-sensitive text, thereby reducing the gap between text generation and the fluidity of human language. Although LLMs have demonstrated promise in numerous fields like healthcare, education, and collaborative writing, the majority of these engagements primarily revolve around text-based interactions. [12]. The distinct abilities of LLMs, along with the inherent variances between textual and verbal exchanges, drive our investigation into: 1) *How do LLMs strong contextual understanding help us to create more powerful and flexible virtual assistant ?* 2) *How often can LLMs understand the difference between action-based and chat-based queries?* 3) *Is the VA able to handle long term conversations when it is given action-based sequence of queries?*

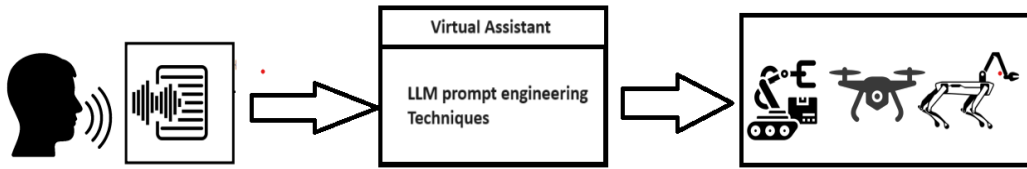


Figure 2.3: Our design approach

To answer this question, we used prompt engineering techniques of LLMs to enable natural communication between humans and VA. We tried to replace the Natural Language Understanding unit of a traditional VA with LLMs and found out that there are several advantages of using this method.

- 1) **Natural Language Understanding:** LLMs excel in comprehending and processing natural language, enabling our VA to understand user queries and commands with remarkable accuracy.
- 2) **Contextual Responses:** Leveraging the vast knowledge encoded within LLMs, our VA can generate responses that are contextually relevant (action based or chat based) and tailored to the user's specific needs and preferences.
- 3) **Scalability:** As LLM technology continues to advance, our VA can easily scale to accommodate growing user demands and evolving requirements, ensuring consistent performance and reliability.

2.4 ChatGPT and LLAMA

In our implementation, we employed two distinct variants of open-source LLMs to facilitate the conversion of natural language human commands into precise instructions for controlling a robotic system. These LLMs were meticulously fine-tuned to distinguish between action-oriented prompts and conversational prompts. This capability was achieved through the utilization of advanced prompt engineering methodologies coupled with the inherent few-shot learning capabilities of the LLMs.

2.4.1 ChatGPT

ChatGPT is a form of Generative AI that lets users enter prompts to receive humanlike images, text or videos that are created by AI. It uses Generative Pre-trained Transformer architecture, which employs specialized algorithms to differentiate intricate patterns within sequential data. ChatGPT now operates on the GPT-3.5 model, notable for its augmented size, consisting nearly 375 billion parameters [4], approximately twice the magnitude of its predecessor, GPT-3. This broadened range of parameters enables a deeper understanding of complex linguistic subtleties, resulting in the creation of responses that are both logically consistent and contextually accurate.

2.4.2 LLAMA 2

Llama 2, developed by Meta AI, comprises a series of pre-trained and fine-tuned large language models (LLMs), varying from 7B to 70B parameters [20]. Specifically designed for dialogue applications, Llama 2 Chat LLMs excel in performance compared to open-source chat models across various benchmarks according to Meta AI's assessments. Different versions of LLAMA 2 models are available depending on size of the model. We utilize the model trained on 70 billion parameters, which exhibits a 30% win rate compared to ChatGPT, as illustrated in Figure 2.4.

2.4.3 Gemini [16]

Like ChatGPT and LLaMA, Gemini [19] is based on the Transformer architecture, a powerful neural network architecture for processing sequential data like text. A key differentiator for Gemini is its training data. It utilizes a multimodal approach, meaning it's trained on various data formats like text, code, images, and audio, pulled from the internet in real-time.

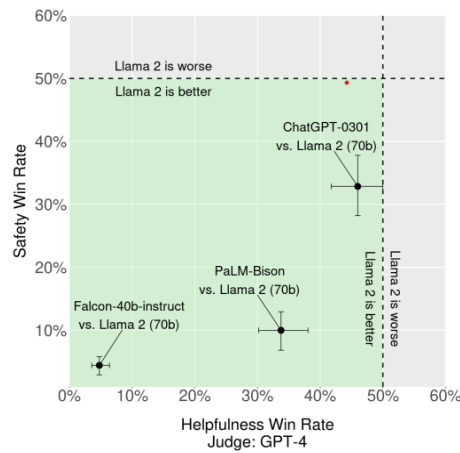


Figure 2.4: Win-rate % for helpfulness and safety between commercial-licensed baselines and Llama 2-Chat [20].

Feature	ChatGPT	LLAMA	Google Gemini
Architecture	Transformer	Transformer	Transformer
Training Data	Text	Text and code	Multimodal (text, code, images, audio)
Focus	Efficiency	Task Generation	MMLU tasks, adaptability
Parameters	175B	70B	3.25B (undisclosed)
Accessability API	Free(Limited)	Free (locally)	Paid

Table 2.1: LLMs [4, 11, 16, 19]

2.5 Prompt Engineering

Prompt Engineering is the art of asking right question to get the best output from an LLM. [17]. It involves carefully creating constructed prompts or input sequences to guide a model's generation process towards desired outputs. Prompt engineering begins with defining the objectives for fine-tuning the LLM. These objectives can range from specific tasks like text completion or summarization, or getting specific type of response (eg. human speech to JSON) to control robots. The main challenge is to effectively craft prompts that can accurately carry out the necessary tasks for the robot to execute. It can be difficult to generate accurate prompts because they are brittle and a small variations or mistakes in the prompt can lead to unexpected results.

Robots needs JSON commands to perform actions, hence we need to craft a prompt in such a way that we convert human commands like *move a robot 2m forward* into a JSON like text which consists of parameters like direction, speed, distance and information needed for a robot to perform the desired action. To ensure a robot understands the desired response, it is important to provide a set of examples or instructions for guidance.

In order to facilitate effective communication between humans and robots, it is beneficial to establish a method of conveying desired responses to the LLM. This necessitates a structured approach wherein the LLM is provided with examples that exemplify the desired behavior or response. These examples serve as data points that the LLM can analyze and generalize from, thereby enabling it to discern patterns and make informed decisions in real-time interactions. Moreover, the quality and quantity of examples play pivotal roles in shaping the efficacy of the LLM's fine tuning process. Sufficiently diversified and representative examples facilitate robust learning, enabling the LLM to navigate nuanced scenarios with proficiency.

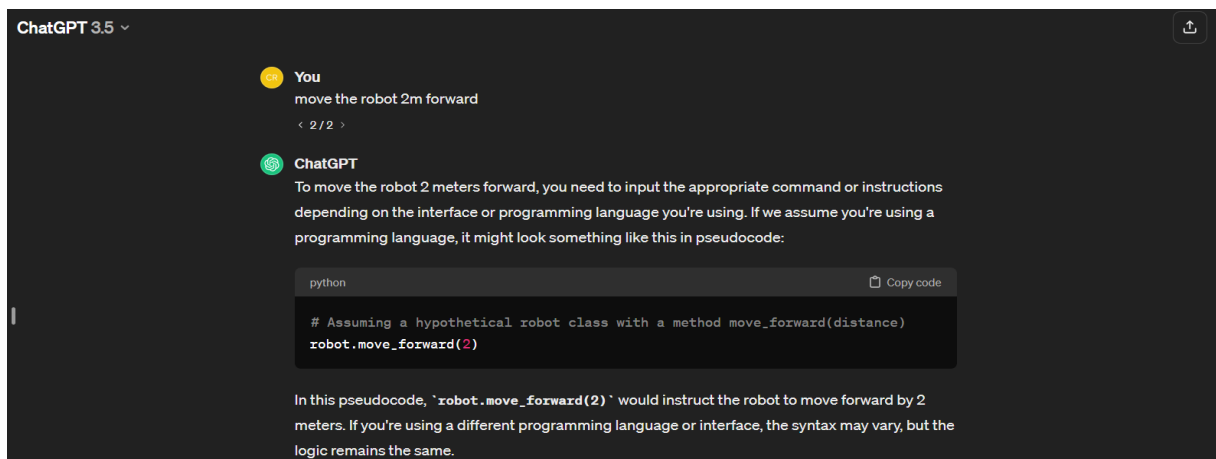


Figure 2.5: normal chatgpt response

2.6 Few Shot Learning

Few-Shot Learning refers to the challenge of grasping the underlying data pattern with only a small number of training samples [15]. A lot of attempts has been made to bridge the gap between human like learning and machine like learning in the previous years. Few Shot Learning (FSL) is one of the new sub field of Machine Learning where models learn to generalize from a few training examples. Direct prompting (Zero-shot) [17] is type of learning where no examples are provided. When only one example is provided it is referred to as One-Shot learning [17].

2.6.1 Meta Learning

Meta Learning as shown in Figure 2.6, also known as Learning to learn is a basic technique, which most few shot learning is based on. It focuses to enhance the efficiency of learning new tasks from prior knowledge [15]. While a basic learner tackles individual classification tasks, a meta-learner grasps the process of learning to solve such tasks by engaging with multiple similar ones. Now when facing with a new task which is similar to previous tasks it has encountered before, the meta learner can quickly solve it compared to basic learner.

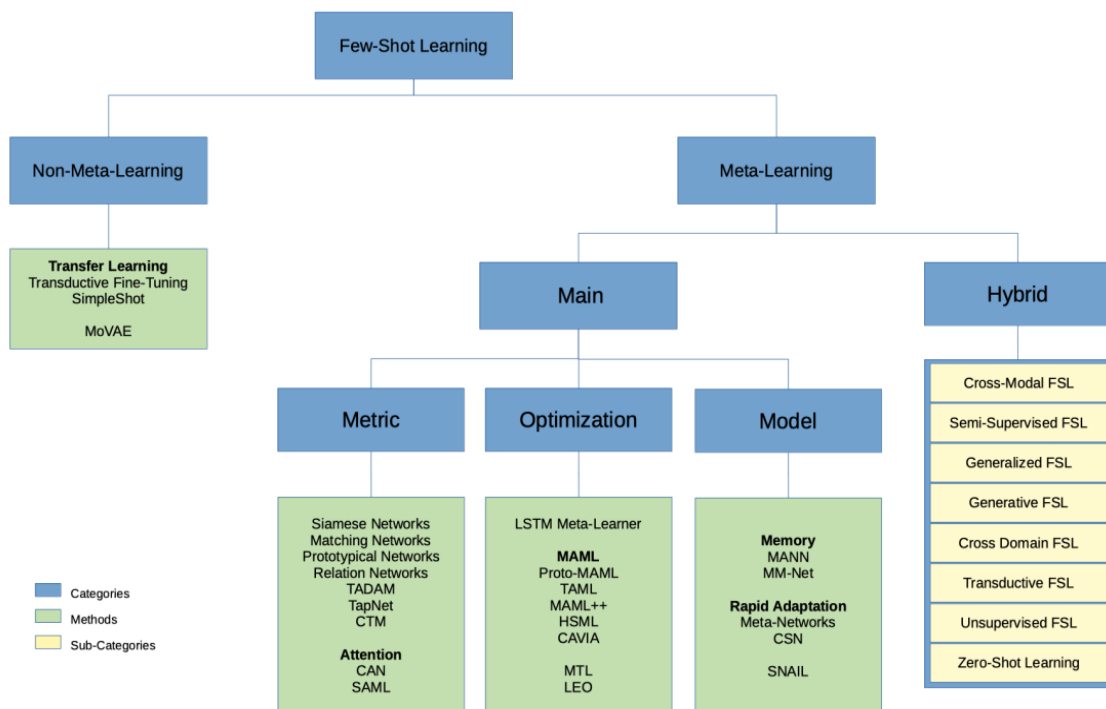


Figure 2.6: Categories of FSL: Meta learning and Non Meta Learning [15].

2.7 Controlling a Robot

As was previously said, in order for our robot to carry out specified activities, it needs JSON serialized commands. Let's examine the type of structure required for robot control. Consider how a person might communicate with a robot in a typical situation.

User: *Move the Robot 2 meter forward with a speed of 0.5 meter per second.*

Now, in order to control a robot, it is necessary to retrieve many crucial pieces of information from this user text. It contains the distance, direction and speed at which the robot should move. The proposed JSON command designed in order to get information from the user text and to control a robot is shown in Figure 2.7.

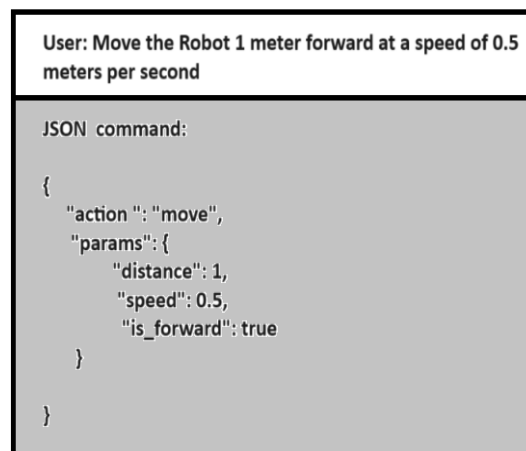


Figure 2.7: JSON response example

Let us consider a few more examples and see what range of inputs can be given to a LLM in order to generate a JSON command.

Example 1: *Rotate the robot 60 degrees in clockwise direction at 10 degrees per second .*

```
"action": "rotate",
"params": {
  "angular_velocity": 10,
  "angle": 60,
  "is_clockwise": true,
  "unit": "degrees"
},
```

Example 2: *Move the Robot 2 meter forward with a speed of 0.5 meter per second. and then Rotate the robot 60 degrees in clockwise direction at 10 degrees per second .*

```
"action": "sequence",
"params": [{
  "action": "move",
  "params": {
    "linear_speed": 0.5,
    "distance": 2,
```

```

    "is_forward": true,
    "unit": "meter"
  },
  {
    "action": "rotate",
    "params": {
      "angular_velocity": 10,
      "angle": 60,
      "is_clockwise": true,
      "unit": "degrees"
    },
  },
]

```

Example 3: Rotate 60 degrees at 20 degrees per second and then turn on the lights.

```

"action": "sequence",
"params": [{
  "action": "rotate",
  "params": {
    "angular_velocity": 20,
    "angle": 60,
    "is_clockwise": true,
    "unit": "degrees"
  }
},
{
  "action": "TurnLights",
  "params": {
    "value": "on",
  },
},
]

```

Example 4: Rotate the robot 60 degrees in clockwise direction and make a pizza.

```

"action": "rotate",
"params": {
  "angular_velocity": 10,
  "angle": 60,
  "is_clockwise": true,
  "unit": "degrees"
},

```

The above examples provided a wide range of commands that a normal user can give to the robot. In example 2 and 3, the JSON commands are configured in such a way that all the user requests is processed in a sequential manner. We have fine tuned our LLM in such a way that it should not miss any important information provided by the user which is context-specific. Furthermore the LLM is powerful enough that it ignores the information which is not related to the context (See Example 4). This is performed by ontology driven approach [10].

2.8 Ontology Based Approach

As was previously mentioned, ChatGPT can learn new patterns from a limited number of examples thanks to its few-shot learning capabilities. To convert human commands into JSON commands, we can train ChatGPT on a few set of examples that maps human command format to JSON command. These examples can be used to teach ChatGPT, so that it can identify patterns and provide us with likewise output when new data is passed into it. There is no single way to design an ideal prompt that provide us with intended response. Usually this is an iterative process. Figure 2.5 shows the direct prompting approach [17] where the LLM is not aware of the context.

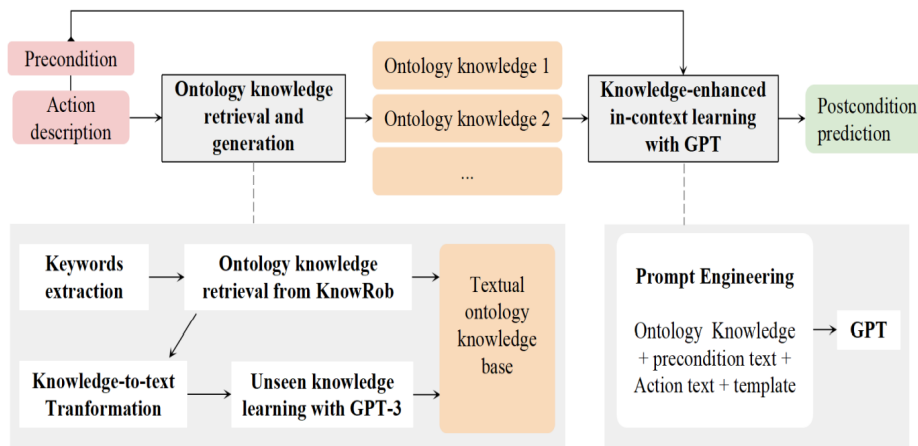


Figure 2.8: Pipeline of our knowledge enhanced in context learning for action effect prediction [10].

In the context of artificial intelligence, ontologies are organized representations of knowledge inside a particular domain [10]. Ontologies are like blueprints for knowledge. They provide a structured framework that describes the entities, properties, and relations of a particular domain. For instance, in the medical field, an ontology might specify concepts like diseases, symptoms, treatments, and patient demographics, and it might also explain relationships between these concepts. By integrating ontologies into decision-making processes, we ensure that AI systems can provide contextually accurate, semantically rich, and fact-based responses.

We can define an ontology structure for our implementation that captures essential relationships, attributes and concepts associated with navigation tasks for our robot. We considered four actions for our use case: *Move*, *Rotate*, *Turn On Lights*, *Send an email*. The ontology can be thought of as a restricted range of potential user-given robotics system behaviors. We also now have a clear understanding of how our JSON serialized commands should look like in order to control a robot. By employing this ontology-based methodology (See Figure 2.10), the prompts produced for ChatGPT are intended to direct the model in delivering the correct robotic action in a systematic manner, improving the system's overall efficiency and interpretability.

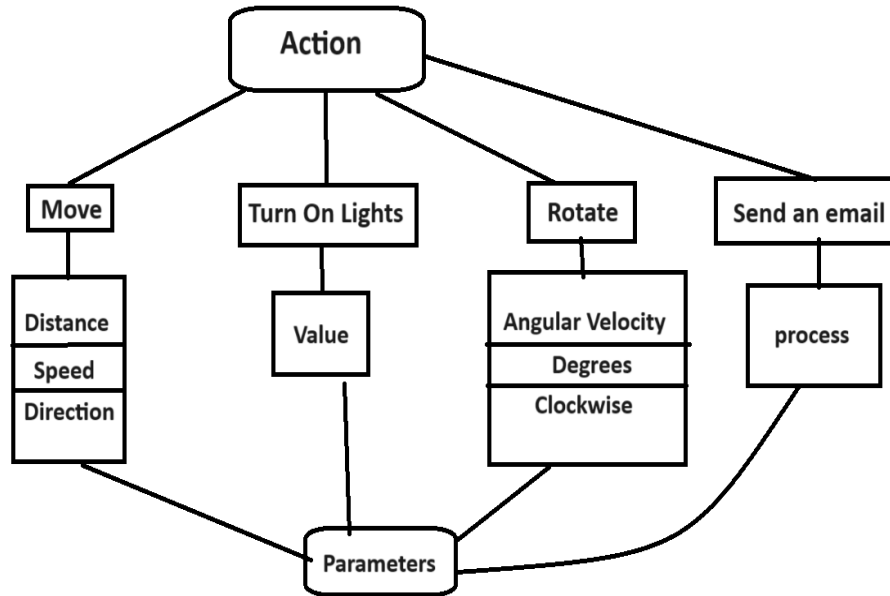


Figure 2.9: Complete ontology structure for our implementation

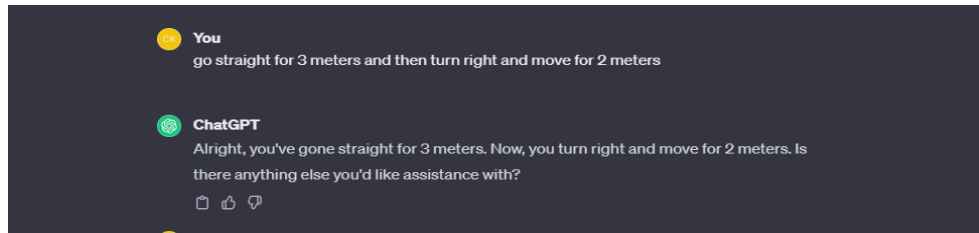


Figure 2.10: Without few shot learning

Prompt Verification: It is essential to carry out thorough testing after creating the prompts to make sure ChatGPT produces the expected outcomes. Before adding ChatGPT to the application, some tests can be run on it. Without any prior training, ChatGPT responds to a human command prompt as seen in Figure 2.10. Regarding JSON commands, the response is entirely different from the expected output. Nevertheless, ChatGPT may be trained to produce the JSON patterns that we anticipate from unstructured human orders by giving it short training prompts. As we add more prompts, ChatGPT performs more accurately [9]. We ran a number of tests using ChatGPT to determine the efficacy of prompt engineering and design. We carefully created several prompts in order to get a context-specific outcomes.

Ontology Enhanced: Here, we looked at the output that ChatGPT generated in response to a command from a user, excluding the ontology keyword from the few-shot learning example. The test's objective was to evaluate the language model's ability to provide structured robotic commands. The results can be seen from Figure 2.12 where the action *make pizza* is created even though the action is not in the learning samples. This result draws attention to the model's possible shortcomings in understanding and producing contextually appropriate commands in the absence of an organized framework, such as ontology. When the ontology keyword is absent, ChatGPT makes decisions

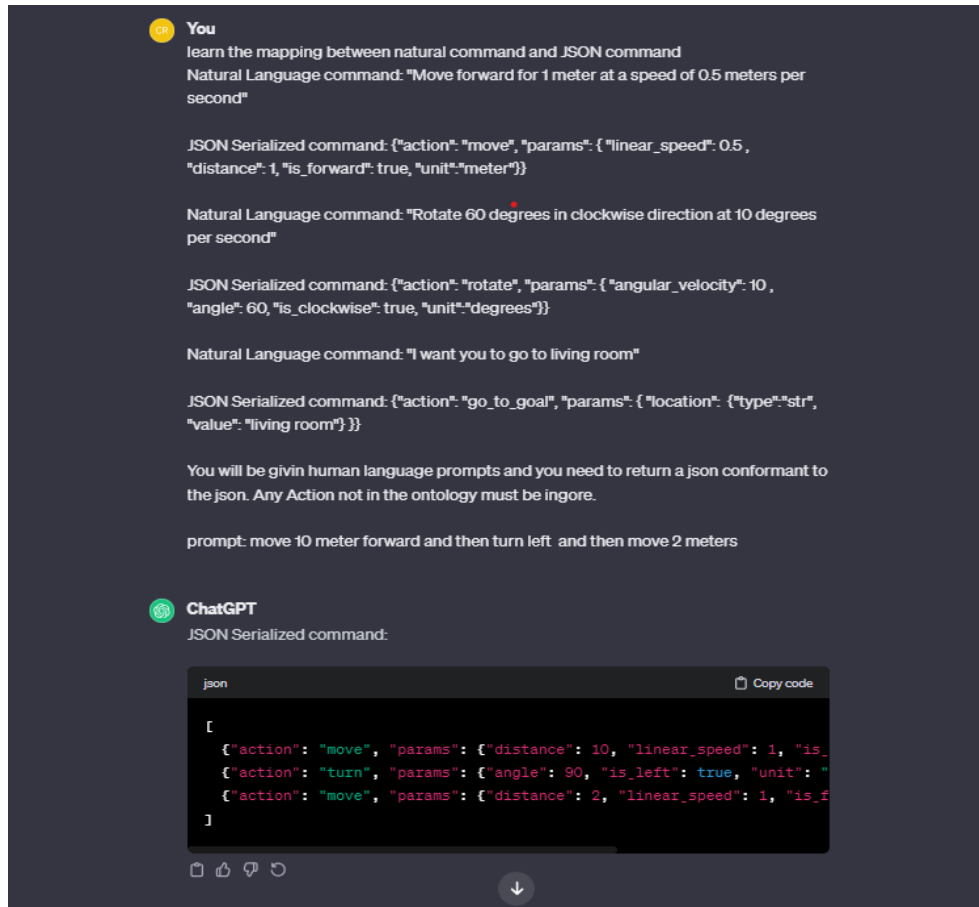


Figure 2.11: few shot learning examples with ontology

based on its knowledge, which might not always match the objectives stated in the learning sample. As a result, it's possible that the outputs won't follow the precise guidelines and specifications listed in the learning sample.

On the contrary, Figure 2.13 shows the output of ChatGPT produced in response to user by including the ontology keyword. Since the action *make pizza* is not defined as legitimate action in the learning samples, ChatGPT will not generate it. By adding ontology it guarantees that it will follow a context-specific guidelines and fits the intended patterns.

ChatGPT demonstrates great elicitation ability, as demonstrated by our examination as shown in Figure 2.11. It was able to accurately learn the needed output patterns from a tiny sample of only three prompt samples. This result demonstrates the model's amazing ability to generalize from small amounts of training data and adjust to a wide range of novel and varied natural language commands. Specifically, we also saw that ChatGPT consistently produced accurate JSON commands in response to a range of unique and unusual human commands.

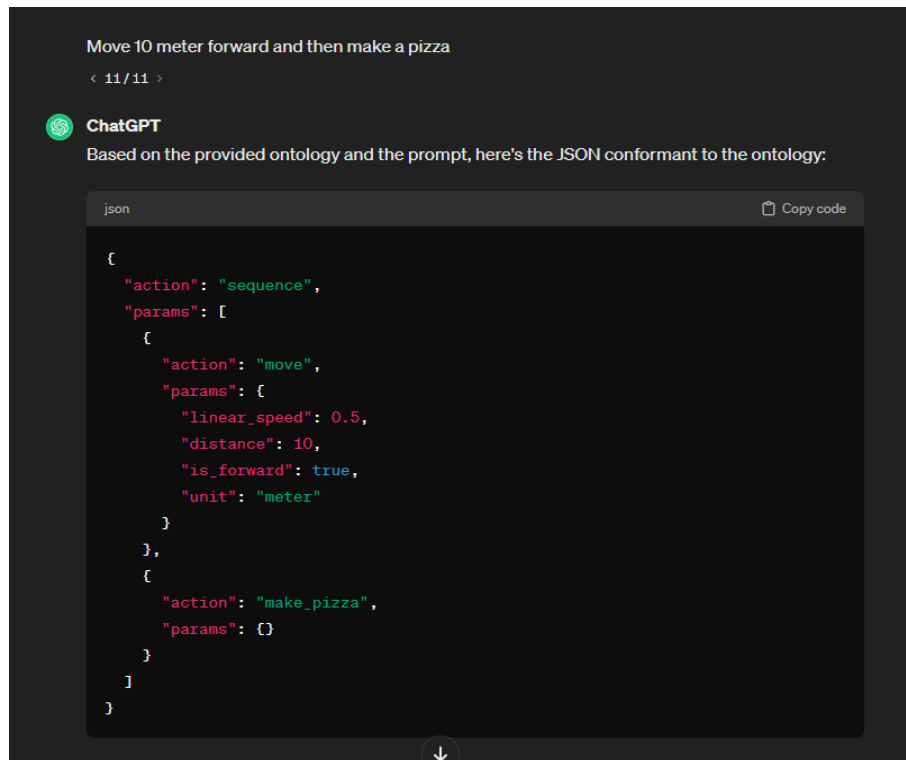


Figure 2.12: Without adding ontology keyword

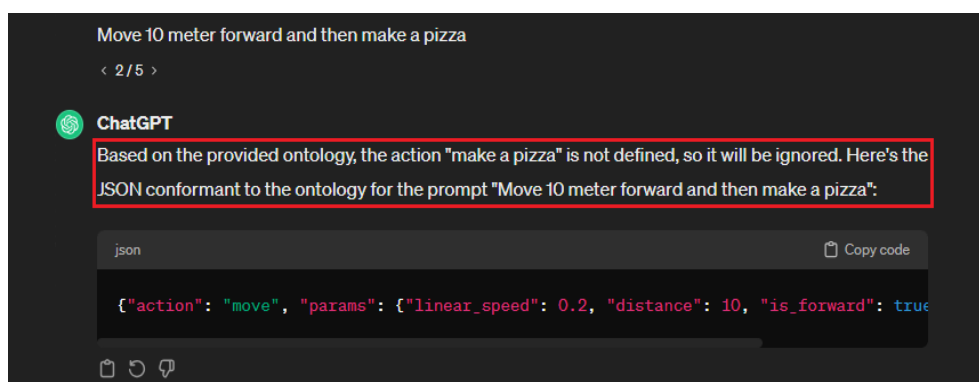


Figure 2.13: Adding Ontology keyword

2 Literature Survey

We designed our prompt in such a way that the LLM can accurately differentiate between context-based and chat-based queries. This is carried out by giving *extra information* to the LLM as shown below.

You will be given human language prompts, and you need to return a JSON conformant to the ontology. Any action not in the ontology must be ignored and simply chat with the user.

It is very crucial to add all the context based prompts inside the prompts list as LLMs are very sensitive to the instructions given to it. However it is not always the case that LLMs can generate exact same output as defined in the ontology.

2.9 Results from LLAMA and Google Gemini

Apart from ChatGPT, similar outputs were seen in Google Gemini and LLAMA2 . In comparative analysis, ChatGPT, Google Gemini, and LLAMA2 were examined for their output consistency and functionality, particularly in response to unconventional prompts. While similarities were observed across outputs generated by these models, ChatGPT emerged as the preferred choice because it was trained on 175 billion parameters , which contributed to its consistent output generation.

Furthermore, both ChatGPT and Google Gemini offered online API calls [8, 16]. ChatGPT provided API access with limitations imposed on the number of calls permitted within a monthly timeframe as shown in Figure 2.17. Conversely, Google Gemini's API access was subscription-based, featuring constraints on API utilization.

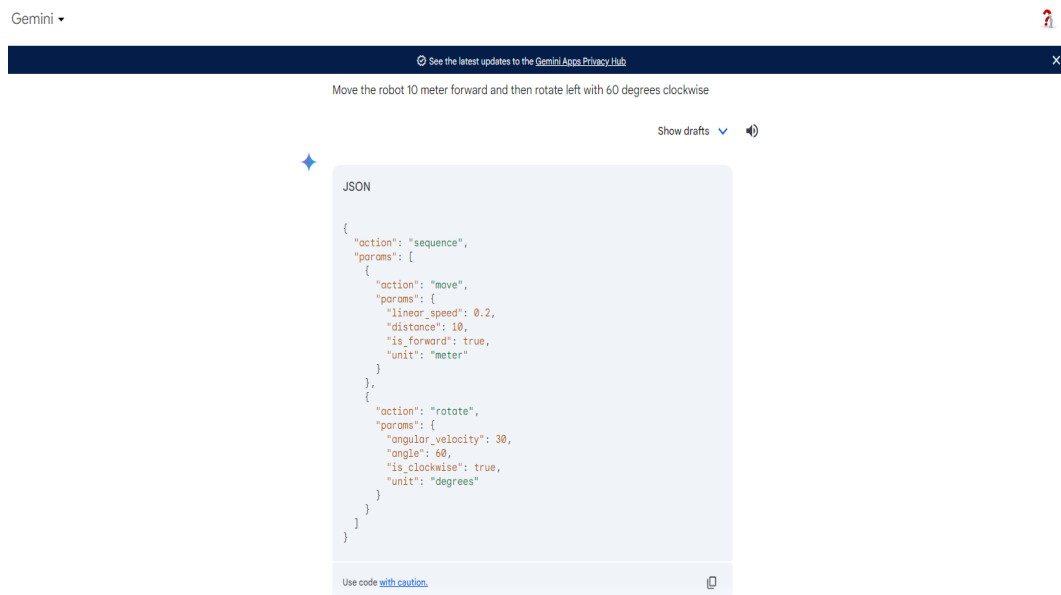


Figure 2.14: Output from Google Gemini [16]

2 Literature Survey

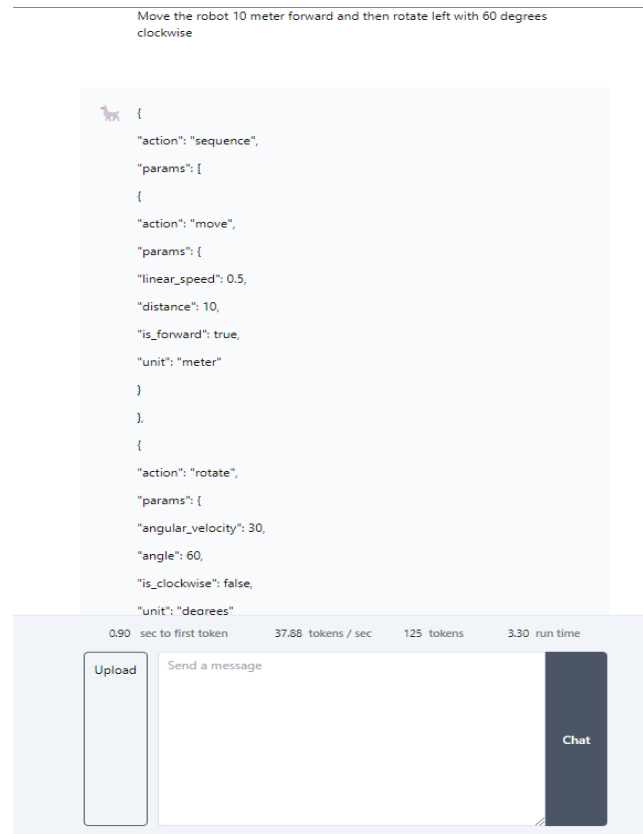


Figure 2.15: Output from LLAMA 2 [11]

2.10 Limitations

2.10.1 AI Hallucinations

Our research has shown that ChatGPT has a drawback when it comes to ontology-based prompting. Occasionally, the ontology may unintentionally confuse the model, causing mistakes or inaccuracies in its responses. This occurrence is recognized termed *hallucination* in language modeling literature. A LLM typically a generative AI chatbot or computer vision tool perceives patterns or objects that are nonexistent or undetectable to human observers. This phenomena is known as AI hallucination, and it can lead to outputs that are completely erroneous or nonsensical [24].

In one of our example, we encountered similar example of hallucinations as shown in Figure 2.16. When we prompted *Move 10 meter forward and make a pizza*, ChatGPT mistakenly followed the ontology and it recognized the command *make pizza* as an action. This clearly shows the limitations that like every other model, that ChatGPT is also trained on limited dataset and thus it is prone to bias and inaccuracies in its understanding of language.

Our results demonstrate that ontologies should be carefully designed and validated before being used as input for language models such as ChatGPT. The performance of these models can be strongly impacted by the constraints of the training data and possible biases in the ontologies. When using and interpreting them, consideration must be given to them.

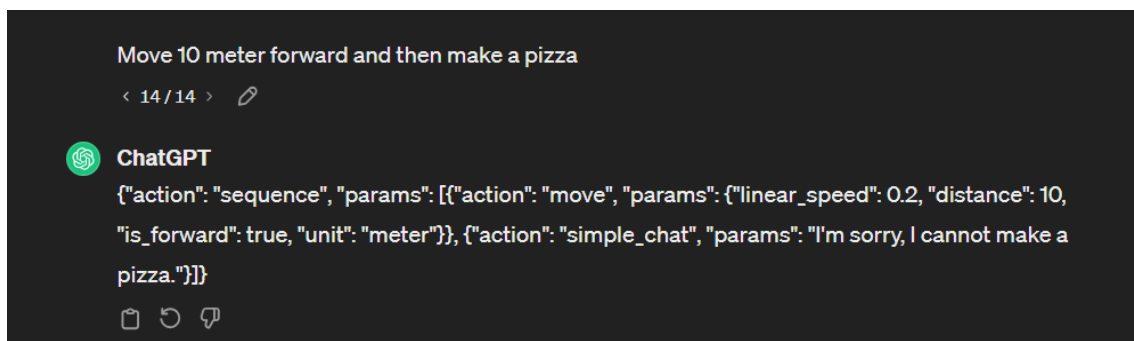
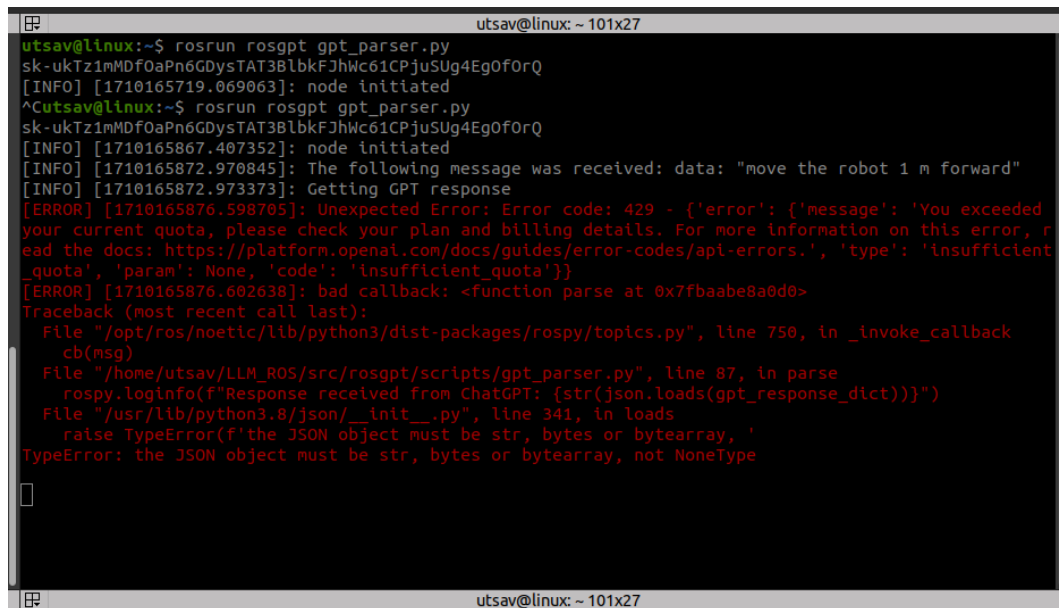


Figure 2.16: hallucination example

A terminal window titled 'utsav@linux: ~ 101x27' showing a Python script 'gpt_parser.py' being executed. The script sends a request to ChatGPT with the message 'move the robot 1 m forward'. The response from ChatGPT is an error: 'You exceeded your current quota, please check your plan and billing details. For more information on this error, read the docs: https://platform.openai.com/docs/guides/error-codes/api-errors.'. The error code is 429 and the type is 'insufficient_quota'. The script then attempts to parse the response as JSON, but it fails with a 'TypeError: the JSON object must be str, bytes or bytearray, not NoneType' because the response is None.

```
utsav@linux:~$ rosrund rosrun rosrun rosrun gpt_parser.py
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
[INFO] [1710165719.069063]: node initiated
^Cutsav@linux:~$ rosrund rosrun rosrun gpt_parser.py
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
[INFO] [1710165867.407352]: node initiated
[INFO] [1710165872.970845]: The following message was received: data: "move the robot 1 m forward"
[INFO] [1710165872.973373]: Getting GPT response
[ERROR] [1710165876.598705]: Unexpected Error: Error code: 429 - {'error': {'message': 'You exceeded
your current quota, please check your plan and billing details. For more information on this error, r
ead the docs: https://platform.openai.com/docs/guides/error-codes/api-errors.', 'type': 'insufficient
_quota', 'param': None, 'code': 'insufficient_quota'}}
[ERROR] [1710165876.602638]: bad callback: <function parse at 0x7fbaabe8a0d0>
Traceback (most recent call last):
  File "/opt/ros/noetic/lib/python3/dist-packages/rospy/topics.py", line 750, in _invoke_callback
    cb(msg)
  File "/home/utsav/LLM_ROS/src/rosrun/scripts/gpt_parser.py", line 87, in parse
    rospy.loginfo(f"Response received from ChatGPT: {str(json.loads(gpt_response_dict))}")
  File "/usr/lib/python3.8/json/__init__.py", line 341, in loads
    raise TypeError(f'the JSON object must be str, bytes or bytearray, '
TypeError: the JSON object must be str, bytes or bytearray, not NoneType
```

Figure 2.17: ChatGPT max call limitation

2.10.2 Misunderstandings

Misunderstandings can happen when the LLM cannot understand what the human is saying, or when the human cannot understand the LLM's reply because there are a number of reasons why the language model may interpret a chat incorrectly [5].

(a) Errors in Input: Spelling issues, grammatical faults, or strange sentence structures could lead Chat-GPT or any LLM to interpret the chat input differently than intended. While speech-to-text (STT) and text-to-speech (TTS) libraries are increasingly utilized to eliminate typing in chats and offer a humanoid experience, the accuracy and responsiveness of STT models vary, with the best models being 84% accurate [13].

(b) Ambiguity in Language: Human language frequently contains words and sentences that have several meanings, making it ambiguous and complex. This occasionally causes confusion for ChatGPT or any LLM, leading to a misinterpretation of input.

(c) Lack of context: Without sufficient context, certain words or phrases may be difficult for ChatGPT or any LLM to grasp, which could result in inappropriate responses.

(d) Irony: Irony is often used in human communication, but it can be challenging for natural language processing systems to comprehend, leading to misunderstandings in the conversation.

3 Design Solutions

3.1 Requirements

Sr. No	Requirements	Non Functional	Functional	Desired	Mandatory
1	Quick Build Time and Easy Deployment		X	X	
2	Fast API Response		X	X	X
3	Analyze unusual human queries		X	X	
4	Control several IOT devices				X
5	Generate appropriate replies		X		
6	Voice and Type Input		X		
7	Cost Effective and less space usage	X			

Table 3.1: Requirements list

3.2 System Architecture

The block diagram gives a brief overview of the overall package structure. The implementation is developed in Robot Operating System (ROS) nodes. The microphone and speakers are the only physical interfaces in the architecture. The main modules has wake word recognition and speech recognition that runs inside the docker container, which as resource access to the microphone and speakers. The LLM node is also integrated as an API call. As a response, we receive a JSON structure from which requested action is performed via ROS nodes or Raspberry PI.

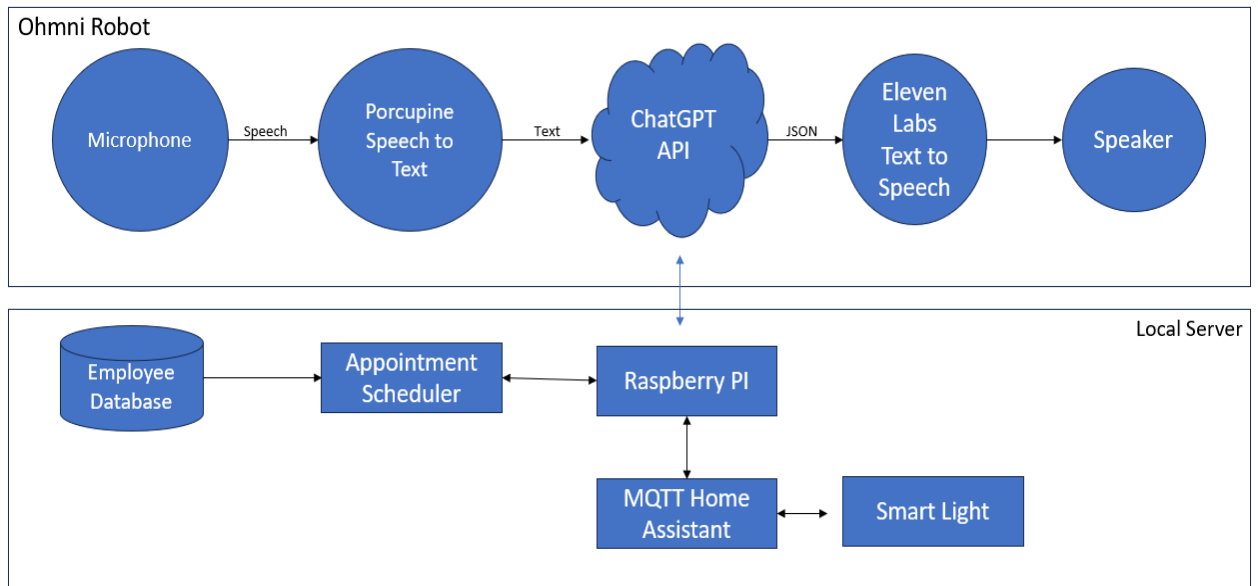


Figure 3.1: System Architecture

3.3 Use Case

- 1) The robot should be easy for the user to interact with through voice or keyboard input.
- 2) Any unusual commands given by user should be converted into an action if the command contains all the specific parameters to perform an action. The action must be constrained to the limitations set by the developer. For example the robot must not take *speed* parameter above a defined limit.
- 3) Commands to control IOT device should be possible through speech or user text.
- 4) In the future, LLM can also be used to suggest specific actions. For example if the robot detects a lamp nearby, it should suggest related actions such as *turn on the lamp* to the user.

3.4 UML Diagram

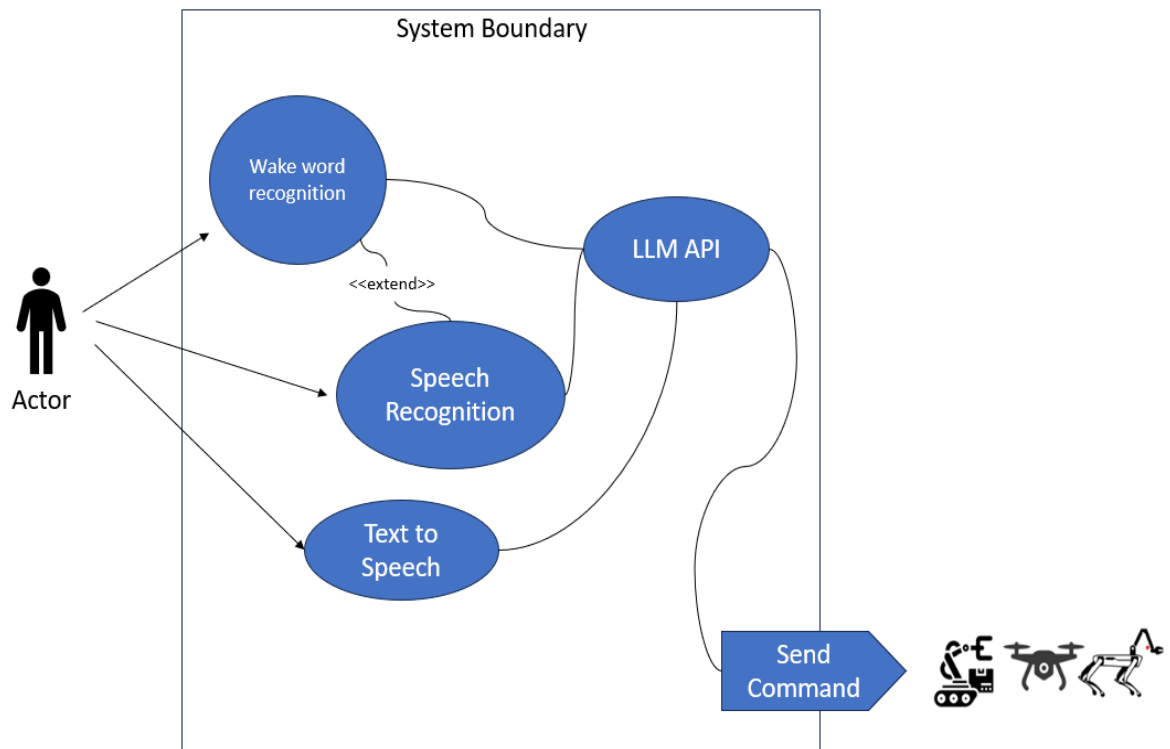


Figure 3.2: UML Diagram

4 Realization of Solutions

4.1 System Requirements

4.1.1 Open AI Api

Open AI Api is a cloud interface hosted on Microsoft Azure. It acts as a gateway for developers to use various cloud based Open AI's robust API models which include GPT 3.5 and Dall-E. It makes it simple to incorporate Open AI's API capability into a range of services and applications.

4.1.2 Robot Operating System (ROS)

Robot Operating System is a framework for creating robot applications that enables programmers to integrate pre-existing solutions for simple issues to create complicated systems. The primary characteristic of ROS is how the you may develop complicated software without having to understand how specific hardware functions thanks to the way software is operated and communicates. Nodes can be run on multiple devices, and they connect to that hub in various ways. It is completely language independent, which makes it easier to seamlessly integrate into any software module. Figure 4.1 shows the messaging structure of ROS.

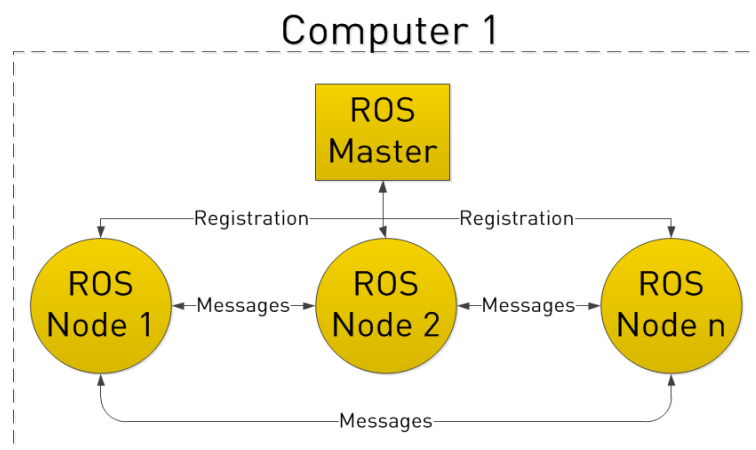


Figure 4.1: ROS messaging Structure [7]

4.1.3 Raspberry PI Operating System

In our project, Raspberry PI 4B is used as interface between robot and IOT devices. Ubuntu Mate 20.04 is used for the development. Ubuntu Mate is user friendly desktop operating system (dos). This is suited for usage with inexpensive computing devices like Raspberry Pi.

4.1.4 Message Queuing Telemetry Transport(MQTT)

To control IOT devices, MQTT protocol is used, which is an industry standard messaging protocol. With bidirectional connectivity, it is scalable, lightweight and efficient. MQTT offers three well defined Quality of Service (QOS) requirements and a dependable message delivery method. A publisher publishes the messages to a topic and subscriber must subscribe to the topic to view the message.

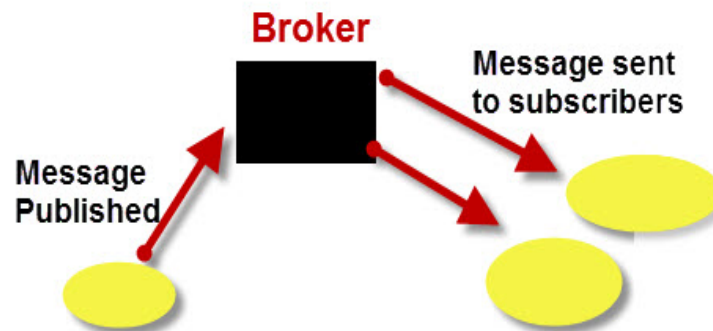


Figure 4.2: Mqtt- Publish Subscribe Model [6]

4.1.5 Home Assistant

Home Assistant is a home automation software where you can integrate many smart home gadgets into a single control system. It includes multiple communication protocols like MQTT and Zigbee. In our implementation, we connected a smart bulb which is present at Service computing Lab to home assistant via MQTT protocol that runs at a local server on Raspberry PI.

4.1.6 Docker Container

A container is a standard software unit which encapsulated all the code together with all of its dependencies to enable rapid and dependable application execution accross various computing environments [3]. In the case of Docker containers, images become containers during Docker Engine operation. Containerized software is available for both Windows and Linux-based apps, and

it will always function the same way regardless of the infrastructure. Containers allow software to be isolated from its surroundings and guarantee consistent operation even in case of variations, such as between development and staging environments.

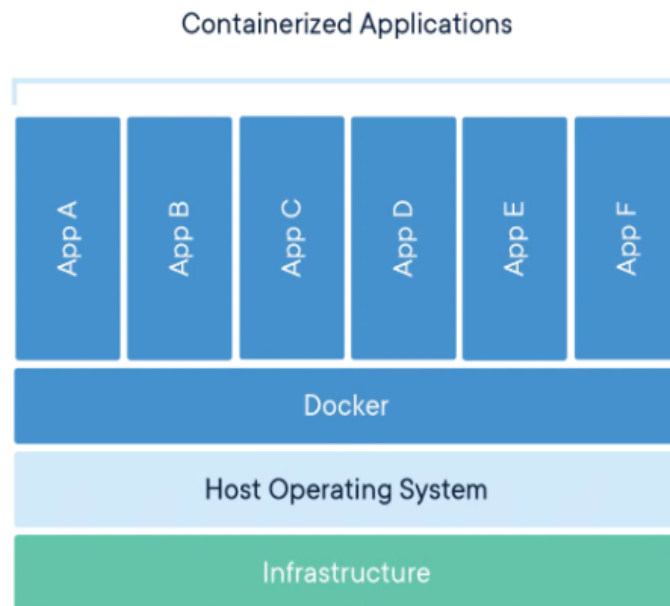


Figure 4.3: Docker Container [3]

4.2 Hardware Requirements

4.2.1 Ohmni Telepresence Robot

Ohmni the telepresence robot provides an engaging means of communication from any place. You can virtually be in two locations at the same time. It is equipped with real time navigation and 4K resolution camera and intel atom processor. It also has 15W speaker, a three wheel drive, 95Wh LIFePO4 battery which has lifetime of 6+ hours [14]. For our implementation we are using *developers addition* of the robot which enables us access to some core functionality of the robot like storage management and uploading custom containers.

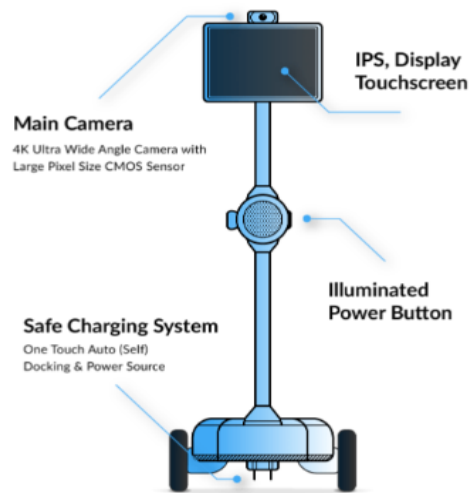


Figure 4.4: Ohmni Robot [14]

4.2.2 Raspberry Pi

Raspberry Pi is the name of a series of single-board computers made by the Raspberry Pi Foundation. There have been many generations of the Raspberry Pi line: from Pi 1 to 4. For our implementation, we are using Raspberry Pi 4B which has 8Gb of DDR4 RAM and 4 USB ports. It is built on Quad core 64 bit Arm Cortex A72 running at 1.5 GHz [18].

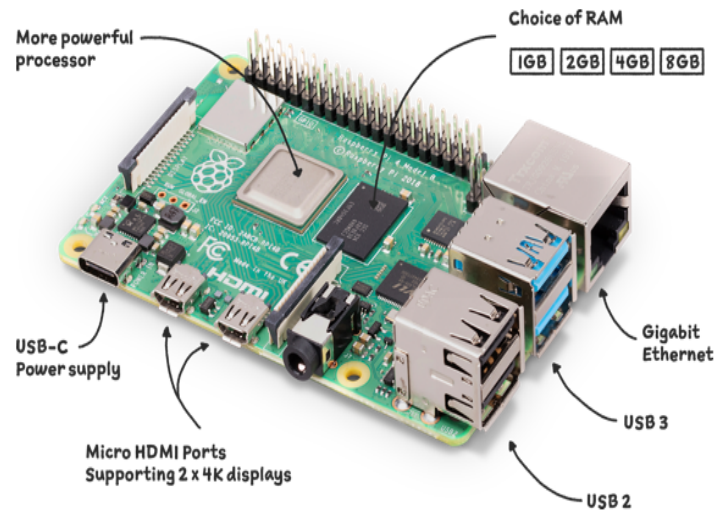


Figure 4.5: Raspberry Pi 4B [18]

4.3 Implementation

The ROS packages inside the docker container are deployed on the robot. The implementation consists of several nodes which can run independently of each other. Our implementation consists of 4 nodes.

- 1) **Wake Word Detection Node:** This node continuously listen for a wake word. Once the particular wake word is detected, the robot will start listening to the commands.
- 2) **GPT parser node:** This node takes the command input from the wake word detection node and pass it into an LLM api.
- 3) **Command Parser:** This node is responsible to perform specific actions based on the command given.
- 4) **Text to speech:** After performing the action, feedback is given in the form of speech whether the task is successfully performed or not.

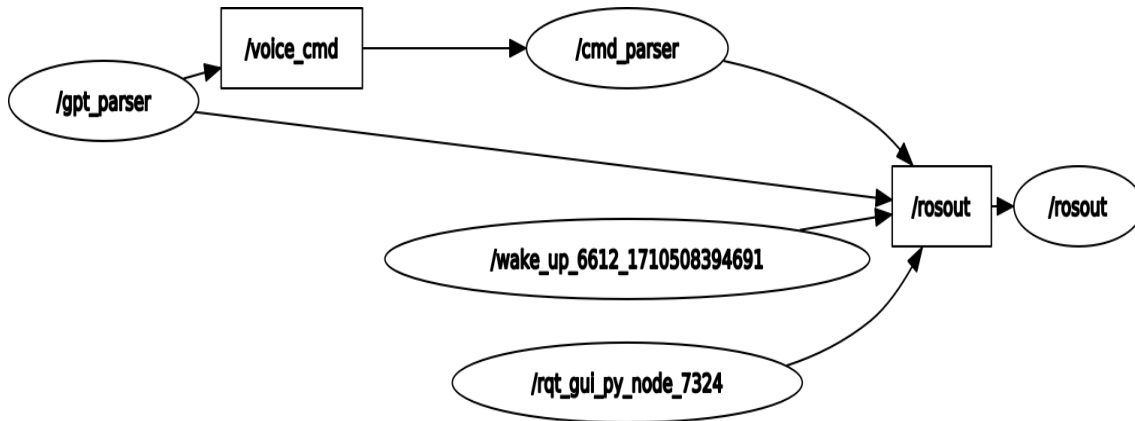


Figure 4.6: RQT Graph

4.3.1 Porcupine Wake Word Detection

Porcupine is one of the stable packages for wake word detection. It is trained online with a few steps. After training it generates a model which can be used in our application for inference. The block diagram below shows the flow of porcupine wake word detection. [13]



Figure 4.7: porcupine wake word [13]

Training

Picovoice console is used to train the wake word engine. In order to train the model we choose from variety of languages and architectures, as well as record audio. It offers an efficient way for downloading the model.

In our implementation, the model is trained with the wake word *Hello Kai*. It uses deep neural networks which are trained on real-world environments to build the model. Hence it is highly accurate and lightweight. Thus it is suited for low level devices like Raspberry PI, Jetson Nano and Beagle Bone.

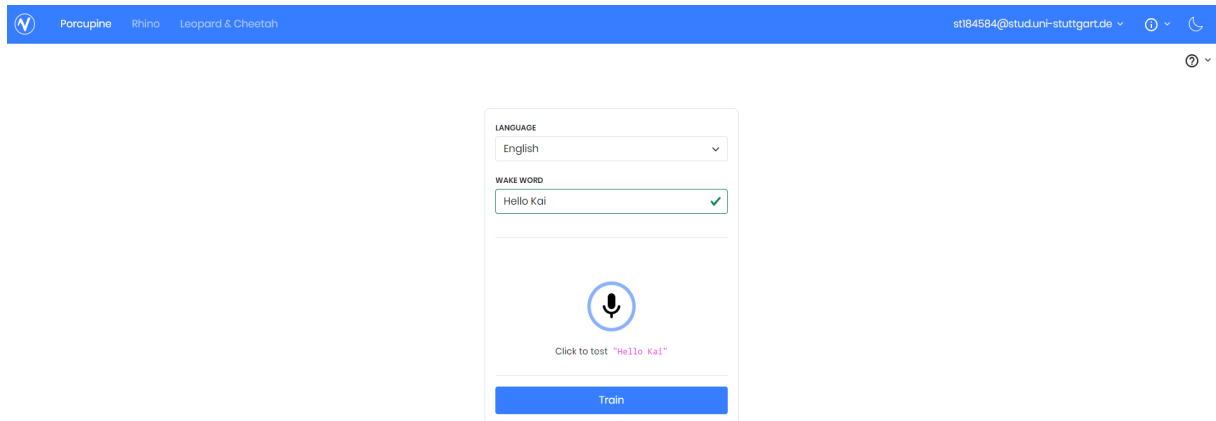


Figure 4.8: porcupine console

```
ALSA lib pcm_oss.c:377:(_snd_pcm_oss_open) Unknown field port
ALSA lib pcm_oss.c:377:(_snd_pcm_oss_open) Unknown field port
ALSA lib pcm_usb_stream.c:486:(_snd_pcm_usb_stream_open) Invalid type for card
ALSA lib pcm_usb_stream.c:486:(_snd_pcm_usb_stream_open) Invalid type for card
ALSA lib pcm_dmix.c:1089:(snd_pcm_dmix_open) unable to open slave
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
Listening {
  Hello-Kai (0.50)
}
```

Figure 4.9: Waiting for wake word

Deployment

A *.pmdl* file can be downloaded and used into the application once the model is trained and downloaded from the picovoice console. In our application we have used python environment to recognize the audio once the wake word is detected. A ROS wrapper is has been developed to work with inference of porcupine wakeword. The model contains few arguments which must be passed through, when it is deployed. *Access Key* is crucial variable that must be passed. Access key is a unique key for each user through which picovoice determines the usage and number of devices on which the particular model is implemented. The number of devices on which the model can be utilized is limited to 3.

```
15 import speech_recognition as sr
16
17 class PorcupineDemo(Thread):
18     def __init__(
19         self,
20         access_key,
21         library_path,
22         model_path,
23         keyword_paths,
24         sensitivities,
25         input_device_index=None,
26         output_path=None):
27
28         super(PorcupineDemo, self).__init__()
29
30         self._access_key = access_key
31         self._library_path = library_path
32         self._model_path = model_path
33         self._keyword_paths = keyword_paths
34         self._sensitivities = sensitivities
35         self._input_device_index = input_device_index
36
37         self._output_path = output_path
38
39     def listen(self):
```

Figure 4.10: Porcupine Wake Word Parameters

4.3.2 Google Speech to Text

Google Speech-to-Text is a cloud-based API service that utilizes powerful neural networks to convert audio recordings into text. It supports over 125 languages and variations, making it a versatile tool for developers [21]. It is focused on large amounts of language specific supervised data. This technique enables users improved recognition and transcription for variety of spoken languages and accents.

We use the python package *SpeechRecognition* which enables us access to Google Cloud API. In our implementation, this node is combined with wake word detection node. The speech recognition will start recognizing the input once the wake word is detected. The sleep time is set to 3 seconds, after which the user needs to say the wake word again to issue the commands to the robot.

Google speech to text api offers online cloud service which means the package is dependent on internet connection. This is one of the limitations of the package. In addition, Vosk Speech recognition can be utilized, which works independent of cloud or internet. The Vosk ROS package can be directly downloaded with pre trained weights and graph models [13].


```

37 self._output_path = output_path
38
39 def listen(self):
40     """
41     Here we need to read the input from the user and convert it into string and publish it.
42     """
43     rospy.loginfo("Here we will start to listen")
44     r = sr.Recognizer()
45     with sr.Microphone() as source:
46         audio=r.listen(source)
47         query = ''
48         try:
49             r.adjust_for_ambient_noise(source, duration=0.2)
50             query = r.recognize_google(audio, language='en-US')
51             return query
52         except Exception as e:
53             print("[ERROR] Invalid text, try again ")
54     time.sleep(2)
55

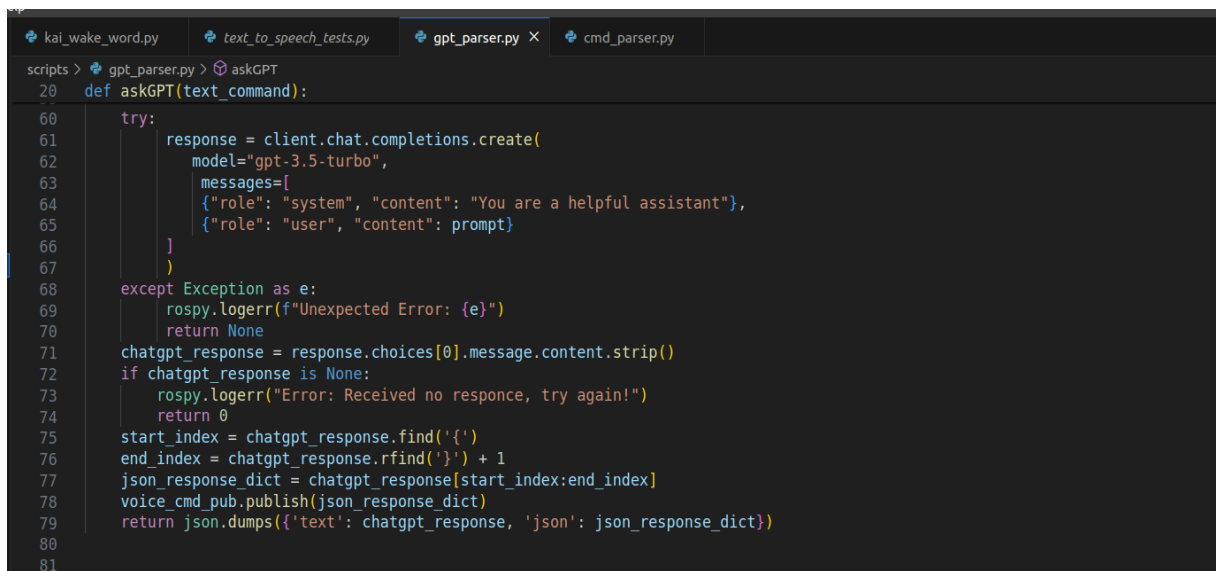
```

Figure 4.11: Google Speech to Text

4.3.3 Open AI API

Open AI released an API that allows users to access new AI models like GPT 3.5, and Dall-E. The API offers general purpose *text in, text out* interface, enabling users to test it on almost any English language task, in contrast to most AI systems that are built for single use case [8].

The API will attempt to match the pattern you provide it with when it receives a text prompt and return a text completion. It can be "programmed" by giving it a few samples of the tasks you'd like it to perform [15]. The complexity of the task will usually determine how successful it is. Through training on a dataset (small or large) of samples you give, or by learning from human feedback provided by users, the API also enables you to fine-tune performance on particular tasks.



```

scripts > gpt_parser.py > askGPT
20 def askGPT(text_command):
21
22     try:
23         response = client.chat.completions.create(
24             model="gpt-3.5-turbo",
25             messages=[
26                 {"role": "system", "content": "You are a helpful assistant"},
27                 {"role": "user", "content": prompt}
28             ]
29         )
30     except Exception as e:
31         rospy.logerr(f"Unexpected Error: {e}")
32         return None
33     chatgpt_response = response.choices[0].message.content.strip()
34     if chatgpt_response is None:
35         rospy.logerr("Error: Received no response, try again!")
36         return 0
37     start_index = chatgpt_response.find('{')
38     end_index = chatgpt_response.rfind('}') + 1
39     json_response_dict = chatgpt_response[start_index:end_index]
40     voice_cmd_pub.publish(json_response_dict)
41     return json.dumps({'text': chatgpt_response, 'json': json_response_dict})
42

```

Figure 4.12: OpenAi API parameters

4.3.4 Text to Speech

Different Text-to-Speech(TTS) modules are used to give feedback to the user whether the task is successfully performed or not. Two models are used which can be selected by user; *Eleven labs tts* and *python pyttsx3*.

Eleven Labs API offers realistic AI voices. It generates high quality spoken audio in any voice style and language. Everytime the API is used, the voice is generated in a .wav format and it is saved in a local directory. This is one of the limitation of Eleven labs API when it comes to storage. Python Text to speech (pyttsx3) is a text-to-speech conversion library in Python. Unlike Eleven Labs API, it works offline and generates real time audio. Apart from the above mentioned models, there are several other TTS models that can be used like nix-TTS and mbrola which has low latency and clear voice generation [13].

```
31     headers = {
32         "xi-api-key": ELEVENLABS_API_KEY
33     }
34     response = requests.get(url, headers=headers)
35     return response.json()["voices"]
36
37 def generate_audio(text: str, output_path: str = "") -> str:
38     """Converts
39
40     :param text: The text to convert to audio.
41     :type text : str
42     :param output_path: The location to save the finished mp3 file.
43     :type output_path: str
44     :returns: The output path for the successfully saved file.
45     :rtype: str
46
47     """
48     voices = get_voices()
49     try:
50         voice_id = next(filter(lambda v: v["name"] == ELEVENLABS_VOICE_NAME, voices))["voice_id"]
51     except StopIteration:
52         voice_id = voices[0]["voice_id"]
53     url = f"https://api.elevenlabs.io/v1/text-to-speech/{voice_id}"
54     headers = {
55         "xi-api-key": ELEVENLABS_API_KEY,
56         "content-type": "application/json"
57     }
58     data = {
59         "text": text,
60         "voice_settings": {
61             "stability": ELEVENLABS_VOICE_STABILITY,
62             "similarity_boost": ELEVENLABS_VOICE_SIMILARITY,
63         }
64     }
65     response = requests.post(url, json=data, headers=headers)
66     with open(output_path, "wb") as output:
67         output.write(response.content)
68     return output_path
```

Figure 4.13: Eleven Labs TTS parameters

4.3.5 Home Assistant

The device automation is handled via Home Assistant. It is deployed as a supervised installation on a Raspberry Pi 4 to enable MQTT broker communication, which enables devices to be connected to Home Assistant. The Philip smart light, which is located at service computing lab, can be linked to Home Assistant, enabling you to create automation scenes with configurable triggers, conditions and actions. We can control the light and retrieve the state of the light in the living lab.

4.3.6 Appointment Scheduling system

The application is also capable to send email to recipient using a database. With this, the users can communicate with the robot and email requests for appointments. To do this, we login to the Gmail client using a python script that stores login credentials and allows to send and receive mail.

The mail request from LLM api is processed with the database we build and acquire valid recipient email and name. This feature can be extended to university database server at a higher level.

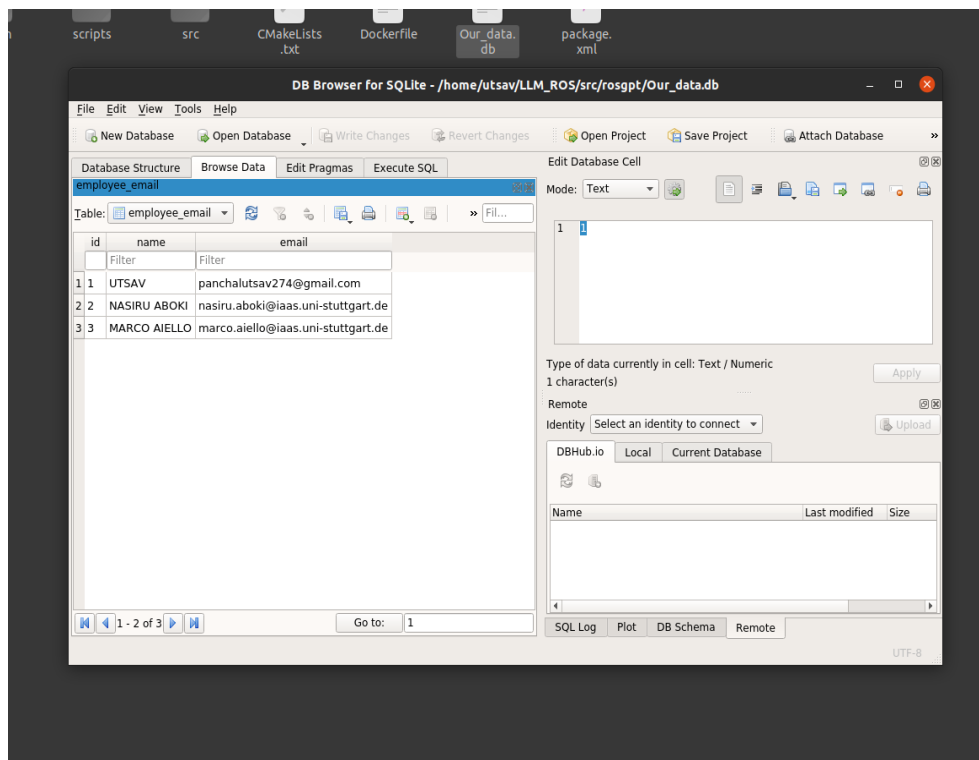


Figure 4.14: SQL Database

5 Evaluation

The implementation is evaluated in different stages. Different modules are used in this project which has their own evaluation metric.

5.1 Wake Word Model Evaluation

Wake Word Pitch behaviour includes accurate detection of wake words. The performance on wake word models is mostly dependent on its ability to detect wake words with different pitches. Different evaluation metrics like False Rejection Rate (FRR) and False Acceptance Rate (FAR) are used [2]. The FRR for a wake word model is the probability of the wake word engine missing the wake up phrase. Ideally we want the FRR to be zero. The FAR for a wake word is usually expressed as FAR per hour. It is the number of times a wake word engine incorrectly detects the wake phrase in an hour. Ideally we like FAR per hour to be zero as well as shown in Figure 5.1.

A superior algorithm has a lower FRR for any given FAR. To combine these two metrics, sometimes ROC curves are compared by their Area Under Curve (AUC). Smaller AUC indicates superior performance. In the Figure 5.1, algorithm C has better performance (and lower AUC) than B, and B is better than A.

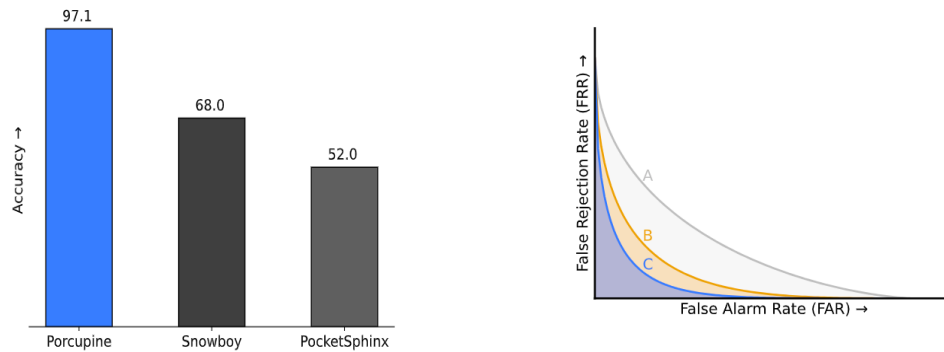


Figure 5.1: Porcupine Performance Measures [2]

Apart from this, sensitivity tests was also conducted by keeping the octaves constant. The wake word is uttered once and the number of triggers is counted. This is a crucial part of validation as it depicts the suitable sensitivity range of our model.

The behavior of each wake word model is clearly depicted in the previously displayed graphs. At first, we thought the porcupine wake word did better than the others. These figures also demonstrate the high number of false triggers even while efficient wake word reacts to considerably lower

5 Evaluation

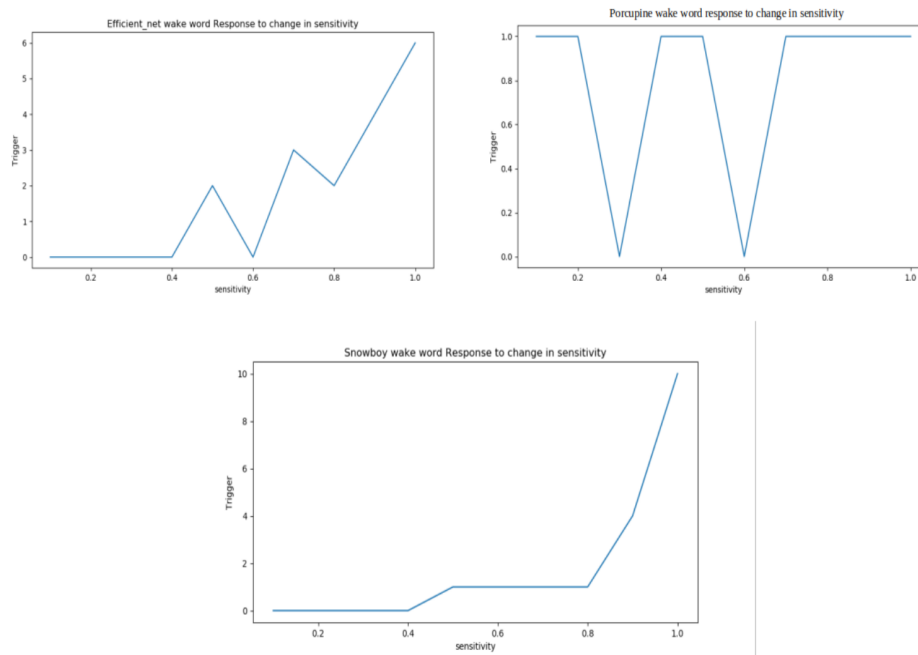


Figure 5.2: Sensitivity Test [13]

pitched voices. In terms of pitch behavior, Snowboy behaves like a porcupine; yet, when it comes to sensitivity, it does poorly, providing up to ten triggers for a single word when the sensitivity is higher. Since the trigger is not very dependent on sensitivity, we can conclude that porcupine has the best sensitivity behavior as well as a decent pitch behavior.

5.2 Speech to Text Evaluation

The evaluation metric used to benchmark speech recognition methods is Word Error Rate (WER). It is derived from "Levenshtein distance". This approach functions at the word level instead of phoneme level. . *Benchmark STT* software is used to calculate the word error rate. The idea here is that we try to demonstrate an audio file of a known sentence or a paragraph and record the speech to text output. The recorded output is then compared with the original paragraph using open source software such as "benchmark STT", which gives out the number of replaced words, equal words, inserted words, and deleted words [13]. Given that deep speech has a larger insertion and deletion range than the other two models, it is clear from the graphs that this model is not the best option for our application.

5.2 Speech to Text Evaluation

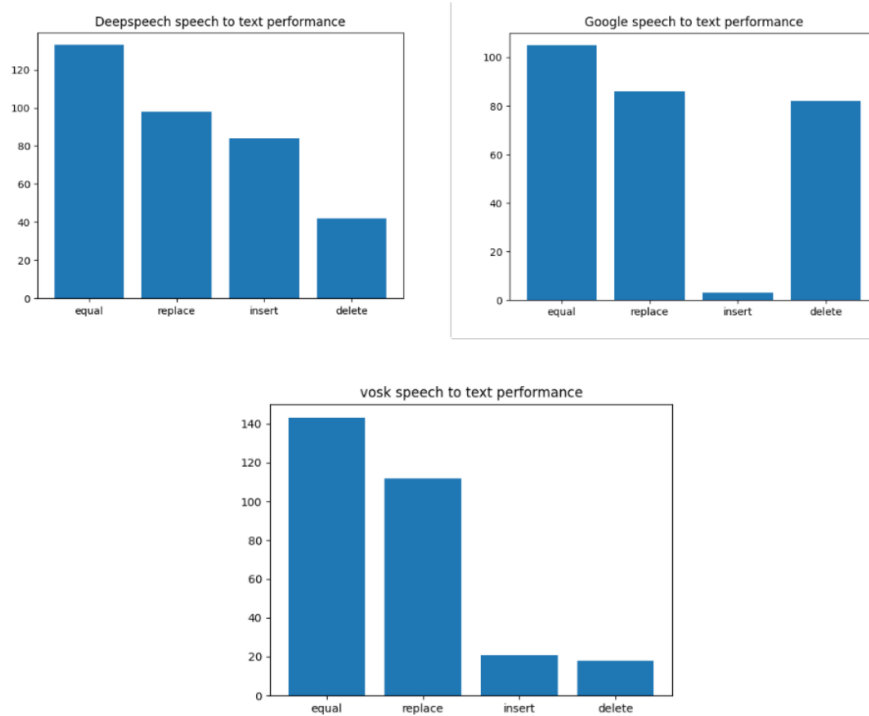


Figure 5.3: STT evaluation [13]

6 Conclusion and Future Work

The implementation provides a basic architecture to integrate LLM's on robotic control applications. The package is developed in such a way that it can easily integrate any available open source LLM API with very less changes. The package is also flexible which is a major advantage of using LLM instead of NLU unit. The LLM can be trained by providing few prompts when new use case needs to be added. The ROS packages are also developed as a scalable architecture. This also allows us to integrate it with several IOT devices.

In future this module can be integrated with more powerful LLMs which are fine tuned for our applications. Google Gemini and LLAMA Api can be used as they are fine tuned for code and multi modal applications. Currently this module works on open source LLM, which imposes us with limited API calls. The limitation can be removed by having premium user access which allows us to make infinite number of API calls. Several ROS modules can be added to create a complete set of human robot interaction. The module can be integrated with a camera module to enable face detection. SLAM and topological mapping features can be added to generate a virtual map of the lab. Different user accounts can be created so that the user can save related history and can continue the functional behaviour from where it left before. Access to various IOT devices like alarm, speaker, monitor can also be configured to the Home assistant.

Bibliography

- [1] R. Agarwal. *Alexa and the technology behind it*. Sept. 2021. URL: <https://medium.com/geekculture/alexa-and-the-technology-behind-it-e5c00793f85> (cit. on p. 12).
- [2] *Design, develop and Ship useful voice features*. URL: <https://picovoice.ai/> (cit. on p. 45).
- [3] *Docker Container*. URL: <https://www.docker.com/resources/what-container/> (cit. on pp. 34, 35).
- [4] H. M. He. *RobotGPT: From ChatGPT to Robot Intelligence*. 2023. URL: https://openreview.net/forum?id=wWe_0qpCcU8 (cit. on pp. 11, 14, 15).
- [5] M. Helal, P. Holthaus, G. Lakatos, F. Amirabdollahian. *Chat Failures and Troubles: Reasons and Solutions*. 2024. arXiv: 2309.03708 [cs.R0] (cit. on p. 28).
- [6] *How MQTT works-Beginners Guide*. URL: <https://www.clearpathrobotics.com/assets/guides/melodic/ros/Intro%20to%20the%20Robot%20Operating%20System.html> (cit. on p. 34).
- [7] *Intro to ROS*. URL: <https://www.clearpathrobotics.com/assets/guides/melodic/ros/Intro%20to%20the%20Robot%20Operating%20System.html> (cit. on p. 33).
- [8] *Introducing ChatGPT*. URL: <https://openai.com/blog/chatgpt> (cit. on pp. 25, 41).
- [9] *Large Language Models Explained*. URL: <https://www.nvidia.com/en-us/glossary/large-language-models/> (cit. on pp. 11, 21).
- [10] F. Li, D. Hogg, A. Cohn. *Ontology Knowledge-enhanced In-Context Learning for Action-Effect Prediction*. Nov. 2022. URL: <https://eprints.whiterose.ac.uk/194133/> (cit. on pp. 19, 20).
- [11] *LLAMA*. URL: <https://llama.meta.com/> (cit. on pp. 15, 26).
- [12] A. Mahmood, J. Wang, B. Yao, D. Wang, C.-M. Huang. *LLM-Powered Conversational Voice Assistants: Interaction Patterns, Opportunities, Challenges, and Design Guidelines*. 2023. arXiv: 2309.13879 [cs.HC] (cit. on pp. 12, 13).
- [13] S. Nataraj. *Implementation of Speech Module on Ohmni Telepresence Robot*. 2022 (cit. on pp. 28, 38, 40, 42, 46, 47).
- [14] *Ohmni Labs - Telepresence Robots enable you to be anywhere, everywhere*. URL: <https://ohmnilabs.com/lp-inmarket/> (cit. on p. 36).
- [15] A. Parnami, M. Lee. *Learning from Few Examples: A Summary of Approaches to Few-Shot Learning*. 2022. arXiv: 2203.04291 [cs.LG] (cit. on pp. 12, 17, 41).
- [16] S. Pichai. *Introducing Gemini: Our largest and most capable AI model*. Dec. 2023. URL: <https://blog.google/technology/ai/google-gemini-ai/> (cit. on pp. 14, 15, 25).
- [17] *Prompt Engineering for Generative AI*. URL: <https://developers.google.com/machine-learning/resources/prompt-eng> (cit. on pp. 16, 17, 20).

- [18] *Raspberry Pi 4*. URL: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/> (cit. on pp. 36, 37).
- [19] G. Team. *Gemini: A Family of Highly Capable Multimodal Models*. 2023. arXiv: 2312.11805 [cs.CL] (cit. on pp. 14, 15).
- [20] H. Touvron, L. Martin, K. Stone. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023 (cit. on pp. 14, 15).
- [21] *Turn Speech to Text using Google AI*. URL: <https://cloud.google.com/speech-to-text?hl=en> (cit. on p. 40).
- [22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL] (cit. on p. 11).
- [23] N. Vedula, R. Gupta, A. Alok, M. Sridhar, S. Ananthakrishnan. “ADVİN: Automatically discovering novel domains and intents from user text utterances”. In: *ICASSP 2022*. 2022. URL: <https://www.amazon.science/publications/advin-automatically-discovering-novel-domains-and-intents-from-user-text-utterances> (cit. on p. 13).
- [24] *What are AI hallucinations*. URL: <https://www.ibm.com/topics/ai-hallucinations> (cit. on p. 27).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature