Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

RANEEM ABDEL HAFEZ

Enhancing Human-Robot Interaction: Integrating Large Language Models and Advanced Speech Recognition into the Pepper Robot

DEGREE PROGRAMME IN DATA ENGINEERING
(ARTIFICIAL INTELLIGENCE)
2024

ABSTRACT

Abdel Hafez Raneem: Enhancing Human-Robot Interaction: Integrating Large Language Models and Advanced Speech Recognition into the Pepper Robot
Bachelor’s thesis
Degree programme: Data Engineering – Bachelor of Engineering
Month Year: February 2024
Number of pages: 92

Conversing with robots was once limited to science fiction, but recent advances in robot integration into everyday human interactions have made it a reality. Pepper, the humanoid robot from Softbank Robotics, is a significant advancement in robotics, designed for seamless communication. However, its natural language capabilities are currently limited. Integrating Large Language Models (LLMs) and Speech Recognition is essential to enhance its communication abilities. Recent advancements in speech recognition, leveraging the Transformer architecture, have greatly improved transcription accuracy and efficiency of spoken language. Additionally, LLMs, like OpenAI's ChatGPT, can now generate human-like responses based on contextual input, pushing the boundaries of natural language understanding and generation.

This thesis investigates integrating LLMs into Pepper to improve its linguistic abilities and facilitate natural communication. The integration enhances Pepper's language understanding and generation, leading to more engaging Human-Robot Interaction (HRI).

Key questions driving this thesis include: How can large language models be integrated in pepper robot? Can Pepper accurately transcribe spoken audio? Are Pepper's responses human-like enough to facilitate meaningful interaction? And does Pepper's response time allow for a natural conversational flow?

The methodology involves the integration of LLMs into Pepper's architecture, alongside an Automatic Speech Recognition (ASR) for accurate speech recognition. Through the utilization of an ASR and evaluation of various LLMs, Pepper demonstrates commendable transcription capabilities and generates responses that are deemed sufficiently human-like for users to understand and engage with. Evaluation of the implemented models reveals notable differences in speed, with some models exhibiting faster response times than others.

Two primary phases are undertaken: establishing a web server infrastructure and configuring Pepper for seamless interaction with the server. Through qualitative assessments and quantitative analyses of response time and performance metrics, the most optimal LLM and ASR for Pepper's communication needs is identified.

Keywords: Language Models, Large Language Model, Pepper Robot, Automatic Speech Recognition, Natural Language Understanding, Natural Language Generation, Human-Robot Interaction, Linguistic Abilities, ChatGPT

FOREWORD

In embarking on this academic journey, I am honored to present this thesis as a result of my personal efforts and commitment towards implementing LLM into pepper robot.

First and foremost, I extend my heartfelt appreciation to my family—my parents, sisters, and brother—for their unwavering support and encouragement throughout this academic pursuit. Their love and belief in my abilities have been a constant source of inspiration. I am especially grateful for their active interest in my thesis. Their curiosity, valuable insights, and practical contributions during the thesis have significantly enriched my work. Whether engaging in discussions or offering suggestions, their involvement has turned this academic journey into a collaborative and rewarding experience.

I am immensely grateful to my supervisor, Aleksi Postari, whose time, dedication, and unwavering commitment to my success have been unparalleled for his guidance and support throughout the entire process, His insightful notes, constructive feedback, and mentorship have not only refined the academic aspect of this thesis but have also been a guiding force in shaping my research approach.

I am really grateful to my friends for supporting me on this journey. Who provided me immense support and encouragement throughout this difficult and long ride. Their company and assistance have turned the challenges into mutual victories.

Acknowledgment is also due to all those who have been a part of my academic trajectory—my teachers who have equipped me with the knowledge and skills essential for conducting this research. Their passion for imparting knowledge has been lightening my intellectual growth path.

CONTENTS

LIST OF SYMBOLS AND TERMS

LLM – Large Language Models

HRI – Human- Robot Interaction

NLP – Natural Language Processing

NLU – Natural Language Understanding

NLG – Natural Language Generating

AI – Artificial Intelligence

RL – Reinforcement Learning

SLM – Statistical Language Models

NLM – Neural Language Models

GNMT – Google Neural Machine Translation

AGI – Artificial General Intelligence

PLM – Pre-trained Language Models

PaLM – Pathways Language Model

GPUs – Graphical Processing Units

CPU – Central processing unit

GPT – Generative Pre-trained Transformer

BERT – Bidirectional Encoder Representations from Transformers

RNN – Recurrent Neural Network

RNNLM – Recurrent Neural Network Language Model

NN – Neural Networks

VAE – Variational Autoencoders

OOD – out-of-distribution

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

In recent years, the field of robotics has witnessed remarkable advancements, particularly in the domain of human-robot interaction (HRI) (Zhang et al., 2023). As the demand for seamless communication between humans and robots continues to grow, integrating sophisticated language understanding capabilities becomes imperative. This thesis explores the implementation of Language Models (LLMs) into the Pepper robot, a humanoid robot designed to engage with users naturally and intuitively.

1.1 Background and Motivation

The use of social robots, like Pepper robot, has increased across several industries, including customer service, education, and healthcare (Mishra et al., 2023). However, these robots need to be equipped with advanced language understanding and generation capabilities to improve the user experience genuinely (Zhang et al., 2023). This necessitates the integration of language models that can interpret user inputs and provide contextually relevant responses.

In this case, the thesis focuses on implementing LLMs in the Pepper robot to enrich its linguistic abilities. The background of this thesis lies in addressing the limitations of existing language processing capabilities in social robots, aiming to create a more immersive and compelling HRI (Zhang et al., 2023).

1.2 Research Questions

The central question guiding this thesis is: How can large language models be effectively integrated into the Pepper robot to enhance its language understanding and generation capabilities.
This thesis seeks to investigate the other research questions that could arise:

1. How well can the pepper robot transcribe spoken audio?

2. Are the responses from pepper robot human-like enough?
3. Is the latency generating the responses from the pepper robot good enough for natural feeling conversation?

This query prompts an exploration of the methodologies and models that can be leveraged to augment the linguistic competence of social robots.

1.3 Objective of the Thesis

The primary objective of this research is to implement LLMs into Pepper robot by incorporating Whisper Automatic Speech Recognition (ASR) model to capture user voice inputs, transcribe them into text form, and utilize LLMs for generating contextually appropriate responses. This involves a multi-step process where pepper built- in microphone captures audio, Whisper ASR accurately converts spoken commands or queries from users into text, which is then passed on to selected LLMs for inference. The evaluation of the three distinct LLMs focuses on response time and overall performance, assessing factors such as speed, accuracy, relevance, and naturalness of the generated responses. By systematically comparing the performance of different LLMs, the research aims to identify the most suitable model that aligns with the features and requirements of the Pepper robot, optimizing its communication capabilities and enhancing its usability and effectiveness in real-world scenarios.

1.4 Experimental Design

To achieve the objective mentioned above, the methodology involves integrating the chosen LLMs into the Pepper robot's architecture. The Whisper ASR model is utilized for accurate speech recognition when users opt to record their voice using the NAOqi libraries. The evaluations of the LLMs are conducted based on response time and performance metrics, ultimately leading to the selection of the most optimal model. The comprehensive process is delineated through an experimental design that unfolds in two primary phases.

In the first phase, the research focuses on establishing a web server, creating the foundational infrastructure necessary for the integration of LLMs. The second phase focuses on configuring the Pepper robot to seamlessly interact with the server, ensuring a smooth integration between the LLMs and the robot's architecture. These two phases constitute essential steps in the experimental process, setting the groundwork for the subsequent evaluation and testing of the integrated system. The evaluations of the LLMs are carried out using a dual approach, combining qualitative assessments of functionality by personally completing tasks and evaluating system effectiveness, and quantitative analyses through testing various methods, including measuring response time and performance metrics. This comprehensive methodology aims to guide the selection of the most optimal LLM for the intended purpose.

1.5 Use of artificial intelligence (AI) in this thesis

In this thesis, I have used ChatGPT as a tool for brainstorming, retrieving information, proofreading, helping me structure my thesis in a more organized way, and formulate sentences/statements that I thought could be written in a much clearer way. When using ChatGPT, I ensured that I wrote what I wanted to deliver to the reader and used the AI tools to advance my writing so that the language would be interesting to read and understandable but still say things in the way I meant them to be said. I also used AI applications to ensure my understanding of the text and ensure what I wrote was understandable to the reader.

I have ensured the authenticity of the content and respect for copyright. I didn’t use any new ideas brought to the text from ChatGPT without checking them for their originality, proof and appropriately referenced them. All sources in the bibliography are used by me, not by the AI. This can be checked from the reference management system I use.

In writing the thesis, I used Grammarly to check the grammar. I have used all AI applications responsibly, with due regard for data protection. AI has not been used in writing this subchapter 1.5.

By addressing these aspects, this thesis contributes to the ongoing discourse on HRI, providing valuable insights into the effective integration of language models for enhanced communication capabilities in social robots like Pepper.

2 THEORETICAL FRAMEWORK

In the theoretical framework section of this thesis, I delve into the foundational aspects shaping the integration of Large Language Models (LLMs) into the Pepper robot. This exploration navigates through the intricacies of the Pepper robot's design, elucidating its features, capabilities, and pivotal components such as Natural Language Processing (NLP), Natural Language Understanding (NLU), Natural Language Generation (NLG), and language models.

Language Models (LMs) play a pivotal role in the field of robotics by serving as a crucial link between the physical world and advanced language processing capabilities. This connection allows LLMs to glean insights from the environment through data collected by sensors, empowering robots to comprehend semantic meanings and engage in flexible dialogue interactions. The primary applications of LLMs in robotics encompass task planning and human-robot collaboration. Research efforts have explored diverse scenarios, including LLM-powered robots controlled through text for assembly tasks in virtual reality and the use of LLMs for conversational robots in specific contexts, such as creating personalized companion robots for interactions with older adults. (Callie et al., 2024.)

Within this theoretical framework, I introduce and justify these components to provide a comprehensive background check. This background check serves as a robust foundation for comprehending the intricate interplay between cutting-edge language technologies and the evolving role of the Pepper robot HRI. The inclusion of sub-sections dedicated to these components acts as both an introduction and a background check, strategically setting the stage for one of the central themes of the thesis—LLMs and their impact on HRI.

The objective of this exploration is to establish a solid theoretical foundation, offering insights into how these elements enhance the Pepper robot's language capabilities. This theoretical groundwork not only provides a backdrop for the practical implementation and evaluation discussed later in the thesis

but also sheds light on the transformative potential of advanced language technologies within the realm of HRI. The theoretical framework thus plays a crucial role in laying the groundwork for understanding and analyzing the synergies between LLMs and Pepper's capabilities, contributing to the broader discourse on the integration of language technologies in robotics.

2.1 Pepper robot: Humanoid robot

With the robotics advancements we see today, companies have taken the lead in building a humanoid robot to help on a day-to-day basis. Personal social robots will most likely be one of the major developments in the robotics industry as the field develops (Pandey & Gelin, 2018). With the advancements seen today, along with many AI and robotics developments, robots will soon coexist with humans and play significant roles in daily life, bringing improvements to everyone's lives.

2.1.1 Pepper: The Sociable Robot

One of the most sociably advanced robot ever created is Pepper. Pepper is an industrially produced humanoid robot developed by SoftBank Robotics and introduced in June 2014. Pepper, the first machine of its kind is designed with the vision to ease and improve human lives while being a social and emotional companion robot. When Pepper was first created, developers initially had the idea of making Pepper a B2B robot, meaning business-to-business, but due to its high interest from all around the world for various applications such as business-to-consumer (B2C), business-to-academics (B2A), and business-to-developers (B2D), it was later adapted for B2C purposes. Pepper is capable of exhibiting body language, perceiving and interacting with its surroundings, and moving around. It also has the ability to analyze/ recognize people’s expressions and voice tones with the help of the latest advancements and proprietary algorithms in voice and emotion recognition. (Pandey & Gelin, 2018.)

2.1.2 Pepper's Features and Capabilities

Pepper is a 3 omnidirectional wheeled 1.2-m-tall robot with the ability to move around smoothly and 17 joints for expressive body language (Pandey & Gelin, 2018). 12 hours of battery life for nonstop activities and the ability to return to the charging station (Pandey & Gelin, 2018). The size and appearance of the machine are intended to make it suitable and acceptable for interacting with people in daily life. It has a tablet (which also makes programming and debugging easier) and is built for various multimodal expressive movements and actions. “The machine has an Atom E3845 processor with a quad-core central processing unit (CPU) and a clock speed of 1.91 GHz. It has a 4-GB double data rate and type-three random access. memory and flash memory of 32 GB embedded multimedia card, of which 24 GB are available for users,” as stated in the original paper by (Pandey & Gelin, 2018). Pepper robot has a range of sensors, allowing it to perceive objects in its surroundings. Figure 1 shows the various sensors equipped with pepper. Here is a breakdown of each component of the robot and its functions:

1. Six-axis Inertial Measurement Unit (IMU) Sensor:
    - Function: Measures motion and orientation.
    - Components: A three-axis gyrometer (angular speed of ~500 °/s) and a three-axis accelerometer (~2 g).
    - Purpose: Estimation of base speed and attitude (yaw, pitch, and roll).

    (Pandey & Gelin, 2018.)

2. Microphones:
    - Function: Capture audio for sound localization.
    - Specifications: Four microphones with a sensitivity of 250 mV/Pa (±3 dB at 1 kHz) and a frequency range of 100 Hz to 10 kHz (−10 dB relative to 1 kHz).

    (Pandey & Gelin, 2018.)

3. Cameras and Three-Dimensional (3-D) Sensor:
    - Function: Capture visual information.
    - Components: Two RGB cameras (forehead and mouth positions), one 3-D sensor (behind the eyes).

- Camera Resolution: 2,560 × 1,920 at 1 frame/s or 640 × 480 at 30 frames/s.
- 3-D Sensor Resolution: Up to 320 × 240 at 20 frames/s.

(Pandey & Gelin, 2018.)

4. Tactile Sensors, Bumper Sensors, and Tablet:
    - Function: Detect touch and impacts and provide user interface.
    - Components: Three tactile sensors (one in the head, one on each hand), three bumper sensors (one on each wheel position), and a tablet attached to the chest.

    (Pandey & Gelin, 2018.)

5. Laser Sensing Modules:
    - Function: Assist in navigation and obstacle detection.
    - Components: Six laser line actuators (generators) and three sensors.
    - Actuator Locations: Three at the front for ground evaluation, three at the lower base for surrounding sensing.
    - Sensor Locations: Front, left, and right sides.

    (Pandey & Gelin, 2018.)

6. Loudspeakers, Sonar Sensors, and Infrared Sensors:
    - Function: Output sound, detect objects, and provide additional sensing.
    - Components: Two loudspeakers (lateral placement on the left and right sides of the head), two sonar sensors (front and back), and two infrared sensors at the base.

    (Pandey & Gelin, 2018.)

7. Network Connectivity Support:
    - Function: Enable communication and connectivity.
    - Connectivity Types: Ethernet (1x RJ-45 10/100/1000 Base-T) and Wi-Fi (IEEE 802.11 a/b/g/n; Security: 64/128-bit WEP, WPA/WPA2).
    - Communication Protocols: RS-485 between motor/sensor board and internal computer; USB cable with Ethernet control model for communication between the tablet and the CPU.

    (Pandey & Gelin, 2018.)

Each of these components plays a specific role in the robot's functionality, including sensing its environment, capturing audio and visual data, providing touch-based interaction, and enabling communication with other devices or networks.

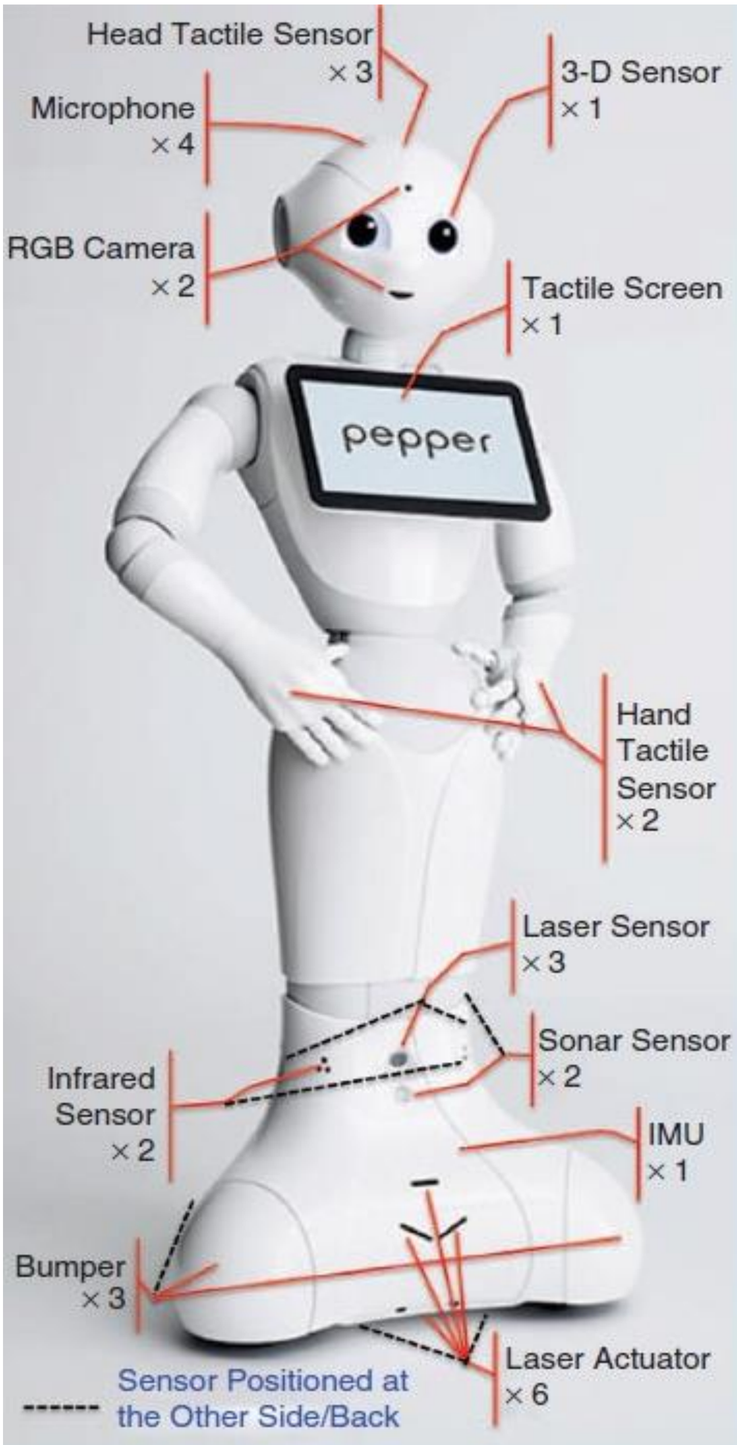


Figure 1. Diagram of Pepper robot’s sensors (Pandey & Gelin, 2018)

2.1.3 The Operating System: NAOqi

NAOqi is the operating system that runs on Pepper and controls it. NAOqi offers a programming framework that makes it possible for developers to customize peppers' behavior and programming by writing software programs that can interact with and control the robot’s hardware and sensors. Many software development kits are offered to control and develop Pepper: The Robot Operating System (ROS) Interface, Python, C++, Java, and JavaScript. The primary requirements for the initial B2B scenario involving the Pepper robot emphasize the importance of human interaction and perception. To perceive humans and avoid collisions, Pepper is equipped with hardware and software capabilities. It has a module called NAOqi People Perception, which has a few built-in APIs to aid in developing sophisticated behavioral and reasoning abilities. Notably, Pepper excels in dialog-based interaction using the NAOqi ALDialog and Qichat modules, enabling natural language input and commands through various APIs. Additionally, it utilizes the Animated Speech and Expressive Listening modules and 17 articulations to enhance its human-like gestures and fluid movements, aiming for a high level of human-robot engagement. The combination of tactile areas, LEDs, and a tablet allows Pepper to interact with humans in a multimodal way, including speech, gestures, and a graphical user interface, leveraging these APIs for versatile interaction. (Pandey & Gelin, 2018.)

2.1.4 Pepper's Specialized Intelligence and Limitations

Pepper's intelligence is limited and specialized for the tasks it was designed/made to do. It doesn’t have a highly intelligent system or advanced AI applications. It has the following: NLP, where pepper can understand and generate natural language to a certain extent. It can answer simple questions, engage in basic conversations, and follow simple verbal Commands. Emotion Recognition, as mentioned earlier, pepper is designed to deal with day-to-day human interactions, and that includes recognizing and responding to human emotions based on their voice tones, context, and facial expressions. With this feature,

Pepper can adapt its behavior to appear more empathetic and understanding. Autonomous Navigation, Pepper has sensors and cameras that enable it to navigate safely and the ability to move autonomously in its environment, avoiding obstacles using anticollision software. (Pandey & Gelin, 2018.)

Pepper's primary use is to engage in friendly interactions with humans in customer services such as hospitality, which also means it is not a groundbreaking advancement in AGI (artificial general intelligence) or human-level intelligence. Its capabilities are specialized and narrow in scope. And that’s where improvements in Pepper's ability to understand, process, and generate human-like natural language should take place.

2.1.5 Enhancing Pepper’s Behavior and NLP Abilities

My focus is on its NLP application, particularly how well it can respond to straightforward questions and how effectively it handles more complex problems. This analysis suggests how NLP advancements can enhance its capacity to comprehend human language and engage in more meaningful interactions genuinely. The developers of Pepper have also stated that for prospects to achieve successful general-purpose sociable robots, the machines must behave in a socially accepted and expected manner, and in creating such robots they need to have the ability to understand their surroundings, act fast with what they have, and dealing with different people and different enquires. To do that, Connectivity, learning, cloud-based collective intelligence to enhance the robots' social intelligence and proactive behavior and comprehend human-robot engagement are some avenues to pursue in this respect stated in the original paper of Pepper. The paper doesn’t explicitly mention NLPs or LLM, but the concepts mentioned, “learning” and “comprehending human-robot engagement,” could potentially involve aspects of NLP in the context of improving social intelligence. Improving a robot’s ability to understand, comprehend, and engage in natural language conversations with humans would fall under the domain of NLP. (Pandey & Gelin, 2018.)

2.2 Natural Language Processing

2.2.1 The beginning of NLP

NLP, Natural language processing, is an intersection of artificial intelligence and linguistics that began in the 1950s, aiming to study the problems derived from automatic generation and understanding of natural language (Liddy, 2001). NLP serves as a method to narrow the communication divide between computers and humans (Liddy, 2001). Its roots trace back to the nascent machine translation (MT) concepts that emerged during World War II. A significant milestone in this journey occurred in 1954 with the Georgetown experiment, where over sixty Russian sentences were automatically translated into English through a collaborative effort between IBM, Figure 2 is a flowchart of part of the dictionary lookup program from Sheridan 1955 and Georgetown University (History and Present of Natural Language Processing "Deep Talk," 2020). Unfortunately, after years of research, following the ALPAC report in 1966, which revealed that the ten-year research hadn't met anticipated outcomes, funding for machine translation was significantly cut back (Liddy, 2001).

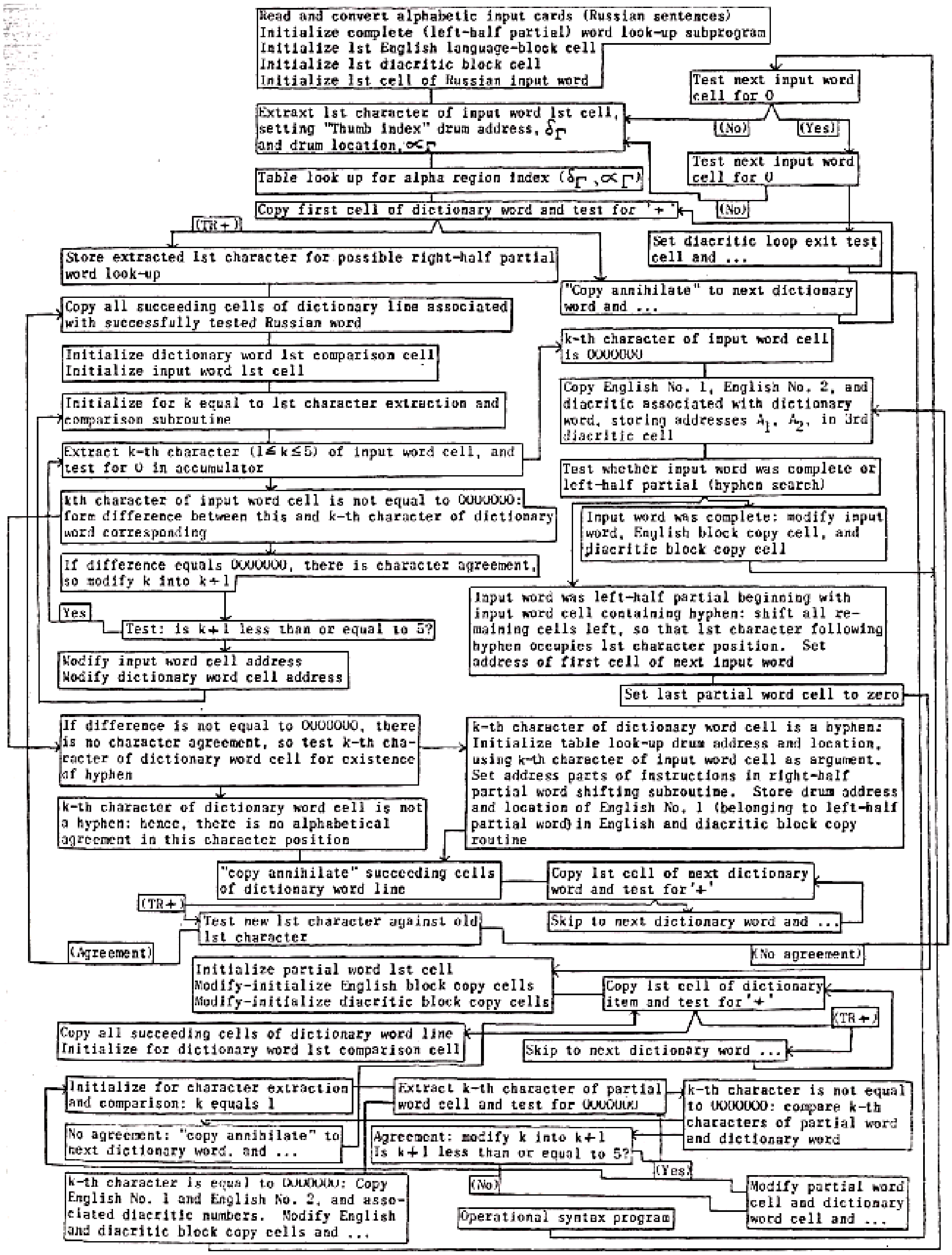


Figure 2. Flowchart of part of the IBM dictionary (Hutchins, 2004)

2.2.2 Key Concepts

Natural Language Processing (NLP) is the application of computational techniques informed by linguistic theories to analyze and represent raw textual content at various linguistic levels, enabling machines to understand, decode, and transform text for data analysis across diverse sectors (Liddy, 2001). NLP employs machine learning to empower computers with the capacity to understand, manipulate, and interpret human language (Liddy, 2001). Its goal is to achieve human-like language processing for various tasks or applications. The key word of the previous sentence is processing and not understanding because, in the early days of AI, NLP was referred to as Natural Language Understanding (NLU). Since the goal of NLP is NLU, that goal has not yet to be accomplished as of 2001 (Liddy, 2001). In Liddy's, the author, words (2001), a complete NLU system would need to be able to do all the following:

1. paraphrase a text,
2. translate texts,
3. answer questions about contents of the text,
4. draw inferences from the text

(Liddy, 2001)

With the earlier NLP technology, it could only accomplish the first three goals since NLP systems cannot, by themselves, draw inferences from texts, and that the NLU was the goal of NLP. As a result of today's significantly advanced NLP systems, NLP can draw inferences from text. These systems use various methods, including machine learning, deep learning, and semantic analysis, to comprehend the relationships, context, and subtleties. Based on the input text, NLP models can spot patterns, extract pertinent information, and generate predictions or conclusions based on the input text. This skill has several techniques, including sentiment analysis, text summarization, keyword extraction, language translation, and many more. Despite the fact that NLP systems have advanced significantly, it can still be difficult to provide correct and contextually relevant conclusions, particularly in complicated or confusing situations. (Liddy, 2001.)

2.2.3 How NLP works

Computers now possess the ability to understand natural language akin to human comprehension, courtesy of NLP, or Natural Language Processing. NLP employs artificial intelligence to analyze real-world information and interpret it in a manner comprehensible to computers, whether in spoken or written form (Fred, 2022). Similar to humans relying on sensory organs like ears and eyes, computers utilize reading programs and microphones to capture audio inputs (Gupta, 2023). Just as humans process information through the brain, computers employ programs to handle diverse inputs (Gupta, 2023), ultimately converting them into computer-readable code during the processing phase (Gupta, 2023). Again, NLP works through linguistics rules, statistical patterns, and machine learning techniques (What Is Natural Language Processing? | IBM, n.d.). The NLP pipeline is not always strictly linear, and some tasks may be performed iteratively or in different orders based on the application; it consists of three main stages: preprocessing, features extraction, and modeling (Tech Gumptions, 2023), as shown in Figure 3. Additionally, advances in deep learning and transformer-based models have led to end-to-end approaches where multiple pipeline stages are integrated into a single model.

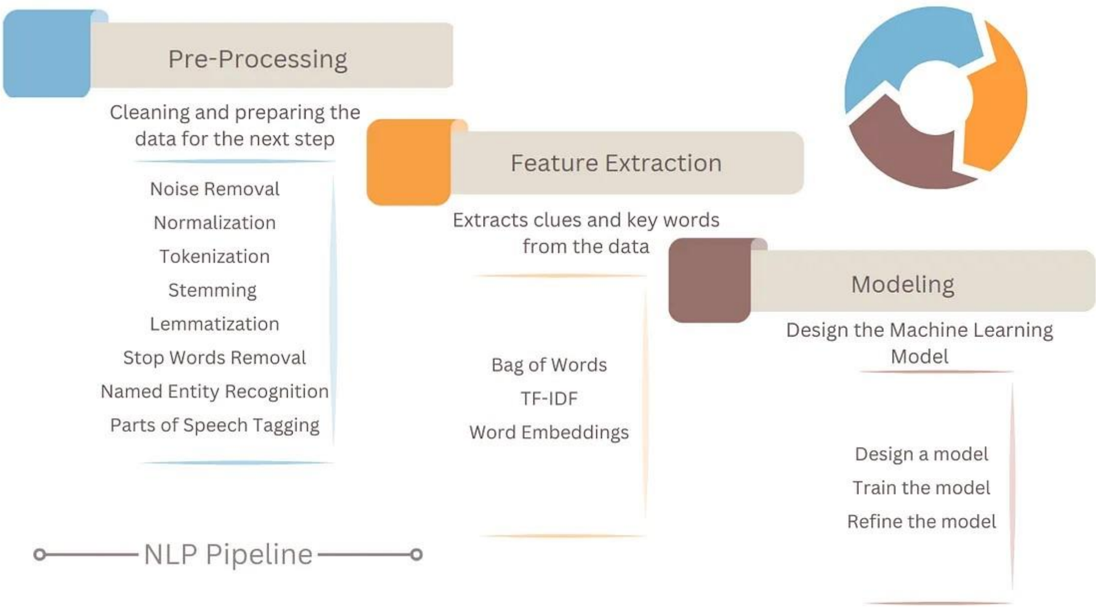


Figure 3. The NLP Pipeline (Tech Gumptions, 2023)

2.2.4 NLP in Customer Communication

Natural language processing (NLP) is critical to analyzing text and speech data thoroughly and efficiently. It deals with the many dialects, slang terms, and grammatical inconsistencies in everyday interactions. Companies use it for several automated tasks, such as to:

- Processing, analyzing, and archiving huge documents.
- Analyzing contact center records or customer feedback.
- -Using chatbots for automated customer service
- -Respond to who, what, when, and where inquiries.
- -Sort and retrieve text

(What Is NLP? - Natural Language Processing Explained - AWS, n.d.)

NLP may be used in systems that interact with consumers to improve customer communication. For instance, a chatbot filters and analyzes consumer inquiries, automatically answering simple inquiries and referring more complicated ones to customer care. This automation lowers expenses, frees agents' time from repetitive inquiries, and boosts client satisfaction. (What Is NLP? - Natural Language Processing Explained - AWS, n.d.)

2.3 Natural Language Understanding

NLU is considered the future of AI, the Holy Grail of AI, since language is the quintessence of human intelligence (Lenci, 2023). As far as today’s technology goes, computers still cannot fully understand human language (Lenci, 2023). As mentioned above, NLPs do not have brains; they have programs that allow them to learn data input from humans. Computers simply cannot learn, think, or dream yet because they lack real human brains. For computers to become more like humans in intelligence and abilities, they must understand how humans talk and think (Simplilearn, 2022). In simpler terms, they need to understand us. That is when NLU comes into action.

2.3.1 NLU Overview

Among the various definitions of NLU, the simplest one is the ability of a computer to understand human language. NLU is a subset of NLP that focuses on analyzing the meaning behind text/ sentences (Akash Takyar, 2023). NLU enables the software to analyze words with multiple meanings or to locate comparable meanings in various texts (Akash Takyar, 2023). NLU can serve various functions, from facilitating human-to-human communication to enhancing technical support in human-machine communication (Canonico & De Russis, n.d.). It’s used in various consumer-facing applications such as voice assistants, automated translation services, chatbots, web search engines, etc. Currently, conversational agents handle the first inquiries for technical support (Canonico & De Russis, n.d.). With language serving as a bridge between technology and users in the future, NLU's development highlights its importance in advancing human-computer interaction (Standford Online, 2023). The entire industry related to web search is being reshaped around NLU technologies (Standford Online, 2023).

2.4 Natural Language Generation

NLG, Natural language generation, heralded as a crucial counterpart to NLU in advancing artificial intelligence, represents the art of transforming structured data into human-like natural language text. Just as NLU strives to imbue machines with the ability to understand and interpret human language, NLG endeavors to equip them with the capacity to communicate in a manner that mirrors human intelligence. NLG conducts information extraction and retrieval, sentiment analysis, and more. While NLP systems lack the cognitive prowess of human brains, NLG bridges the divide by enabling machines to generate coherent narratives, summaries, and explanations from data inputs. NLG serves a multifaceted role, from interpreting data and generating content to supporting decision-making and enhancing user interactions. NLG aims to make machines speak and write in ways that humans can comprehend,

marking a significant stride toward AI's aspiration to emulate human-like language and intelligence. (Paramita Ghosh, 2022.)

NLG is a technique that enables machines to generate coherent and contextually appropriate narratives, reports, or explanations in natural language, bridging the gap between unstructured/ raw data and text that humans understand (Samyuktha jadagi, 2023). In simpler words, Natural language generation turns computer-readable data into human-readable text. This area of artificial intelligence has found several applications in various fields, providing insightful data, automating content production, and improving communication between humans and machines. It has become a crucial element in artificial intelligence in the age of data-driven decision-making and cutting-edge technology.

2.5 Language Models

In the rapidly evolving landscape of artificial intelligence and natural language processing, cutting-edge technologies like Language Models (LMs) have emerged as a transformative force (Owais, 2023). These intelligent systems, rooted in the power of deep learning and neural networks (Samant et al., 2022), are crucial to unlocking the full potential of human-computer interaction. A language model is a natural language probabilistic model trained on text corpora in one or more languages that can estimate the probability of a set of words (FutureBeeAI, 2023).

LMs have revolutionized how we understand, process, and generate human language, making significant strides in language translation, sentiment analysis, text summarization, and more (Liu et al., 2021). Language models have found applications across numerous fields, including natural language processing, chatbots, virtual assistants, and content recommendation, reshaping how we interact and harness the power of written and spoken language. That is why language models play a central role in enabling the capabilities and

advancements in modern NLP, making them a crucial and integral part of NLP technology, the backbone of Modern NLP (Lipenkova, 2022).

Alongside its evolving landscape, Language model (LM) research has progressed through four significant stages of development (Figure 4). The first stage introduced Statistical Language Models (SLMs), like n-gram models, which estimate word likelihood based on previous work occurrences. The second stage brought Neural Language Models (NLMs), employing neural networks, particularly RNNs like LSTM and GRUs (Gated recurrent unit), to predict the next word's probability based on preceding words. The third stage introduced Pre-trained Language Models (PLMs), which use neural networks to capture word meaning and context in vector representations, including ELMo and BERT. The fourth stage witnessed the emergence of Large Language Models (LLMs), such as GPT-3 and GPT-4, trained on vast text data and excelling in various natural language processing tasks. These four stages of LM development represent significant advancements in the field, with each stage building upon the previous one and pushing the boundaries of what machines can achieve in NLP and Computer Vision. (Hadi et al., 2023.)

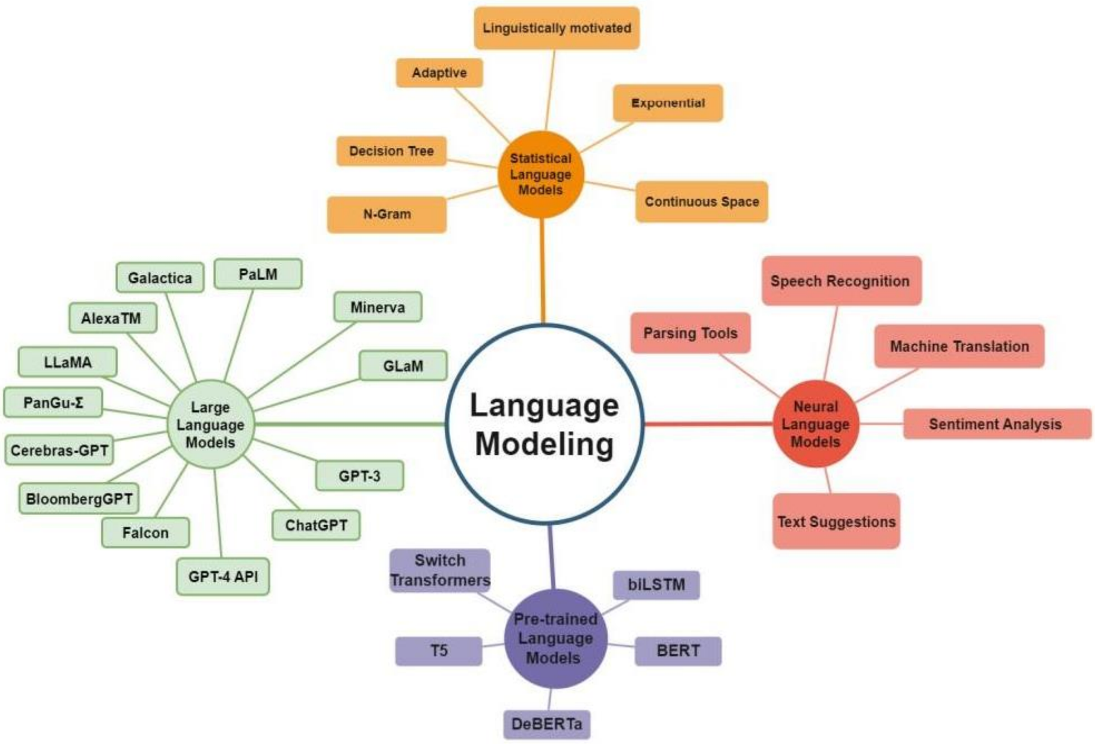


Figure 4. Development/ Stages of Language Models (Hadi et al., 2023)

However, in this thesis, the main focus is on LLMs. LLMs have revolutionized artificial intelligence and have found use in various fields, including content generation, knowledge dissemination, and communication. The history, education, and operation of LLM are briefly covered in the following section.

2.6 Large Language Model

LLMs are AI and Machine learning models that generate and process natural language texts (What Is a Large Language Model (LLM)?, 2022). They are created based on massive amounts of large bodies of text data to generate numerous outputs for NLP tasks using deep learning techniques to learn patterns and structures of languages (What Is a Large Language Model (LLM)?, 2022). The history of LLMs can be traced back to the earlier days of NLP research (Figure 5). NLP researchers have always aimed to develop algorithms and models to understand and generate human language. However, as we know them today, significant advancements in LLMs gained momentum in the mid-2010s.

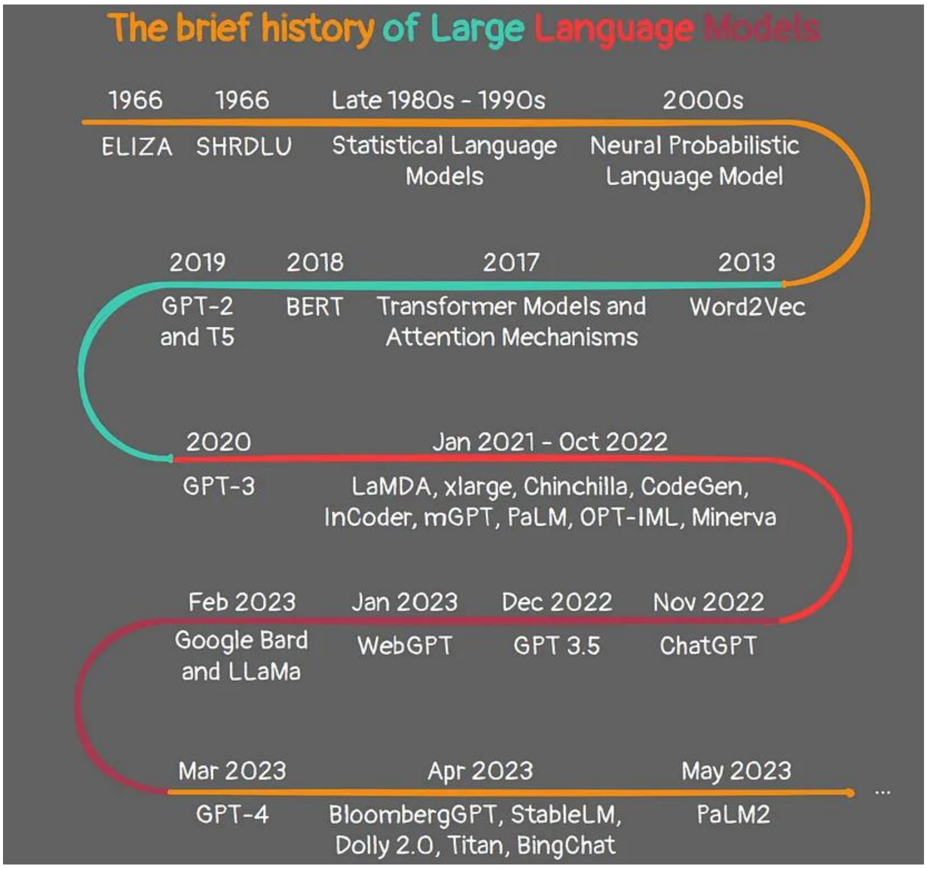


Figure 5. A brief history of LLMs (Armin Norouzi, 2023)

2.6.1 History

1950- 1960s:

As mentioned earlier, the first language model was developed in the 1950s to translate Russian sentences to English automatically. The linguistic features and rules used by these rule-based algorithms to analyze language were manually created, in other words, handcrafted. They had a restricted range of capabilities and could not manage how intricate NLP was. (Hadi et al., 2023.)

An early example of computer software created to simulate conversation with a human user is Eliza. Joseph Weizenbaum, a computer scientist at MIT, developed it in the middle of the 1960s (Tomáš Zemčík, 2019). One of the earliest conversational agents or chatbots is frequently referred to as Eliza. Eliza communicated with people via text using basic pattern matching and rephrasing strategies (Armin Norouzi, 2023). Asking open-ended inquiries and reflecting user statements was primarily created to emulate the conversational approach of a Rogerian psychotherapist (Weizenbaum, 1966; Armin Norouzi, 2023). Despite the fact that Eliza's answers were relatively basic by modern standards, they were revolutionary at the time. They generated attention in the fields of natural language processing and artificial intelligence (Armin Norouzi, 2023).

1980 – 1990:

In the 1980s and 1990s, statistical language models were developed. Using probabilistic methods, these models determined the likelihood of a certain collection of words in a given circumstance. They could handle higher data quantities and were more accurate than rule-based models. However, they could still not properly understand the language's semantics and context. (Juang & Rabiner, 2004.)

Mid- 2010s:

The creation of neural language models in the middle of the 2010s marked the subsequent significant advance in language modeling. These models Extracted language structures and patterns from significant text data using deep learning techniques. The recurrent neural network language model (RNNLM),

which was created in 2010, was the first neural language model. Compared to earlier models, RNNLM could predict word context and generate more naturally sounding text. (Hadi et al., 2023.)

2015- 2017:

Google unveiled the Google Neural Machine Translation (GNMT) technology in 2015 as the first extensive neural language model. This model could perform at the cutting edge on jobs requiring machine translation since it was trained on vast multilingual text data. With the release of the Transformer model in 2017, the advancement of LLMs proceeded. The Transformer made it feasible to train considerably bigger models by allowing parallel training on several Graphical Processing Units (GPUs) and learning the longer-term relationships in language. (Hadi et al., 2023.)

2018- 2020:

With its transformer-based design, the 2018 release of OpenAI's GPT-1 represented a significant advancement in NLP as it was their first iteration of a language model using the Transformer architecture. GPT-1 showed the potential of transformers to revolutionize NLP tasks by producing contextually meaningful phrases with 117 million characteristics. It was trained utilizing a two-step procedure that involved supervised fine-tuning followed by unsupervised pre-training. This approach attracted much interest from academic and scientific circles. Despite its flaws, GPT-1 paved the way for later, more potent models, ushering in a new era of AI research and fierce competition in the field of LLMs. (Hadi et al., 2023.)

Google's BERT (Bidirectional Encoder Representations from Transformers), introduced in 2018, revolutionized NLP with its bidirectional context approach, influencing subsequent LLMs. Several significant Large Language Models (LLMs) have made notable contributions to natural language processing (NLP). OpenAI's earlier model, GPT-2, with 1.5 billion parameters, was unveiled in 2019 and initially withheld due to concerns about misuse. Among them, GPT-3, developed by OpenAI in 2020 with 175 billion parameters, garnered immense attention for its exceptional capabilities across diverse NLP

tasks. Its last knowledge update was in September 2021. (Armin Norouzi, 2023.)

Alongside GPT-3, other significant LLMs, such as XLNet, ERNIE, T5, RoBERTa, and ALBERT, had already made substantial contributions to NLP, each offering unique advancements in pre-training, fine-tuning, context comprehension, and task versatility. These models collectively shaped the landscape of large-scale language understanding and generation, setting the stage for further developments in the field. (Kalyan, 2024.)

2022- Today:
The GPT-3 architecture, introduced in 2020, has upgraded to GPT-3.5 architecture, garnering significant attention in the AI community. This cutting-edge language model is built upon a transformer neural network, a deep learning model that has revolutionized the field of NLP. (Kalyan, 2024.)

In contrast to its predecessor, GPT-3.5 features notable differences, including 3 model variants, each with 1.3B, 6B, and 175B parameters. Starting with a reduced parameter count of 1.3 billion and a design that enables it to operate within frameworks based on human values. GPT-3.5 represents an enhanced version of GPT-3, with the ability to comprehend and generate natural language and code. The same datasets used for GPT-3 are used to train GPT-3.5, but with the additional fine-tuning process to incorporate "reinforcement learning with human feedback," or RLHF to the GPT-3 model. In 2023, Bard was launched. Bard is an LLM chatbot developed by Google AI.

Google Bard is newer, and some say it is a more powerful large language model (LLM) than GPT-3.5. It is based on the Pathways Language Model (PaLM) architecture, which is more complex and computationally expensive to train than the Generative Pre-trained Transformer (GPT) architecture used by GPT-3.5. However, PaLM also gives Bard several improvements from any previous LLM created, Including:

- Better performance on a wider range of tasks: Bard can generate more accurate and up-to-date information, translate languages more accurately, and write more creative text formats.
- Real-time access to the internet: This allows Bard to provide more up-to-date and relevant information than GPT-3.5.
- Image-based responses: Bard can generate image-based responses, while most other LLM/ GPT do not.
- Capacity to read responses aloud: Bard has native text-to-speech capabilities.

It is better at generating accurate and up-to-date information and can be used for a broader range of tasks. (Dhaduk, 2023.)

In the same year, 2023, OpenAI launched their most recent and advanced language model yet, GPT-4. GPT-4 is larger and more powerful than GPT-3, with the ability to generate, understand, and process even more coherent and natural-sounding text (Hadi et al., 2023). GPT-4 is an enormous model with 175 trillion parameters, making it the most extensive available. GPT-4 uses a neural network structure based on the transformer model. This architecture enhances comprehension of word relationships within text. Additionally, it integrates an attention mechanism, enabling the neural network to discern the significance of different data segments (Wagh, 2023).

With the great success seen from these LLMs, many organizations have developed newer and more accessible LMs in the past few years. Numerous researchers and organizations have even created open-source, freely available models for people to use and contribute to, fostering collaboration and knowledge sharing within the AI community. This open-access approach has helped democratize access to advanced AI technologies and led to the development of various applications and solutions across various domains. As AI research continues to evolve, we can expect to witness even more sophisticated and versatile language models in the future. (Hadi et al., 2023.)

2.6.2 How LLMs work

LLMs are, as referred to before, a type of AI that can mimic human intelligence. They work by processing and comprehending enormous amounts of text data using advanced statistical models and deep learning techniques. These models learn the intricate patterns and relationships in the data, enabling them to generate new content that closely resembles the style and characteristics of a specific author or genre. (Hadi et al., 2023.)

The process starts with pre-training, shown in Figure 6, during which the LLM is exposed to a huge corpus of material from numerous sources, including books, papers, and websites. Using context from the words that come before it, the model learns to predict the next word in a phrase through unsupervised learning. Because of this, the model will develop an understanding of the relationship between grammar, syntax, and semantics.

Step 1:
In the pre-training pipeline, the first step is the pre-training corpus, which refers to the collection of text data used to train the model before moving into the next step of the procedure. Pre-training corpus sources can be roughly divided into two categories: general data and specialized data. It is essential to preprocess the data to create the pre-training corpus when a significant amount of text data has been collected, mainly by removing pointless/ unnecessary, noisy, redundant, and potentially harmful information. Normalize the text, including converting it to lowercase and handling special characters and punctuation. (Hadi et al., 2023.)

Step 2:
Quality filtering is crucial in the second step of the data processing pipeline for pre-training LLMs. Here, low-quality, and irrelevant/unwanted data is removed from the training corpus using techniques such as language filtering, statistic filtering, and keyword filtering. This step is vital for refining the dataset and preventing the model from learning noise or biases, ensuring the LLM

generates high-quality, contextually relevant text in subsequent NLP tasks. (Hadi et al., 2023.)

Step 3

The third stage is deduplication, crucial in improving language models. Previous research has revealed that duplicate data in a corpus can significantly diminish the diversity of these models, leading to an unstable training process and, consequently, a negative impact on model performance (Hadi et al., 2023).

By effectively removing duplicate content, Deduplication enables us to train models that produce memorized text far less frequently, ultimately reducing the number of training steps required to achieve the same or even better accuracy. Additionally, this process helps to reduce train-test overlap, a problem that affects over 4% of the validation set in standard datasets, thereby facilitating more accurate model evaluation. (Lee et al., 2021.)

Step 4

In the fourth step of privacy reduction, privacy issues related to the pre-training of language models using web-based data must be addressed. There is a chance of privacy violations since this data frequently includes user-generated material that contains private or sensitive information. Consequently, privacy redaction procedures must be used to exclude personally identifiable information (PII) from the pre-training corpus. (Hadi et al., 2023.)

Step 5

Tokenization, the final stage in data preprocessing, is pivotal in transforming raw text into a format suitable for large language models (LLMs). Its primary objective is dissecting the unprocessed text into a series of individual tokens, creating a structured input with which LLMs can effectively understand and work. This crucial step ensures that LLMs can process and generate text precisely, making it an integral part of natural language processing workflows. (Hadi et al., 2023.)

Step 6

Following the initial pre-training phase, the Language Model (LLM) undergoes a process known as fine-tuning. This entails training the model on a specific task or within a particular domain. During this phase, labeled examples are presented to the model, guiding it to produce more precise and contextually fitting outputs for the assigned task. Fine-tuning enables the LLM to specialize in diverse applications such as language translation, question-answering, or text generation. The success of the LLM depends on its capacity to capture the statistical patterns and subtle linguistic nuances within the training data. By comprehensively understanding language, LLMs can generate coherent and contextually relevant responses when analyzing extensive amounts of text or data. During inference, a user inputs a prompt or query to interact with the LLM. The model processes this input and delivers a response based on its knowledge and understanding of the user's context. The generated response employs probabilistic methods, considering the likelihood of various words or phrases given the input context. (Hadi et al., 2023.)

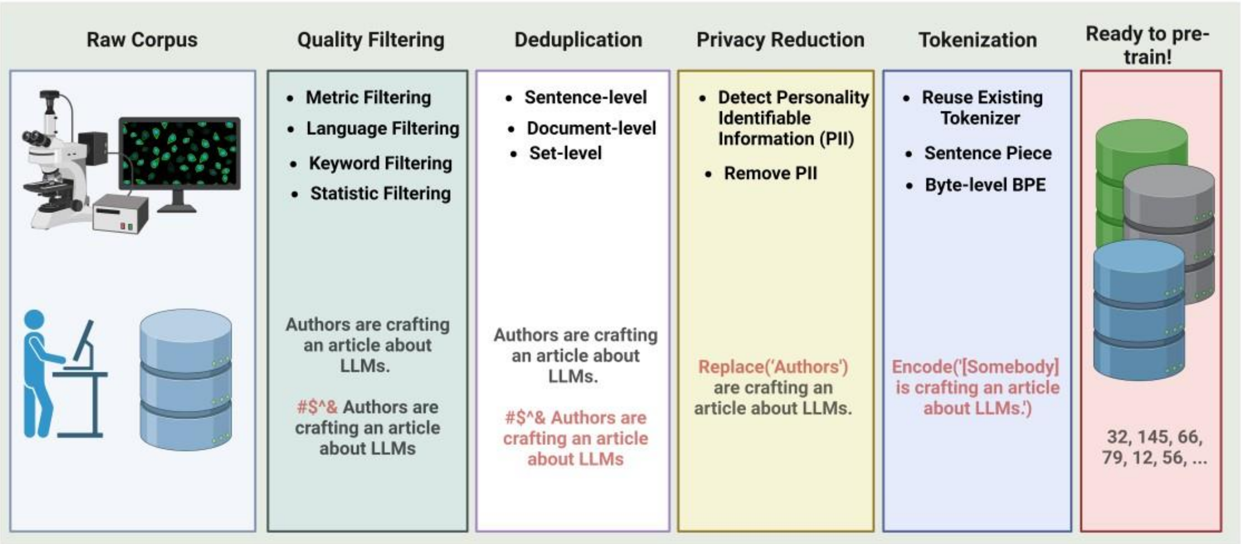


Figure 6. Data processing pipeline for pre-training LLMs (Hadi et al., 2023)

2.6.3 Transformers

Transformers is a deep learning model introduced in 2017 and has gained widespread acceptance across various fields, such as NLP, computer vision (CV), speech processing (Vaswani et al., 2017). Initially designed in 2014 as a sequence-to-sequence model for machine translation, subsequent research has demonstrated the effectiveness of Transformer-based pre-trained models (PTMs) in achieving state-of-the-art results across various tasks (Lin et al., 2021). Consequently, the Transformer architecture has emerged as the preferred choice in NLP, particularly for PTMs as well as other AI applications such as CV and audio processing and other disciplines such as chemistry and life sciences (Lin et al., 2021). “The Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence aligned RNNs or convolution” (Vaswani et al, 2017). Achieving state-of-the-art results in translation quality through increased parallelization (Vaswani et al., 2017). The Transformer's architecture features stacked self-attention and pointwise, fully connected layers in both the encoder and decoder (Vaswani et al., 2017).

Types of Transformers

The Vanilla Transformer

The Vanilla Transformer, introduced in 2017 as the original transformer model, forms a foundational architecture in deep learning designed explicitly for sequence-to-sequence applications like machine translation (Lin et al., 2021). This model employs self-attention mechanisms, enabling dynamic weighing of sequence elements, and utilizes an encoder-decoder architecture for bidirectional processing (Lin et al., 2021). Vanilla Transformers have shown themselves versatile beyond their original concept, impacting various real-world applications. This includes incorporating positional encoding to comprehend the sequential order and multi-head attention for gathering contextual information (Lin et al., 2021). In parallel, the Transformer model presents a groundbreaking

departure from traditional recurrence, relying solely on self-attention to establish global dependencies between inputs and outputs (Vaswani et al., 2017).

Operating as a sequence-to-sequence model, further refines this architecture with stacked, identical blocks, each housing a multi-head self-attention module and a position-wise feed-forward network in the encoder (Lin et al., 2021), as shown in the left half of Figure 7 and introducing cross-attention modules and position-wise FFNs in the decoder solidifying its role as a cornerstone in deep learning model design as shown in the left half of figure 7. This approach allows the Transformer to reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs (Vaswani et al., 2017). (Lin et al., 2021.)

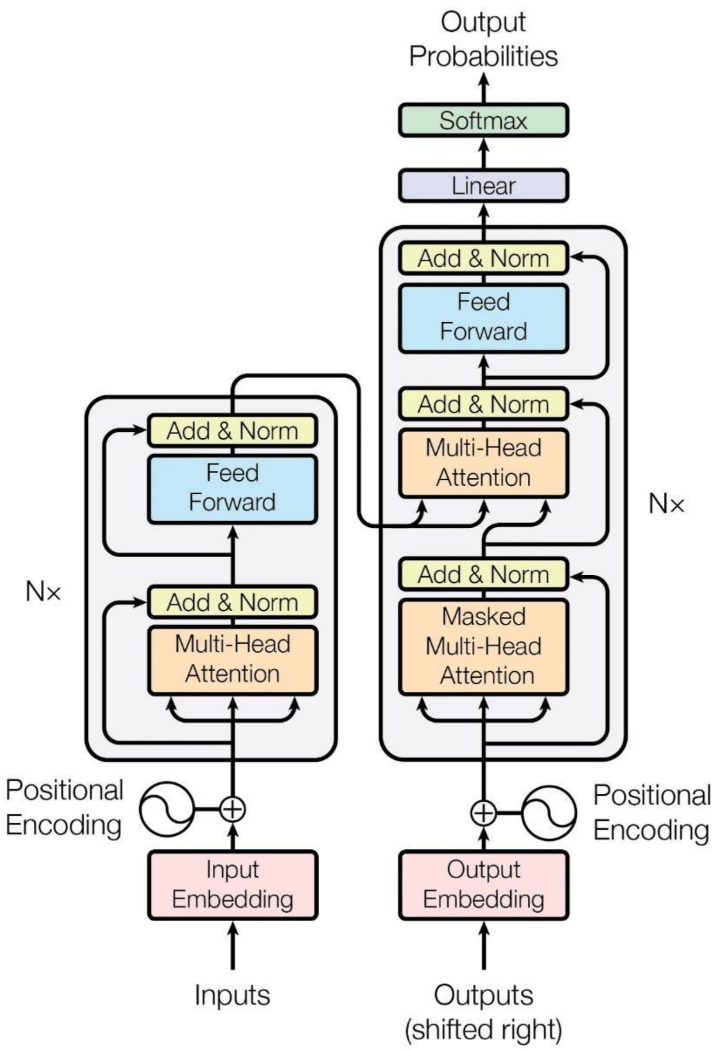


Figure 7. The vanilla Transformer – model architecture (Vaswani et al., 2017)

The Auto-regressive Transformer

Expanding upon this architecture, the auto-regressive transformer follows a similar design but introduces an autoregressive decoding strategy (Devansh, 2023). With this approach, a single output token is created at a time based on already generated tokens. This sequential generation makes AutoRegressive Transformers well-suited for applications such as language modeling and text generation, where the order of output tokens is crucial (Abideen, 2023). Auto-regressive models, unlike sequence-to-sequence models, do not require an explicit input sequence and are well-suited for text generation tasks (Devansh, 2023). Their popularity has surged, and they showcase adaptability through fine-tuning, rendering them highly beneficial across diverse applications, particularly within business environments (Devansh, 2023). “Autoregressive language models, such as GPT, GPT-2, and GPT-3, have gained significant attention for their ability to generate high-quality text and perform various language-related tasks.” (Abideen, 2023). The left part of Figure 8 shows the GPT model architecture, and the right side shows the different transformations applied to the input sequence for different fine-tuning tasks.

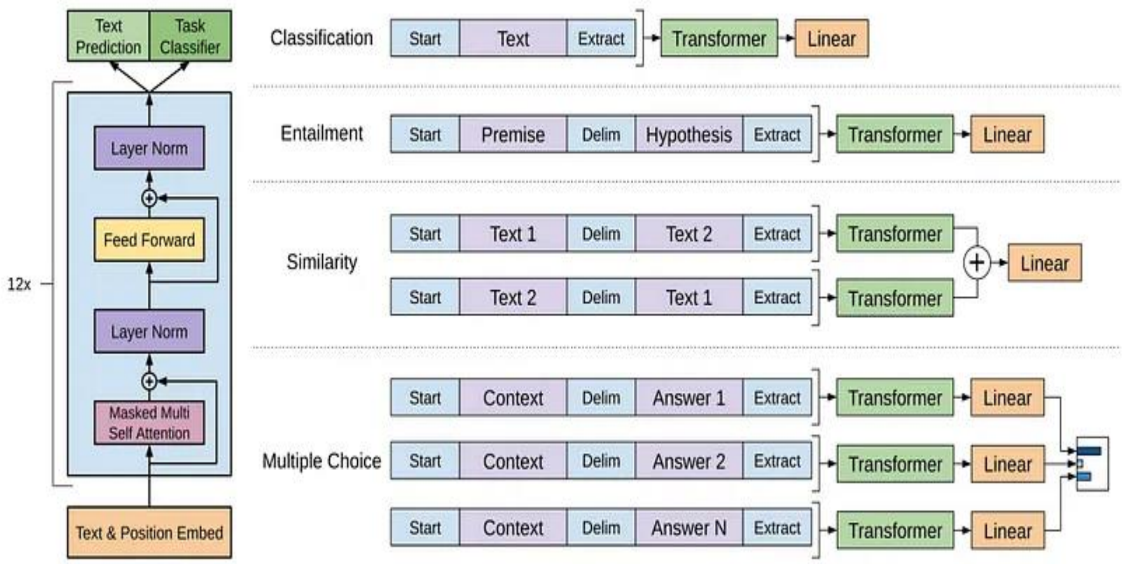


Figure 8. GPT model architecture and input sequence for finetuning tasks (Abideen, 2023)

The Auto-Encoding Transformer

Similarly, the Auto-Encoding Transformer diverges in its objective, focusing on learning a meaningful representation of input data through an encoding-decoding process (Devansh, 2023). By leveraging techniques like variational autoencoders (VAEs), the Auto-Encoding Transformer learns a continuous latent space representation of the input data (Park & Lee, 2021). This representation can then be used for data generation or feature extraction tasks. Both the Autoregressive and Auto-Encoding Transformers contribute to the versatility and applicability of the transformer architecture across diverse tasks, showcasing adaptability in sequence modeling and representation learning.

3 METHODOLOGY

The approach used in this thesis is a methodically designed combination of frameworks and technologies that are purposefully selected to accelerate the incorporation of LLMs into the Pepper robot. This section unveils the systematic approach undertaken to achieve the thesis objectives. From the computational backbone provided by the SAMK AI Virtual Server to the evaluation of diverse LLMs—Falcon, GPT4All, and OpenAI's ChatGPT to a whisper as a speech recognition model— each component plays a crucial role in shaping the advancements in HRI. A commitment to versatility, adaptability, and computational efficiency drives the deliberate selection of these tools and models.

3.1 SAMK AI Virtual Server: AI Server

For my thesis, I had the opportunity to use the SAMK AI virtual server, a pivotal resource generously provided by my university, SAMK, with the primary aim of supporting high computational power needs for my thesis. This server served as a crucial bridge between Python 2 and 3 environments, acting as the central repository for essential Python 3 codes, including components like LLM (Language Model), Whisper, and the web server, ensuring a unified and efficient development environment. The exceptional computational prowess of the server features a dedicated NVIDIA RTX A5000 GPU with 24GB of exclusive memory strategically reserved for high-performance tasks crucial for advanced artificial intelligence computations. The GPU played a pivotal role in achieving remarkably high-speed results during the development and testing stages of the thesis. Complemented by 40GB of system memory and the processing power of 5 CPUs, the server operates within a containerized environment, specifically utilizing the Ubuntu22_04_torch_2_0_1 image. It's worth noting that this robust infrastructure underscores the university's commitment to supporting innovative projects and fostering advancements in artificial intelligence.

3.2 Language Models Used

In the implementation phase of this thesis, an approach was adopted to systematically test and evaluate three LLM models to gather insights into their unique strengths and functionalities. The language models (LLMs) chosen in this thesis consist of two free, open-source models, Falcon and GPT4All, and an API-powered model utilizing the OpenAI API key. The deliberate selection of two open-source models and one API-powered model was driven by the aim to explore a broad spectrum of testing and evaluations. This strategic approach allows for a nuanced understanding of the diverse functionalities offered by each model, paving the way for a detailed analysis of their respective strengths in generating responses, contextual understanding, creativity, adaptability, resource efficiency, and speed. It's noteworthy that each model scales differently on the leaderboard, contributing to a more comprehensive assessment of their relative performance across various criteria.

3.2.1 Falcon

Falcon LLM (Large Language Model) stands out as a cutting-edge generative language model crafted by the Technology Innovation Institute (TII) in Abu Dhabi, UAE. Boasting a remarkable capacity for diverse applications, Falcon LLM excels in tasks such as text generation, translation, question answering, summarization, and code generation. The Falcon LLM family comprises three distinct models: Falcon 180B, Falcon 40B, and Falcon 7.5B, each varying in parameter count. The Falcon LLM family is characterized by open-source accessibility, high performance, and adaptability for research and commercial applications. Trained on an extensive dataset of over 3.5 trillion tokens, Falcon LLM remains under active development, continuously evolving with new features to cater to the dynamic needs of users in diverse fields. (Falcon LLM, 2023.)

I have considered using Falcon LLM as one of my LLM models in my thesis for several compelling reasons. For several compelling reasons, falcon-7B

Stands out as an ideal choice for language modeling endeavors. With its 7 billion parameters, this model strikes a harmonious balance between computational demands and performance, making it accessible and efficient on consumer-grade hardware and adaptable to various deployment scenarios. Its high accuracy is evident across various tasks, including text generation, translation, question answering, and summarization, showcasing its remarkable versatility. As an open-source project, Falcon-7B benefits from a dynamic community of developers and researchers, ensuring continuous improvement and innovation. Notably, incorporating Flash Attention technology enhances processing speed and efficiency, resulting in faster responses and smoother interactions (Tiiuae/Falcon-7b · Hugging Face, 2023).

3.2.2 GPT4All

GPT4All, developed by Nomic AI, is a comprehensive open-source ecosystem designed to seamlessly integrate Large Language Models (LLMs) into diverse applications (Anand et al., 2023). Within this innovative framework, GPT4All presents a diverse array of chatbots meticulously trained on an extensive corpus of pristine assistant data. This inclusive ecosystem empowers developers to train and deploy customized LLMs effortlessly using everyday hardware. Notably, the training process for this model involved approximately 1 million prompt-response pairs executed using the GPT3.5 Turbo OpenAI API during the period spanning from March 20, 2023, to March 26, 2023. This timeframe encapsulates the dedication to refining and enhancing the model's performance, ensuring its effectiveness and relevance in real-world applications.

In its commitment to versatility, GPT4All offers a spectrum of models tailored for both commercial and non-commercial applications. As previously highlighted, developers utilizing GPT4All gain the flexibility to train and deploy their LLMs for personal use or share them with the wider community. In this context, the GPT4All model used in this thesis is the gpt4all-falcon-q4_0.guff variant. Renowned for its exceptional speed in Generating responses, this model, trained by TII, fine-tuned by Nomic AI, and licensed for commercial use,

exemplifies the cutting-edge capabilities inherent in the GPT4All ecosystem. GPT4All Falcon (gpt4all-falcon-q4_0.gguf) is known for its fast responses and high accuracy, making it a powerful tool for various applications. The model's size in terms of storage space is 3.92 GB, and the recommended RAM required to load and run the model is 8GB.

3.2.3 OpenAI's text generation model: ChatGPT

OpenAI's text generation models, such as GPT-4 and GPT-3.5, are powerful generative pre-trained transformers that understand natural and formal language. These models respond to inputs, also known as "prompts," and can be programmed through prompt design, which typically involves providing instructions or examples. GPT-4 is versatile, handling tasks like content or code generation, summarization, conversation, and creative writing. Tokens represent common character sequences; one token is approximately four characters or 0.75 words for English text. There are limitations regarding the maximum context length for prompts and inputs, which vary for each model and can be found in the model index.

When utilizing one of these models via the OpenAI API, a user sends a request that includes the inputs and their API key. In response, they receive output generated by the model. The latest models, gpt-4 and gpt-3.5-turbo are accessed through the chat completions API endpoint.

In the context of this thesis, one of the Language Models evaluated and tested is the GPT-3.5 Turbo. Renowned for its proficiency in understanding and generating both natural language and code, this model has been selected for its versatility in completing general tasks and specific chat-based interactions. Despite the introduction of GPT-4, the continued popularity of GPT-3.5, attributed to its cost-effectiveness and enhanced processing speeds, makes it an optimal choice for current research. Recognized as the most capable model within the GPT-3.5 family, GPT-3.5 Turbo serves as a cornerstone for advancing natural language processing capabilities in the investigations undertaken.

3.3 Whisper

Whisper is an open-source robust speech recognition and transcription machine learning model created by Openai and released in September 2022. Whisper, an automatic speech recognition (ASR) system, stands out with its training on a massive and diverse dataset of 680,000 hours of noisy speech recognition data web scraped from the internet. This extensive training enhances its robustness, making it more adept at handling accents, background noise, and technical language. (Introducing Whisper, 2022.)

The architecture, as seen in Figure 9, is based on 1.5 billion parameters in a simple autoregressive encoder-decoder Transformer, also referred to as sequence-to-sequence model (Gandhi et al., 2023); the input audio is split into 30-second chunks, converted into a log-Mel spectrogram, and then passed into an encoder. A decade is trained to anticipate the appropriate text caption, intermixed with unique tokens instructing the single model to carry out multilingualism speech transcription, phrase-level timestamps, language identification, and to-English speech translation (Introducing Whisper, 2022). A log-mel spectrum represents a sound signal's frequency content, utilizing a non-linear mel scale and a logarithmic transformation to emphasize lower frequencies and compress the dynamic range. It is commonly employed in speech and audio processing, providing a compact and perceptually relevant description often used in tasks like speech recognition and music analysis. Five English variants of the OpenAI Whisper models are summarized in Table 1. Whisper's zero-shot performance notably surpasses specialized models, making 50% fewer errors. Approximately one-third of its dataset is non-English, contributing to its effectiveness in speech-to-text translation. It notably outperforms the state-of-the-art in CoVoST2 to English translation zero-shot scenarios. CoVoST2 is the Common Voice Speech Translation 2 dataset, featuring multilingual and multitask speech data aligned with Translated text to support research in automatic speech translation (Wang et al., 2020). It extends the Common Voice project, providing a valuable resource for speech recognition and translation tasks. “When scaled to this quantity of data, Whisper yields

competitive results with fully supervised, but in a zero-shot setting without any fine-tuning.” (Gandhi et al., 2023.)

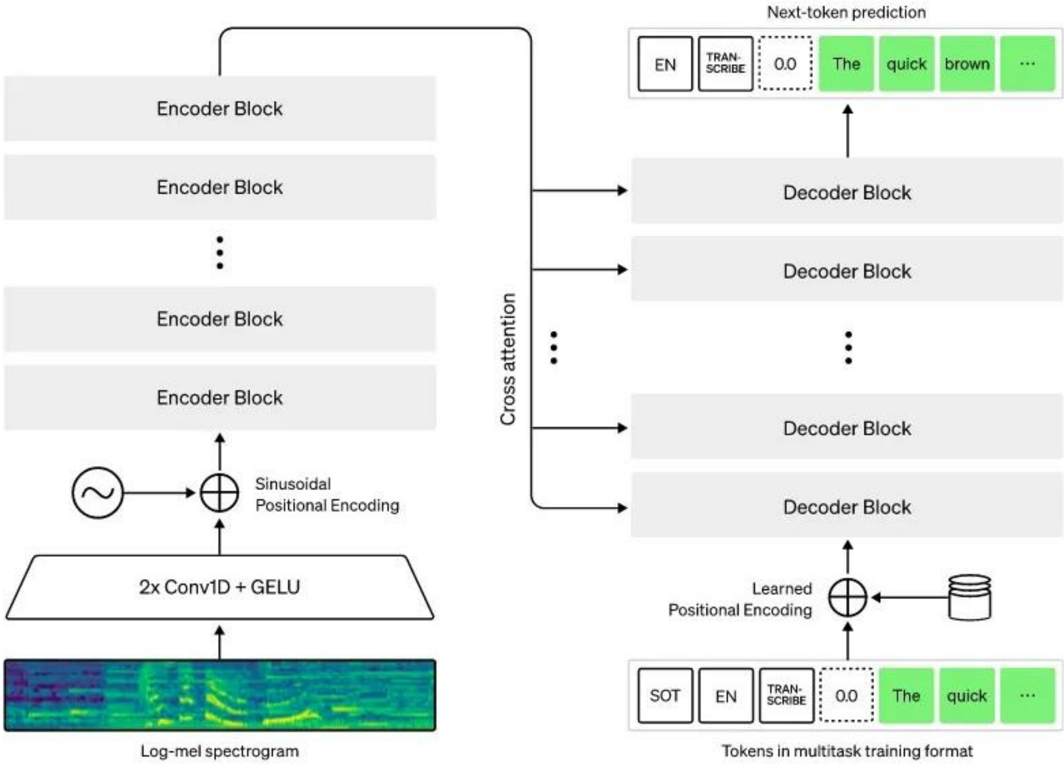


Figure 9. OpenAI Whisper - Seq2Seq Transformer architecture (*Introducing Whisper*, 2022)

Table 1. Checkpoints details of the pre-trained Whisper model family and Distil- whisper models. Distil-Whisper retains the Word Error Rate (WER) performance (Gandhi et al., 2023).

					Short Form		Long Form	
Model	Lay-ers	Width	Heads	Parame-ters	Rel. Latency	WER	Rel. Latency	WER
tiny.en	4	384	6	**39 M**	6.1	18.9	5.4	18.9
base.en	6	512	8	74 M	4.9	14.3	4.3	15.7
small.en	12	768	12	244 M	2.6	10.8	2.2	14.7
me-dium.en	24	1024	16	769 M	1.4	9.5	1.3	12.3
large-v2	32	1280	20	1550 M	1	**9.1**	1	11.7
distil-me-dium.en	24	1024	16	394 M	**6.8**	11.1	**8.5**	12.4
distil-large-v2	32	1280	20	756 M	5.8	10.1	5.8	**11.6**

Integrating an automatic speech recognition (ASR) system with Pepper's microphones is a strategic enhancement, addressing the inherent limitations of Pepper's weak speech recognition capabilities. As mentioned, Pepper's speech recognition module is constrained by its capacity to recognize only a limited set of words. Therefore, in this thesis, integrating a Whisper automatic speech recognition (ASR) system with Pepper's built-in microphones significantly enhances HRI. Pepper can better understand and respond to user commands, irrespective of accents, background noise, or technical language. This implementation improves the accuracy and efficiency of speech recognition and enhances the overall user experience with Pepper. The goal is to create a more intuitive and adaptive communication experience between users and Pepper, fostering a richer and more engaging robotic interaction.

In the experimentation phase of this thesis, a methodical strategy was employed to assess and compare four different Whisper models, forcing on their unique capabilities and features. The Whisper models selected for evaluation include the Insanely Fast Whisper from Vaibhavs10's GitHub repository, the distill Whisper model published on Hugging Face, a speech-to-text model powered by an OpenAI API key called Audio API; the ASR model used is whisper, and the open-source Whisper model by OpenAI. Similar to the selection of language models in the previous phase, this deliberate choice of diverse Whisper models aims to explore a broad spectrum of testing and evaluations. By incorporating various models, this approach facilitates understanding the distinct functionalities each Whisper model offers. This comprehensive analysis enables an in-depth examination of their speed, efficiency, and open-source transparency strengths, contributing to a well-rounded assessment of their overall performance across various criteria.

3.4 NAOqi

This thesis employed several critical modules within the NAOqi framework to enhance the capabilities of the Pepper robot in the context of HRI. The Text-to-Speech (TTS) module, accessible through ALProxy, was instrumental in

converting text into expressive and natural speech. By interfacing with the ALProxy text-to-speech service, Pepper seamlessly incorporated verbal communication into its interactions with users.

The ALProxy ALAudioRecorder module was crucial in leveraging Pepper's built-in microphones for audio capture. This module facilitated the collection and analysis of ambient audio, enabling Pepper to respond intelligently to auditory cues. Through this integration, Pepper demonstrated heightened responsiveness to the surrounding environment, a key aspect in fostering dynamic and adaptive HRI.

The AL Tablet service's functionalities were harnessed to provide a user-friendly interface. The 'ShowWebView' and 'HideWebView' features allowed for seamless content display on the table. The 'Show Input Text Dialog' feature allows a dialog of interactive content on Pepper's tablet. This integration enhanced the user experience and showcased the versatility of NAOqi in creating a seamless interface between humans and robots, empowering Pepper to engage users through dynamic visual interfaces.

Moreover, integrating Qi face detection, AL Memory, and AL Face Detection modules enabled Pepper to recognize and respond to human faces, a fundamental aspect of social interaction, adding more sophistication to Pepper's interactions. Upon detecting a human face, the Qi face detection module triggered the initiation of the user interface, creating a responsive and context-aware interaction scenario.

By leveraging these specific modules, this thesis aimed to elevate the communicative and perceptive capabilities of the Pepper robot, showcasing the practical application of cutting-edge technology in the realm of HRI.

4 IMPLEMENTATION

4.1 Experimental Design

In this section, the experimental design is delineated, outlining the comprehensive process of integrating LMs with the Pepper robot. The experiment unfolds in two primary phases: establishing a web server and configuring the Pepper robot to interact with this server seamlessly. The first phase of the design was to create a web server.

4.1.1 Setting up the Server

In the initial phase of the experiment, a web server is developed to facilitate communication between language models and the Pepper robot. This journey begins with an exploration of various LLMs to evaluate their suitability. Subsequently, a decision is made to construct a web server utilizing HTML and Flask codes, accompanied by illustrative Figure 10 showcasing what the webserver looks like.

Flask, a lightweight micro-framework for Python, is chosen for its flexibility and minimalistic design, making it well-suited for projects of varying complexity. Its simplicity empowers developers to tailor applications to their specific requirements while providing the necessary tools for building web applications or APIs. (Saini, 2024.)

Additionally, Flask's support for creating RESTful APIs enhances its suitability for projects involving backend services or client-server communication. Creating the web server is essential to establish a smooth pathway for communication exchange between the server and Pepper robot, ensuring efficient interaction and response generation. The selection of GP4ALL as the preferred LLM is based on its cost-effectiveness, delivery of high-quality responses, and efficient performance. The integration of the Whisper ASR model further

enhances the server's capabilities, ensuring seamless interaction between users and the Pepper robot.

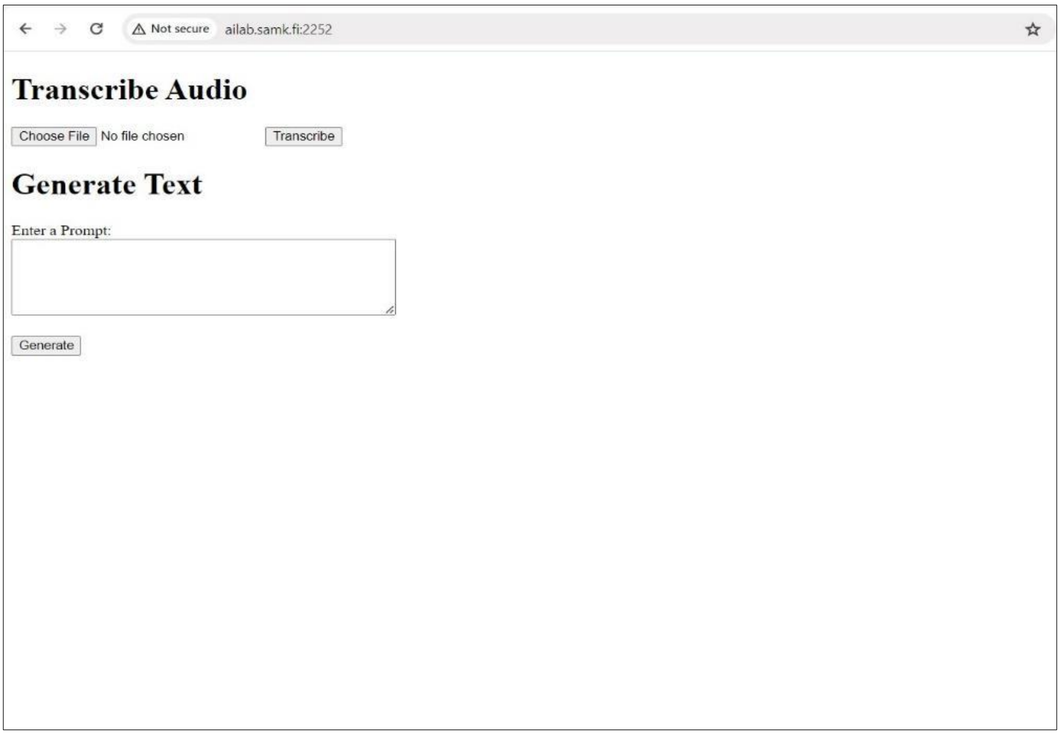


Figure 10. Image of the Webserver

4.1.2 Setting up Pepper Robot

The initial step in working with Pepper Robot was to download the software requirements and installation of essential software development kits (SDKs) and modules for utilizing Visual Studio Code with Python 2.7 for coding purposes. This sets the stage for preliminary feature testing. I explored Pepper's capabilities, encompassing activities such as making it articulate greetings, move, perform basic gestures, and test its tts.say function. `tts.say` is a function in NAOqi used to allow pepper say whatever is defined in that function using peppers speakers to output the audio.

The integration with the web server is a pivotal component of the experimental design. It encompasses the development of code enabling Pepper's seamless connection to the server. The version of Python 2.7, coupled with the request library, aligns perfectly with the compatibility requirements of Pepper's NAOqi extension modules. As the experiment progresses, attention is directed

towards the pepper’s speech recognition module, revealing its limitations. This led to the necessity for an ASR model, further influencing the integration strategy.

This experimental design is a comprehensive roadmap, providing an insightful overview of the meticulous steps undertaken to establish the web server, select the optimal LLM, and configure the Pepper robot for seamless interaction. After working with Pepper, it became clear that the operation's intricacies, the operations' brains, could not be solely orchestrated within Pepper's software but could be in the AI server. This was one of the limitations of integrating it into pepper architecture.

4.2 How it works

The system operates collaboratively as the Flask Server and the Pepper Interaction Script synergize seamlessly. On one hand, the Flask Server is responsible for serving as the backend infrastructure. It actively listens to incoming requests, intelligently generates text based on the provided input, whether textual or audio and promptly sends the generated text as a response. Simultaneously acting as the interactive front-end, the Pepper Interaction Script connects to Pepper and engages in face-detection events. When a face is detected, the input text dialog module appears and is ready for the user to use (Figure 11). For the interaction to be successful, both codes should be running first, running the flask server code, and then executing the pepper code. The culmination of this collaboration results in the visually displayed and audibly vocalized output on Pepper's tablet, enhancing the interactive experience for users, whether they input text or initiate audio recording.

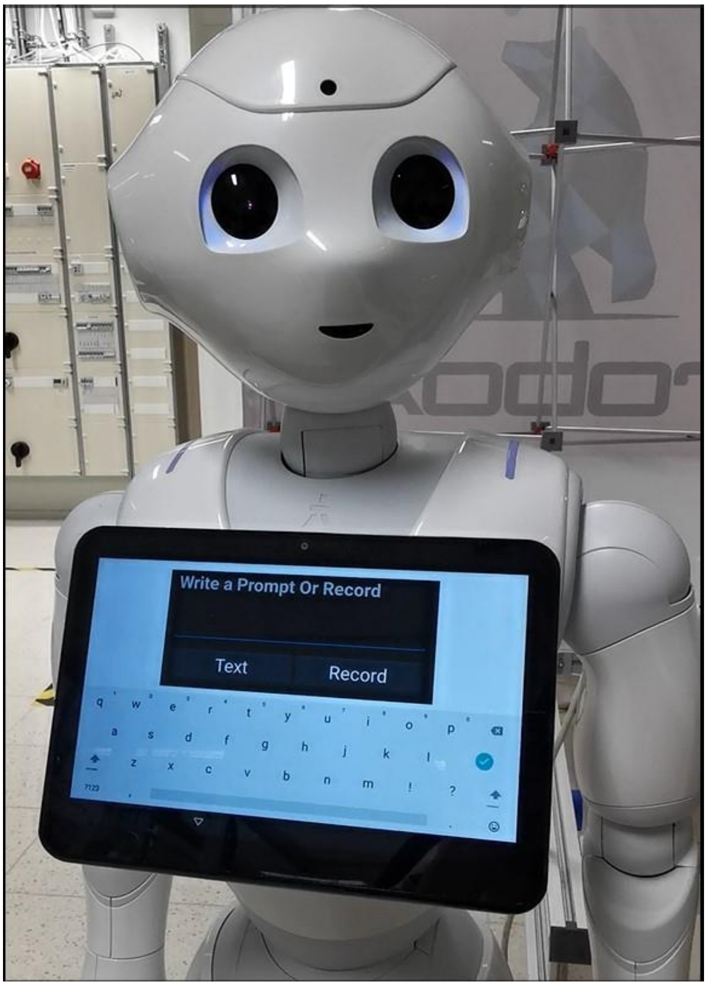


Figure 11. Pepper Input text dialog

In the case of inputting text, the user is prompted to write a prompt on Pepper's tablet; as shown in Figure 12, the user gets to write the prompt/ question they want and press Text. After the user presses Text, Peppers tablet screen will display “Generating a response, please wait” (Figure 13). While the user waits, this textual prompt is sent to the Flask Server. The Flask Server, equipped with the gpt4all-falcon-q4_0.gguf language generation model, processes the textual input, generates an appropriate response, and stores the generated text. The response, including the generated text, is then retrieved by the Pepper Interaction Script. Pepper, utilizing its text-to-speech capabilities, vocalizes the generated text to provide an audible response to the user (Figure 14).

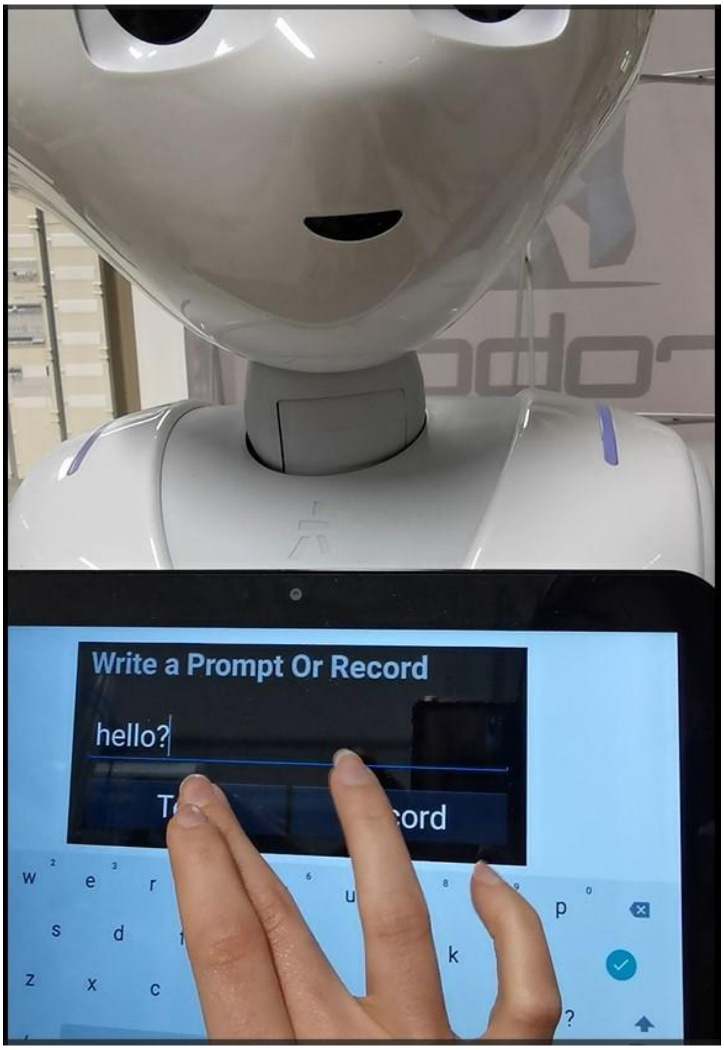


Figure 12. The user is typing in the text, “hello?”.

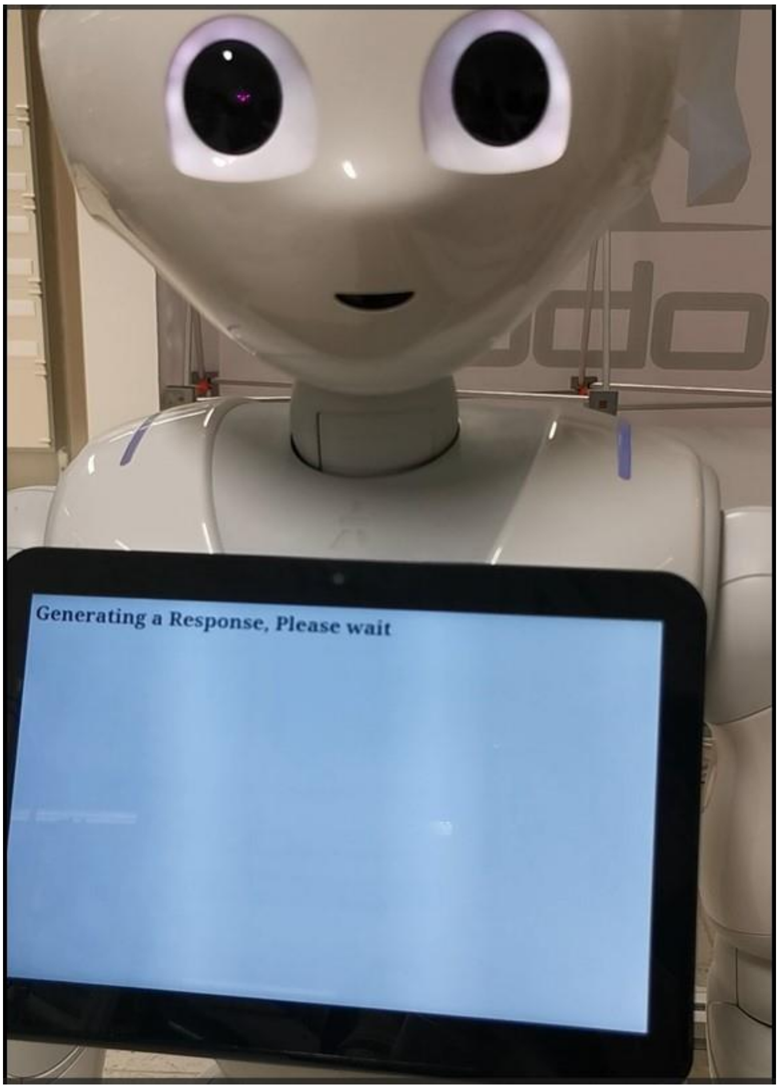


Figure 13. Pepper displaying, generating a response

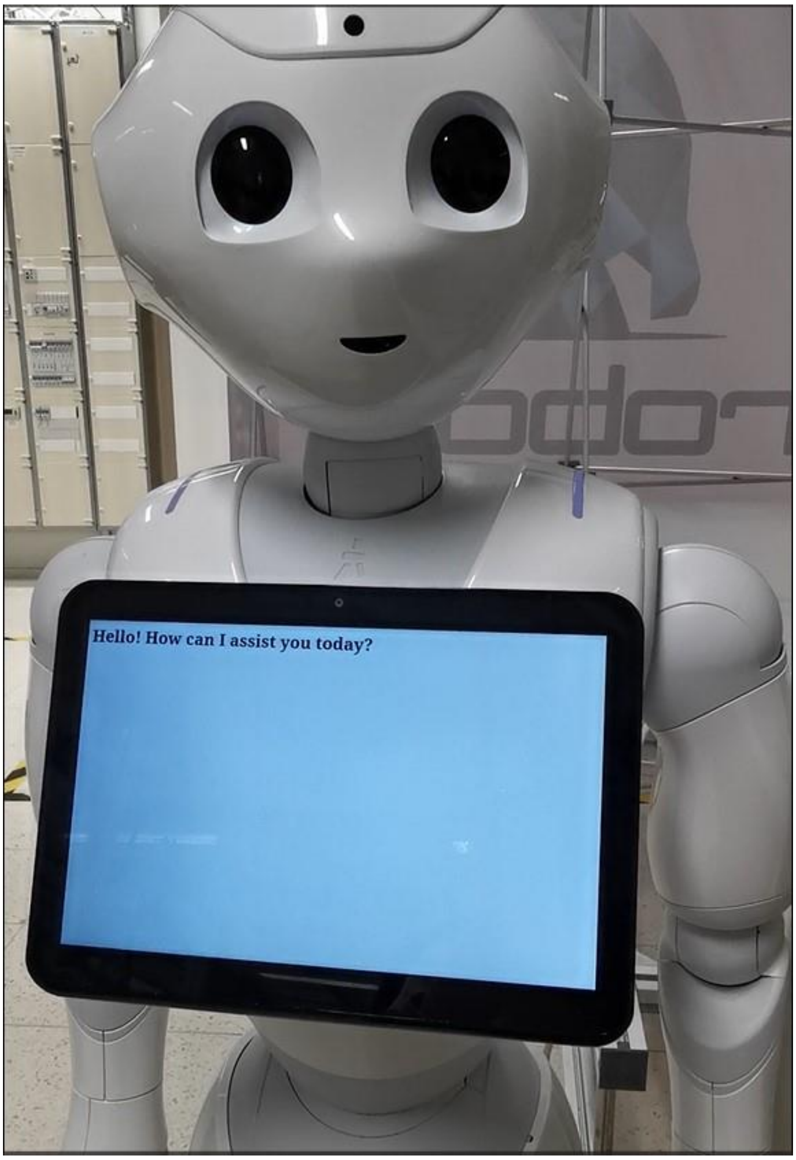


Figure 14. Generated response for Textual input

For audio recording, the process involves additional steps. In a key segment of the Pepper Interaction Script, the system utilizes Paramiko for secure SSH communication, enhancing the audio recording process. As shown in Figure 15, if the user chooses to record their voice, they will have to press the Record button on the pepper screen and have 10 seconds to record. When recording starts or stops, pepper will notify the user by showing “Recoding Started ” and “Recording stopped” on the screen. After recording, the recording stops, and the audio file is saved in the pepper system PATH. Therefore, Paramiko establishes a secure connection to a remote server, transferring the audio file using the SCP protocol. This secure and efficient transfer ensures data integrity and confidentiality. The integration of Paramiko enriches the system's

robustness, allowing secure transmission of recorded audio files for centralized processing, exemplifying best practices in handling sensitive information within the HRI framework. While all of this is happening, pepper will display, letting the user know that it's generating a response (Figure 13).

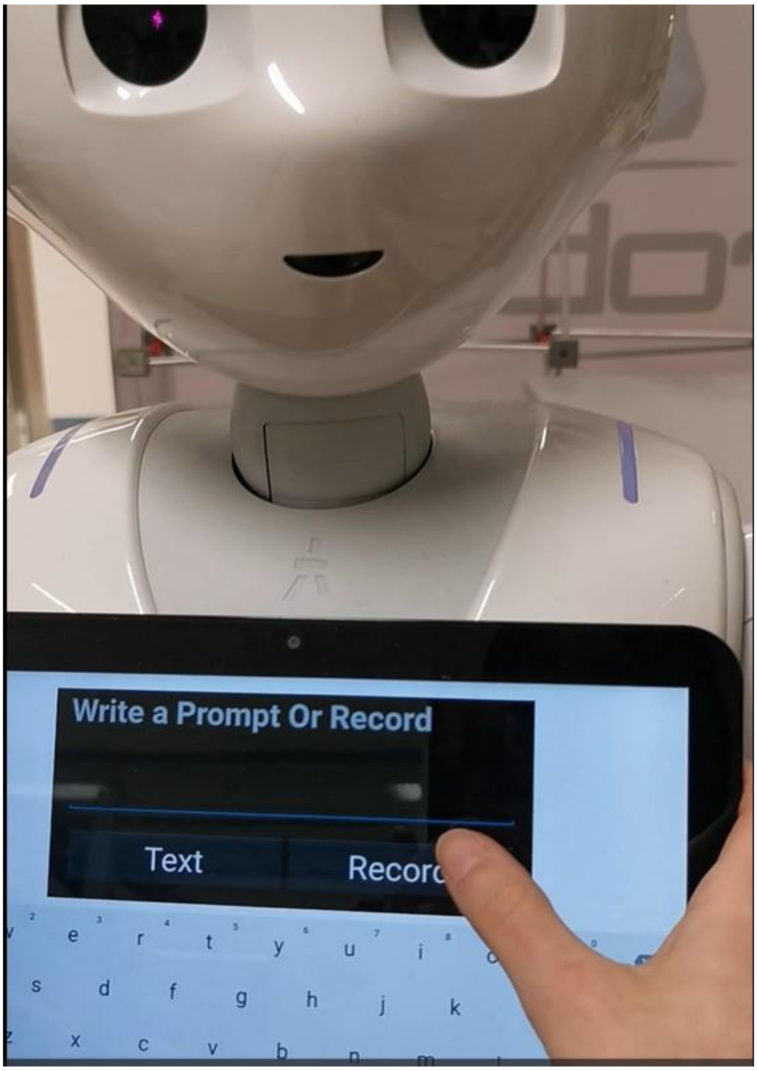


Figure 15. The User is choosing to record

After initiating the audio recording, the Pepper Interaction Script sends the recorded audio file to the Flask Server for transcription using the Whisper ASR model. The transcribed text becomes the input for the language generation model, similar to the text input scenario. In this example, the user records ask, "What is Python?". The Flask Server processes this transcribed text, generates an appropriate response, and stores the generated text. The response and

generated text are then retrieved by the Pepper Interaction Script, and Pepper vocalizes the content using its text-to-speech capabilities (Figure 16).

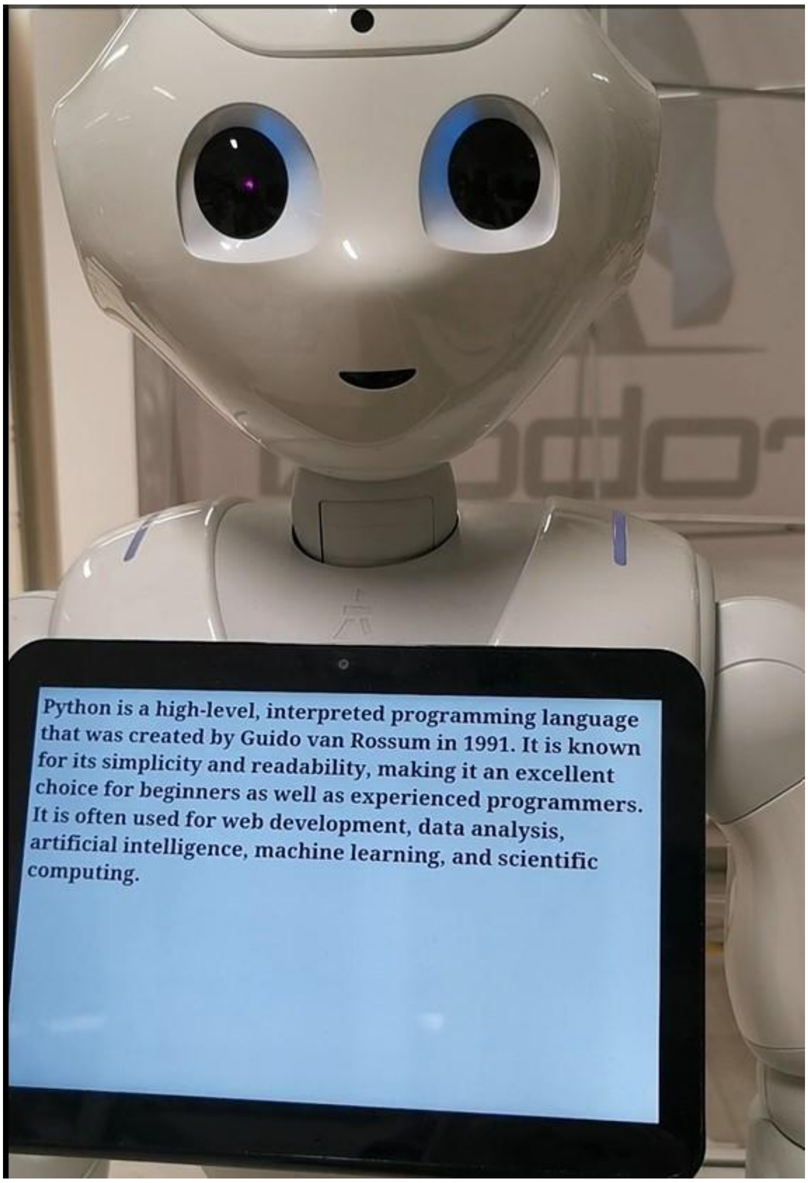


Figure 16. Pepper displays the generated text

This comprehensive process ensures a consistent interaction flow, whether the user chooses to input text or record audio, highlighting the seamless integration of text generation and speech synthesis in the overall HRI experience. Figure 17 summarizes the whole process in flow chart form.

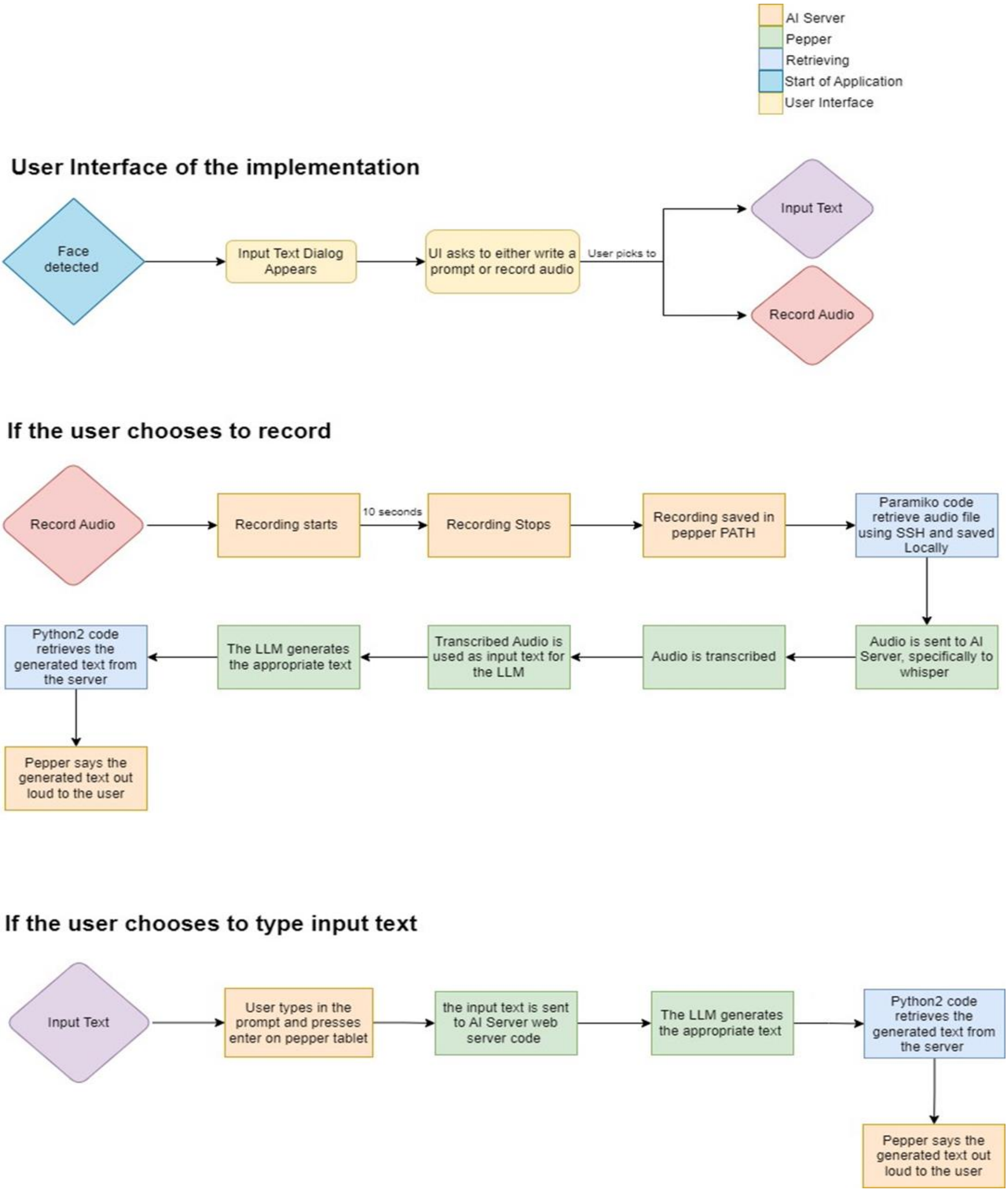


Figure 17. Flowchart of the implementation

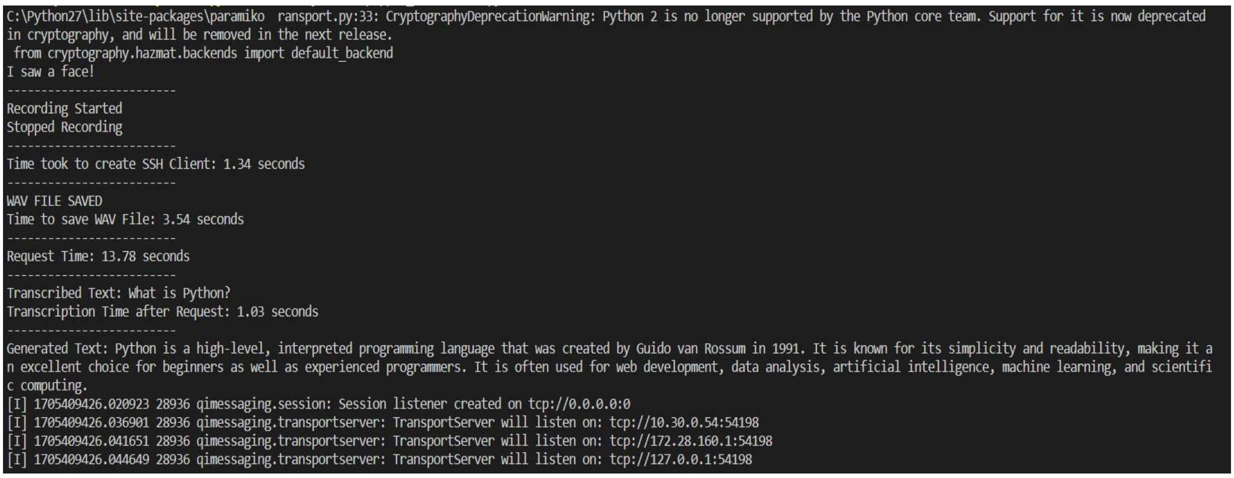
4.3 Summary of the Codes

4.3.1 Pepper Code

The Python script is designed for the Pepper robot, integrating various functionalities such as face detection, text handling, and interaction with external services. It begins by importing necessary libraries, including `qi` for the Naoqi framework, `requests` for HTTP operations, and `threading` for concurrent execution. The `HumanGreeterAndTextHandler` class initializes key attributes and establishes a connection to the Pepper robot. Methods within the class handle different aspects of interaction. For instance, the `handle_text_button` method connects to Pepper, sends input text to a Flask server, and displays or reads the generated response. The `handle_record_button` method records audio, transmits it to a server for transcription and text generation, and displays or reads the results. Multithreading is employed for parallel execution of tasks, and exception handling ensures graceful error management. The script showcases user interaction through Pepper's tablet, creating a dynamic and engaging experience by leveraging Pepper's capabilities for face detection and text-to-speech functionalities. It effectively integrates with external servers for advanced language processing tasks. Figure 18 shows the terminal output when executing peppers code if the user chooses to record, and Figure 19 shows if the user chooses to input text

Snippet of pepper code:
```
# Send Audio File for Transcription:
files = {'file': open(filename, 'rb')}
response = requests.post(server_url, files=files, ver-
ify=False)  # HTTP POST request to the server


# Process Transcription Response:
if response.status_code == 200:
    # Process the response for transcription
    response_text = response.text
    soup = BeautifulSoup(response_text, 'html.parser')

    # Extract transcribed text
    transcribed_section = soup.find('h2', text='Tran-
scribed Text:').find_next('p')
```

```
    transcribed_text = transcribed_section.get_text() if
transcribed_section else "Transcribed text not found"
    print("Transcribed Text:", transcribed_text)
    # ...

def extract_generated_text(response_text):
    generated_text_match = re.search(r'var generatedText
= (\[.*?\]);', response_text)
    if generated_text_match:
         generated_text = json.loads(gener-
ated_text_match.group(1))
         return ' '.join(generated_text)
    else:
         return None


# Use Transcribed Text as Input to Language Model:
# Extract and use generated text
generated_text = extract_generated_text(response_text)

if generated_text:
    print("Generated Text:", generated_text)
    generated_text_str = str(generated_text)
    # ...

# Use Pepper's text-to-speech module to say the generated
text
    tts = ALProxy("ALTextToSpeech", "10.30.0.53", 9559)
# Replace with the appropriate IP and port
   tts.say(generated_text_str)

else:
    print("Generated text not found in the API re-
sponse.")
```



```
C:\Python27\lib\site-packages\paramiko  ransport.py:33: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated
in cryptography, and will be removed in the next release.
 from cryptography.hazmat.backends import default_backend
I saw a face!
------------------------
Recording Started
Stopped Recording
------------------------
Time took to create SSH Client: 1.34 seconds
------------------------
WAV FILE SAVED
Time to save WAV File: 3.54 seconds
------------------------
Request Time: 13.78 seconds
------------------------
Transcribed Text: What is Python?
Transcription Time after Request: 1.03 seconds
------------------------
Generated Text: Python is a high-level, interpreted programming language that was created by Guido van Rossum in 1991. It is known for its simplicity and readability, making it a
n excellent choice for beginners as well as experienced programmers. It is often used for web development, data analysis, artificial intelligence, machine learning, and scientifi
c computing.
[I] 1705409426.020923 28936 qimessaging.session: Session listener created on tcp://0.0.0.0:0
[I] 1705409426.036901 28936 qimessaging.transportserver: TransportServer will listen on: tcp://10.30.0.54:54198
[I] 1705409426.041651 28936 qimessaging.transportserver: TransportServer will listen on: tcp://172.28.160.1:54198
[I] 1705409426.044649 28936 qimessaging.transportserver: TransportServer will listen on: tcp://127.0.0.1:54198
```

Figure 18. Terminal output of peppers code when recording


```
C:\Python27\lib\site-packages\paramiko  ransport.py:33: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated
in cryptography, and will be removed in the next release.
 from cryptography.hazmat.backends import default_backend
Input text: hello?
('Generated Text: 'Hello! How can I assist you today?')
('Generated Time: ', 4.046999931335449)
---------------------
[I] 1705409426.020923 28936 qimessaging.session: Session listener created on tcp://0.0.0.0:0
[I] 1705409426.036901 28936 qimessaging.transportserver: TransportServer will listen on: tcp://10.30.0.54:54198
[I] 1705409426.041651 28936 qimessaging.transportserver: TransportServer will listen on: tcp://172.28.160.1:54198
[I] 1705409426.044649 28936 qimessaging.transportserver: TransportServer will listen on: tcp://127.0.0.1:54198
```
Figure 19. Terminal output of peppers code when inputting text

4.3.2 Flask server Code

The Flask application is a web interface that seamlessly integrates GPT-4-All for text generation and distil whisper for automatic speech recognition. The code begins by importing essential libraries and setting up global variables, including the model’s name, port number, and maximum token limit. The utility functions handle the loading and saving of generated texts, while the `generate_text` function utilizes GPT-4-All to generate text based on input. The Flask routes are designed to handle both GET and POST requests, dynamically rendering an HTML template (`index-whis.html`) for user interaction. The application processes various types of requests, such as form submissions, file uploads, and JSON requests, employing the Whisper model for transcription and GPT-4-All for text generation. Results, including model name, input text and generated text, timestamps, and time, are stored in a JSON file as soon as the LLM generates the response; see Figure 20 to see the JSON format saved. The application utilizes Flask templates to render dynamic content, creating an engaging user experience. Upon execution, the Flask server initializes, making the web interface accessible, enabling users to input text or upload audio files and receive transcriptions and text generation results, all displayed dynamically on the web page.

For context, the variation in time occurs due to the request line code. It takes 1.409 seconds, figure 18, for the server to generate a response, while it takes 4.0469 seconds for urllib to open, request, and generate the response (Figure

19). Therefore, the total time for sending input text and retrieving generated text is 2.6379 seconds (4.0469 - 1.409).


```json
    {
        "model": "gpt4all-falcon-q4_0.gguf",
        "input_text": "hello?",
        "generated_text": "\nHello! How can I assist you today?",
        "generatedAt": "2024-01-16T15:33:12.380992+02:00",
        "generatedTime": 1.4094276428222656
    }
```


Figure 20. Saved JSON format example

Snippets of Flask server code:

```python
app = Flask(__name__)

model_name = "gpt4all-falcon-q4_0.gguf"
port_num = ####
max_token_num = 100
say = whisper.load_model("base")

# Create an empty list to store the generated texts
generated_texts = []

def load_generated_texts():
    try:
        with open('generated_texts.json', 'r') as file:
            return json.load(file)
    except FileNotFoundError:
        return []

def save_generated_texts():
    with open('generated_texts.json', 'w') as file:
        json.dump(generated_texts, file, indent=4)

def generate_text(input_text):
   # ...
    return generated_text, generated_time, gener-
ated_text1  # Return the generated text and generation
time

@app.before_first_request
def setup():
    # ...

# Combined route for both GET and POST requests
@app.route('/', methods=['GET', 'POST'])
def combined_route():
```

```
    transcribed_text = None
    transcription_time = 0

    if request.method == 'POST':
        if 'input_text' in request.form:
            input_text = request.form['input_text']
        elif 'file' in request.files:
            # If a file is uploaded, transcribe it, and
use the text
            file = request.files['file']
            if file.filename != '':
                # ... (Other code for handling file up-
loads and transcription)

                generated_text, generated_time, gener-
ated_text1 = generate_text(input_text)
            else:
                return "No selected file"
        else:
            input_text = request.json['input_text']

        # Generate text using GPT-4-All
        generated_text, generated_time, generated_text1 =
generate_text(input_text)

        # ... (Other code for storing generated text and
rendering the template)

        if request.content_type == 'application/json':
            return jsonify(text_entry), 200
        else:
            return render_template('index-whis.html',
transcribed_text=transcribed_text,
                                   generated_text=gener-
ated_text, generated_time=generated_time,
                                   transcrip-
tion_time=transcription_time)

    return render_template('index-whis.html')


if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=port_num)
```

4.3.3 Connection between Pepper robots code and Flask Server Code

The synergy between the Pepper robot script and the Flask application lies in their collective effort to enhance HRI through advanced language processing and interaction capabilities. While the Pepper script focuses on real-time engagement, leveraging face detection and tablet-based interaction, the Flask application extends this capability to a broader context, incorporating automatic speech recognition (ASR) and text generation through GPT-4-All. Both systems share a common thread in using threading for parallel execution, exception handling for robustness, and seamless integration with external services. Together, they showcase a comprehensive approach to user interaction, combining the physicality of Pepper's interactions with the web-based capabilities of GPT-4-All and Whisper. This collaborative integration results in a versatile and engaging platform that leverages the strengths of both the physical robot and advanced language models for a more interactive and dynamic user experience.

4.4 Implementation Challenges and Solutions

4.4.1 Dynamic IP Address and Network Connectivity:

During the implementation phase of this project, several challenges emerged, necessitating careful investigation and resolution. One continuous issue encountered pertained to the dynamic nature of Pepper's IP address. The robot exhibited a tendency to alter its initial IP address, particularly when connecting to networks I didn’t have access to. Potential solutions explored included manual assignment of a static IP address or the implementation of a dynamic DNS solution. Additionally, attempts were made to access Pepper's settings on Pepper's tablet; unfortunately, access to this feature was constrained due to the locked state of the associated settings. Consequently, in order to alter Pepper's settings, I had to browse Pepper's IP website for configuration

adjustments when necessary. Turning pepper off and on until the correct IP address was initiated was the only solution in network transitions.
Other network-related impediments also surfaced, such as where to work with Pepper and the AI server simultaneously; connecting to a network that grants access to both systems was necessary.

4.4.2 Python Version Disparity

A significant hurdle arose from the disparity in Python versions between the Hugging Face Transformers library, which mandated Python 3.8, and Pepper's SDK, NAOqi extension module, compatible only with Python 2.7. The deprecated status of Python 2.7 presented compatibility challenges with modern libraries. Efforts were made to mitigate this discrepancy by considering a potential bridge, as elucidated by many developers. One paper titled “Language Models for Human-Robot Interaction” used ZeroMQ as a communication bridge between the Chatbot service written in Python 3.10 and NAOqi extension modules written in Python 2.7 (Billing et al., 2023). Fortunately for this thesis, the SAMK AI server was a great advantage in using it as a bridge within itself. In the server, all the Python 3x compatible codes will be located there, while all the Python 2.7 and NAOqi compatible codes will be located locally on a PC.

4.4.3 Speech Recognition Feature

Regarding the speech recognition and microphone module, I've successfully activated the microphone; however, the built-in Pepper Sound Recognition module only captures sound without actual speech recognition capabilities. This module is specifically designed to identify and respond to certain sounds in Pepper's environment. Essentially, it involves teaching Pepper to recognize specific sounds and instructing it to respond when those sounds are detected.

In selecting a suitable open-source speech recognition solution that can be integrated into the AI server, I've opted for Whisper by OpenAI. It's important

to note that any external speech recognition solution needs to be compatible with either Python 2 or Python 3. Similarly, the Language Model (LLM) must be bridged with the selected speech recognition solution in the compatible Python version.

As it turns out, Whisper is only compatible with Python 3, but I have successfully integrated it into the AI server. This ensures seamless communication and functionality within the specified Python environment.

4.4.4 Audio Implementation Challenges

To facilitate speech recognition, I explored the possibility of utilizing OpenAI Whisper and found it a promising tool. Despite encountering some issues, I believe it can be an excellent model for our purposes once the identified issues are resolved.

One notable challenge was the compatibility with Python 3, necessitating a solution for integrating the speech recognition model with Pepper's microphones, which currently operate in Python 2. I actively worked on finding a suitable approach to bridge this gap.

To address this concern, I contemplated the feasibility of connecting Whispers through the server, similar to the language model, while keeping Pepper's microphones independent in Python 2 outside the server. However, I remained uncertain about the viability of this approach, given that the microphones are inherently tied to the physical features of Pepper.

In the testing phase, I successfully incorporated the audio transcribe Whispers library into the LLM web server, observing its effectiveness. Nevertheless, a critical aspect remains unresolved – achieving real-time audio transcription. Currently, users are required to upload audio files, after which the Whispers model transcribes the content. Subsequently, the Language Model generates the corresponding text based on the transcription. I recognize the need to

enable live audio functionality and am actively exploring solutions to enhance the user experience.

4.4.5 WAV File Challenges

During my thesis work, I successfully established a connection between Python 2 code and the web server whisper module. I verified that the transcribed text could be printed on the terminal to validate the functionality. Additionally, I ensured the proper operation of Pepper's microphone by successfully recording and playing back audio.

Despite these achievements, a challenge surfaced during attempts to save the WAV file. The saved file consistently appeared empty, displaying a duration of 0:00/0:00. Once this hurdle is overcome, the next step involves testing the integration of Pepper's microphone with the whisper module and enabling Pepper to articulate the appropriate responses generated from the transcribed text. Subsequently, I plan to explore the necessary steps for developing an application that seamlessly incorporates all these components within Pepper.

In an attempt to seek assistance, I reached out to one of the developers on the Pepper chat GitHub repository. Unfortunately, I did not receive a response from them, and given the progress I have made independently, I have decided to continue with my work without relying heavily on their input for the time being.

Addressing the issue of saving the WAV file, the code I employed seemed to function partially, as Pepper successfully captured audio. However, when attempting to play back the recorded audio, the WAV files saved on my laptop exhibited a duration of 0:00, indicating an absence of sound. Despite devoting considerable time to resolving this matter, a solution remained elusive.

After days of persistent efforts, I ultimately resolved the WAV file-saving issue. The solution involved saving the audio file in Pepper's system path

("/home/nao/audio.wav") and utilizing paramiko to transfer that file from Pepper's system via SSH. Subsequently, after retrieving the audio file, I successfully saved it locally, where the Python 2 codes are located.

4.4.6 JSON File Encounters

An error was encountered when the user chose to record; the system saved the data twice in the JSON file on the server, creating a duplicate entry for each function and resulting in a slow process. This issue arose due to the implementation of two distinct request routes. Upon calling these separate routes, the system would generate entries for each route, causing a slowdown in the code execution. The problem was addressed by modifying the Flask web server Python 3 code. The route was adjusted to be accessible via both GET and POST requests to one root ("/") URL, resolving the inefficiency in the code execution.

4.4.7 Request Post Slowness

Upon resolving the JSON file issue, the system's performance saw an improvement from its previous state, although it still exhibited a moderate pace. The primary factor contributing to this moderate speed was identified as the request.post operation took a minimum of 12 seconds, depending on the audio data. Recognizing the necessity for a solution, I conducted a thorough exploration of potential fixes. Unfortunately, none of what I tried worked; Consequently, I continued utilizing the request.post, considering the possibility of a server connection issue or a similar constraint contributing to the performance delay.

In summary, these encountered challenges, meticulously documented, and addressed throughout the implementation process, contribute valuable insights to the overall understanding of the complexities inherent in integrating Pepper with modern technologies and frameworks.

4.5 Data and Results

In the following section, I comprehensively explore the data and results obtained through the chosen LLM and ASR models. This selection was carefully made based on a thorough examination of this thesis's specific needs and objectives. This introduction sets the stage for an in-depth analysis of the collected data and the ensuing results.

4.5.1 LLM Results

Choosing and evaluating the optimal language model for this thesis involves considering a balance between the time it takes to generate a response, the quality of the response, and the cost of using the model. The time it took to generate the respond also depended on the length of the generated text, for instance, for a generated text that consists of 6 words will take less time than a text that consists of 200 words. In the examples in Table 2 showing falcon-7b audio results and Table 3 showing falcon-7b input text results, it is good to note that falcon-7b is not considered a viable option in this comparison. This model might be excluded due to its longest response times, amongst the all the models, of 12.17 seconds for audio enquires (Table 2) and approximately 10 seconds for input text enquires (Table 3). As well as the confusing, questionable, and long generated text in both the transcribed audio example and input text example, potentially making it less suitable for this thesis with time and response constraints. Therefore, gpt4all-falcon-q4_0.gguf and gpt-3.5-turbo stand out as contrasting choices

Table 2. Falcon-7b Audio results

model	tiiuae/falcon-7b
transcribedAudio	Hello, I am Pepper. How can I assist you today?
inputText	-
generatedText	Hello, I am Pepper. How can I assist you today? I’m sorry, I don’t see any record of your payment. I can send you a link that you can use to make a payment and then I will resubmit your application. Pepper & I are going round and round in circles. It’s like she doesn’t even know what I’m doing. Or saying. She’s like, a robot, and I think she’s going to break my heart. So I’m going to try and make this happen in person. I’m go-ing to get my passport renewed, and then I’m going to fly to the USA and go in person to the USCIS in California. If I can get the right paperwork with me, I should be able to get my visa, and then we can start on the paperwork for Pepper’s green card. But I have no idea how much money we’ll need. I’ve been looking through the website, and it’s all a blur to me. And there are so many different categories of green card and I am going round and round in circles and I think I’m going to have to go and talk to a human to help me. But I can do that in California. I can go there and I can see someone and I can make an appointment to talk to someone and then we’ll be able to work this all out,
generatedTime /s	12.16025233

Table 3. Falcon-7b Input text results

model	tiiuae/falcon-7b
transcribedAudio	-
inputText	What is python?
generatedText	What is python? Python is a high-level, interpreted, object-oriented scripting programming language, which was designed by Guido Van Rossum and first released by him as open source in 1991. Python is a widely used language in many industries. Python can be used for web application development, software development, data analysis, scientific computing, machine learning etc. What is the use of python? Python is used for data science and data analysis. It can be also used to create GUI based applications and web apps. Python can be used for developing machine learning algorithms for data science. It can be used to write code for web applications. What are the different versions of python? There are different versions of python. - Python 3: Python 2 was the previous version and was used from 1991 to 2016. Python 2 is still being supported by the Python community. However, it is now considered as deprecated and has reached the end of its life. - Python 2: It was the first release of python. There are many features of python 2 that are no longer in the Python 3. - Python 3: It is the latest version of python. The major difference in the 2 and 3 versions is that 3 is more object-oriented than the 2 version.
generatedTime /s	10.26021338

gpt4all-falcon-q4_0.gguf, in Table 4 and Table 5, demonstrates a cost-effective solution with relatively quick response times, 2.5 seconds for audio prompt and about 12 seconds for an input text prompt. While both of the responses are grammatically and contextually correct and simply understandable for the user, the input text generated text example could require more nuanced detail, in some cases; from a more sophisticated model like gpt-3.5-turbo. On the other hand, gpt-3.5-turbo delivers a higher-quality, contextually relevant responses, showcasing its versatility for various tasks, shown in Table 6 for an audio example and Table 7 for input text example. Its generating time takes almost the same time as gpt4all-falcon-q4_0.gguf, 2.38 seconds for audio prompt and 11.18 seconds for input text enquiry. However, it comes at a cost: for input usage, 0.0030$/ 1K tokens, and output usage, 0.0060/ 1K tokens.

Table 4. gpt4all-falcon-q4_0.gguf audio results

model	gpt4all-falcon-q4_0.gguf
transcribedAudio	Hello, I am Pepper. How can I assist you today?
inputText	-
generatedText	Hello! How may I assist you today?
generatedTime /s	2.488550901

Table 5. gpt4all-falcon-q4_0.gguf input text results

model	gpt4all-falcon-q4_0.gguf
transcribedAudio	-
inputText	what is python?
generatedText	Python is a high-level, interpreted programming language that was created by Guido van Rossum in 1991. It is widely used for general-purpose programming and has gained popularity due to its simplicity, readability, and ease of use. Python's syntax is designed to be easy to learn and read, making it an ideal choice for beginners as well as experienced programmers. It can be used for a wide range of applications, including web development, data analysis, artificial intelligence, machine learning, and more.
generatedTime /s	11.90644336

Table 6. gpt3.5 turbo Audio results

model	gpt-3.5-turbo
transcribedAudio	Hello, I am Pepper. How can I assist you today?
inputText	-
generatedText	Hello Pepper! Nice to meet you. I actually don't need any assistance at the moment, but thank you for asking. Is there anything interesting you can tell me about yourself?
generatedTime /s	2.376833916

Table 7. gpt3.5 turbo input text results

model	gpt-3.5-turbo
transcribedAudio	-
inputText	what is python?
generatedText	Python is an interpreted, high-level, general-purpose programming language. It was created by Guido van Rossum in the late 1980s and released publicly in 1991. Python emphasizes code readability and simplicity, making it a popular choice for beginners and experienced programmers alike. Python is known for its clean and easy-to-understand syntax, which allows programmers to express concepts in fewer lines of code compared to other languages. It supports multiple programming paradigms, including object-oriented, procedural, and functional programming. Python has a vast standard library and a large ecosystem of third-party libraries, making it versatile for various applications such as web development, data analysis, scientific computing, artificial intelligence, machine learning, automation, and more. One key feature of Python is its focus on readability, with the use of indentation instead of brackets or parentheses to define blocks of code. This promotes coding conventions that enhance clarity and maintainability. Python is open-source and has a strong and active community, constantly contributing to its development and improvement. It runs on multiple operating systems and is widely used in both industry and academia.
generatedTime /s	11.17668533

The decision depends on the thesis's specific needs; if simplicity and budget are prioritized, gpt4all-falcon-q4_0.gguf is sufficient. Alternatively, if the thesis demands advanced language understanding and high-quality responses, the enhanced capabilities of gpt-3.5-turbo could justify the associated costs.

In the context of this thesis, gpt4all-falcon-q4.gguf from GPT4ALL emerges as the preferred option due to its open-source nature, zero-cost advantage, efficient response time, and the good quality of its generated response.

4.5.2 ASR Results

I have chosen to incorporate Distil Whisper as my Automatic Speech Recognition (ASR) system based on the promising outcomes revealed in publications by the developers of the Distil Whisper model. Illustrated in Figure 21, Distil Whisper represents a distilled version of Whisper, boasting a remarkable 49% reduction in size, a 5.8-fold increase in speed, and maintaining performance within 1% Word Error Rate (WER) on out-of-distribution (OOD) short-form audio. Notably, when tested on OOD long-form audio, Distil Whisper surpasses the performance of Whisper, which is attributed to a decrease in hallucinations and repetitions (Gandhi et al., 2023).

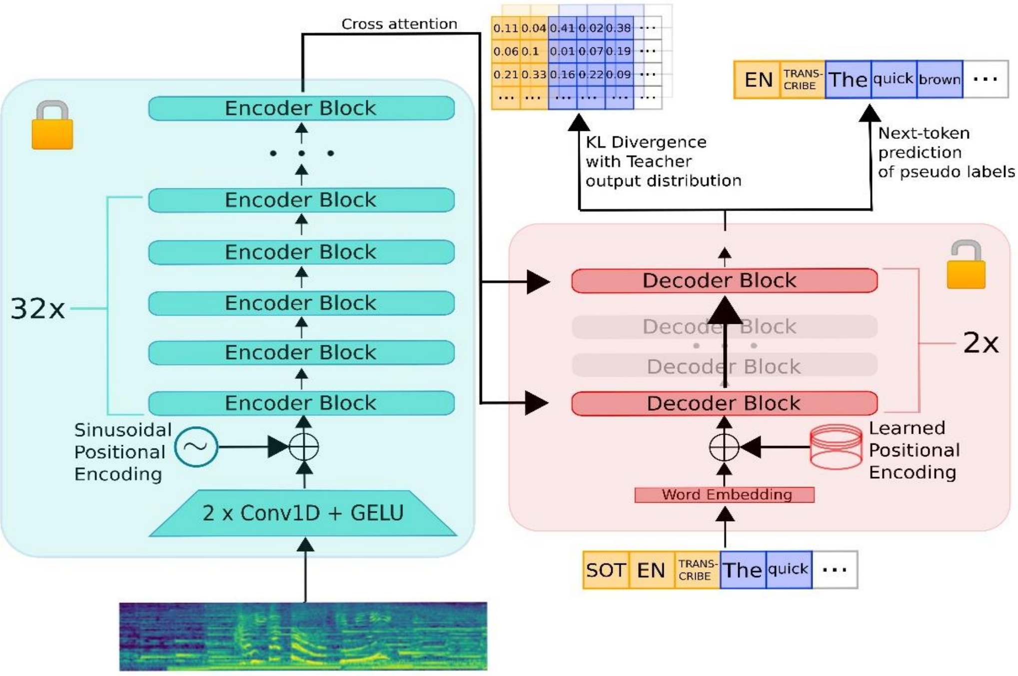


Figure 21. Distil- Whisper model architecture (Gandhi et al., 2023)

To validate these assertions and refine my ASR models, I conducted several tests to determine the most suitable speech recognition model. Table 8 provides an overview of the results obtained from various Whisper models during these tests. The evaluation involved three audio files with differing lengths and accents. My objective was to identify a model that accurately transcribes audio, considering factors such as speed, computational requirements, and associated costs.

Table 8 presents a comparative analysis of various models' performance in transcribing audio inputs, showcasing significant variations in processing times. Notably, the processing times range from exceptionally low durations, such as 1.14 seconds for the 'whisper-1' model accessed via the OPENAI API KEY, to considerably longer durations, such as 69.8 seconds for the 'distil-whisper/distil-large-v2' model running on CPU (Table 8). The differences in processing times are substantial, as shown in Table 8, with the shortest duration being approximately 1.14 seconds and the longest being approximately 69.8 seconds, indicating a 98.38% difference .

Transcribed audio: “Hello.”

Upon closer examination, it becomes evident that the choice of device and the utilization of flash attention mechanisms significantly impact processing efficiency. For instance, the 'openai/whisper-large-v2' model exhibits varying processing times depending on the activation of flash attention, with times of 58.15 seconds and 63.3 seconds observed for flash attention enabled and disabled configurations, respectively shown in table 8. Similarly, the 'distil-whisper/distil-large-v2' model demonstrates a considerable difference in processing times when run on different devices, with times of 6.17 seconds and 6.13 seconds observed for flash attention enabled and disabled configurations, respectively, when executed on CUDA:0 (Table 8).

The "openai/whisper-large-v2" model demonstrates in Table 8 varying generation times based on the flash attention setting. When flash attention is

enabled, the model achieves a relatively faster generation time of 48.83 seconds, compared to 54.1 seconds when flash attention is disabled. This variation suggests that flash attention can have a discernible impact on the model's processing efficiency.

Transcribed audio of 1:10 minutes

Similarly, the "distil-whisper/distil-large-v2" model exhibits notable differences in generation times across different configurations. While the model achieves a generation time of 55.69 seconds on a CPU without flash attention, an error occurred when flash attention was enabled, resulting in an unrecorded generation time (Table 8). On CUDA-enabled models, the transcription times vary significantly, with 8.24 seconds when flash attention is enabled and 7.96 seconds when it's disabled (Table 8). These differences highlight the importance of considering both device specifications and model configurations when evaluating performance.

Moreover, the "whisper-1" model, utilizing the OPENAI API, demonstrates relatively faster generation times compared to other models, achieving a time of 8.23 seconds as illustrated in Table 8. Conversely, the whisper open source "tiny" model, although not specified with device information, exhibits the fastest generation time among the listed models, completing the task in just 6.06 seconds (Table 8). These findings suggest that model architecture and implementation can significantly influence processing speed, with smaller models potentially offering faster inference times. And an 89.11% difference between the lowest and highest time it took to transcribe the audio.

Transcribed audio of 1 minute

The "openai/whisper-large-v2" model, when executed on the CUDA device without flash attention, exhibited a transcription time of approximately 66.81 seconds (Table 8). In contrast, as presented in Table 8, when flash attention

was enabled on the same model and device configuration, the transcription time reduced notably to 64.3 seconds. This reduction in time highlights the potential efficiency gains achieved through the optimization of model configurations, emphasizing the importance of parameter tuning in enhancing performance.

The "distil-whisper/distil-large-v2" model showcased varying performance outcomes based on the device and flash attention settings. When executed on a CPU without flash attention, the transcription time was recorded at 61.82 seconds, slightly faster than its counterpart with flash attention enabled, which resulted in an error and thus no transcription time was provided (Table 8). This discrepancy underscores the influence of hardware acceleration and model architecture on processing efficiency. Moreover, when the model was run on a CUDA device without flash attention, a considerable decrease in transcription time was observed, with timings of 8.1 seconds and 8.67 seconds for configurations without and with flash attention, respectively detailed in Table 8. This time declining signifies the significant impact of hardware utilization in expediting computational tasks.

In contrast, the "whisper-1" model, presumably executed via the OpenAI API, demonstrated remarkable efficiency, achieving a transcription time of merely 5.99 seconds, shown in table 8. This rapid processing speed underscores the efficacy of cloud-based inference platforms in facilitating swift execution of natural language processing tasks. Similarly, the whisper open source "tiny" model as illustrated in Table 8, while not specifying the device used for execution, also displayed commendable performance, with a transcription time of 6.19 seconds. These results highlight the trade-offs between model complexity and processing speed, showcasing the potential benefits of utilizing simpler architectures for expedited inference with a 91.03% difference between the highest differences.

Table 8. Whisper models results in webserver

model_name	device	flash_attention	transcribed_audio	time /s
openai/whisper-large-v2	cuda:0	true	Hello.	58.15
		false	Hello.	63.3
distil-whisper/distil-large-v2	cpu	true	Error	-
		false	Hello.	69.8
	cuda:0	false	Hello.	6.17
		true	Hello.	6.13
whisper-1	OPENAI API KEY	-	Hello.	1.14
tiny	-	-	Hello?	5.38
openai/whisper-large-v2	cuda:0	true	You will hear a number of different recordings and you will have to answer questions on wh	48.83
		false	You will hear a number of different recordings and you will have to answer questions on wh	54.1
distil-whisper/distil-large-v2	cpu	false	You will hear a number of different recordings and you will have to answer questions on wh	55.69
		true	Error	-
	cuda:0	true	You will hear a number of different recordings and you will have to answer questions on wh	8.24
		false	You will hear a number of different recordings and you will have to answer questions on wh	7.96
whisper-1	OPENAI API KEY	-	You will hear a number of different recordings and you will have to answer questions on wha	8.23
tiny	-	-	You will hear a number of different recordings and you will have to answer questions on wh	6.06
openai/whisper-large-v2	cuda:0	false	So in college, I was a government major, which means I had to write a lot of papers. Now, w	66.81
		true	So in college, I was a government major, which means I had to write a lot of papers. Now, w	64.3
distil-whisper/distil-large-v2	cpu	false	So in college, I was a government major, which means I had to write a lot of papers. Now, w	61.82
		true	Error	-
	cuda:0	false	So in college, I was a government major, which means I had to write a lot of papers. Now, w	8.1
		true	So in college, I was a government major, which means I had to write a lot of papers. Now, w	8.67
whisper-1	OPENAI API KEY	-	So, in college, I was a government major, which means I had to write a lot of papers. Now, wh	5.99
tiny	-	-	So in college, I was a government major, which means I had to write a lot of papers. Now, w	6.19

Choosing the appropriate ASR

Distil Whisper from HuggingFace stands out as a compelling choice for ASR applications based on several key factors. The model consistently demonstrates high accuracy in transcribing short and long audio segments, as evidenced by error-free outputs in the "Flash Attention" column. Its competitive speed, particularly when executed on GPU (cuda:0), further enhances its appeal for real-time applications. The flexibility of Distil Whisper is noteworthy, supporting both CPU and GPU (cuda:0) and allowing users to tailor their hardware choices based on specific computational requirements. The model exhibits commendable consistency across various lengths of transcribed audio, reflecting its reliability for diverse use cases. As an open-source offering from HuggingFace, Distil Whisper benefits from community support, ensuring ongoing improvements, updates, and bug fixes. Additionally, the model's adept handling of different audio inputs without encountering errors underscores its robustness.

Distil Whisper from HuggingFace presents a compelling choice over the Whisper API Key, despite the latter's faster speed, primarily due to cost

considerations. While the Whisper API Key may offer swifter processing times, the financial implications of utilizing an API key can be significant. Distil Whisper provides a balance between commendable speed, accuracy, and the advantage of being an open-source solution, all without incurring additional costs associated with API usage. The flexibility of Distil Whisper to operate on both CPU and GPU allows users to optimize their computational resources without being constrained by the expenses linked to API calls. In scenarios where cost-effectiveness is a priority, choosing Distil Whisper ensures a robust and reliable text-to-speech solution without compromising performance. With the cost per minute at $0.006, rounded to the nearest second, Distil Whisper from HuggingFace emerges as a well-rounded and dependable solution for ASR needs, combining accuracy, speed, flexibility, consistency, and the advantages of being an open-source model.

5 DISCUSSION

To complete the thesis, several sequential steps must be undertaken to demonstrate the seamless integration of advanced language models and speech recognition technologies into the Pepper robot. Initially, the process entails capturing audio inputs from the Pepper robot by activating its microphone module and ensuring proper connectivity with the required hardware and software components. Subsequently, the transcribed audio is processed using the selected ASR model, Distil Whisper, to convert it into text format for further analysis. Once transcribed, the text inputs are submitted to the chosen LLM, gpt4all-falcon-q4_0.gguf, for inference, where the model generates responses based on the input received. These responses are then captured and retrieved from the language model, ready for playback or display on the Pepper robot, enabling it to articulate the generated text and engage in interactive conversations with users.

Additionally, the thesis involves implementing functionality within the Pepper robot to prompt users with written text inputs or questions. These written prompts are then submitted to the language model for inference, where the model generates responses based on the provided prompts. The generated responses are subsequently captured and retrieved, ready for display or playback on the Pepper robot, enabling it to respond to written inputs from users in a conversational manner. Through these sequential steps, the thesis aims to showcase the effective integration of advanced language models and speech recognition technologies into the Pepper robot, enhancing its conversational abilities and facilitating more natural and interactive interactions with users.

In choosing the optimal language model for my thesis, I weighed response time, quality, and cost. Falcon-7b was excluded due to its lengthy response times (12.17 seconds for audio, approximately 10 seconds for input text) and confusing text. I found that gpt4all-falcon-q4_0.gguf offered a cost-effective solution with quick response times (2.5 seconds for audio, about 12 seconds

for input text), and its responses were of good quality although sometimes lacked detail. In contrast, gpt-3.5-turbo provided high-quality responses at a higher cost (0.0030$/1K tokens for input, 0.0060$/1K tokens for output). Ultimately, I decided to go with gpt4all-falcon-q4.gguf from GPT4ALL due to its open-source nature, zero-cost advantage, efficient response time, and decent response quality.

The comparative analysis of various models' performance in transcribing audio inputs reveals significant variations in processing times. For instance, processing times range from exceptionally low durations, such as 1.14 seconds for the 'whisper-1' model, such as 69.8 seconds for the 'distil-whisper/distil-large-v2' model with cpu and flash attention set to false. However, 6,13 second when model set to CUDA:0 with flask attention set to true, when transcribing an audio consisting of one word “hello”.

Device choice and the utilization of flash attention parameter significantly impact processing efficiency. For instance, when transcribing audio of approximately one minute and 10 seconds taken from a sample test of an IELTS listening exam, the 'openai/whisper-large-v2' model exhibits varying processing times depending on flash attention activation, ranging from 48.83 seconds to 54.1 seconds. Similarly, the 'distil-whisper/distil-large-v2' model demonstrates differences in processing times when run on different devices, with durations ranging from 55.69 seconds to 7.96 seconds. The inclusion of flash attention can notably impact processing efficiency, as evidenced by varying generation times.

Different model architectures and implementations influence processing speed. The "whisper-1" model, utilizing the OPENAI API, and the open source whisper "tiny" model exhibit relatively faster generation times compared to other models, with time ranging from 5.99 seconds to 6.19 seconds of a one minute audio. “distil-whisper/distil-large-v2” model took about 8 seconds transcribing the audio of a random conversation of a one minute audio, demonstrating its close duration to “whisper-1” model. Model complexity and

implementation significantly influence processing speed, with smaller models potentially offering faster inference times.

Choosing the appropriate ASR is crucial for efficient transcription. Distil Whisper from HuggingFace stands out as a compelling choice due to its high accuracy, competitive speed, flexibility in hardware utilization, reliability across various audio lengths, and open-source nature. Despite the faster speed of the Whisper API Key, Distil Whisper offers a balance between speed, accuracy, and cost-effectiveness, making it a robust and reliable solution for ASR needs.

The implementation phase of the project encountered several challenges, including issues with dynamic IP addresses and network connectivity, disparities in Python versions, limitations in speech recognition capabilities, audio implementation hurdles, and difficulties with file management. Despite these challenges, solutions were identified and implemented, contributing valuable insights into the complexities of integrating Pepper with modern technologies. These experiences underscore the importance of adaptability, problem-solving, and collaboration in overcoming obstacles encountered during the development process. Overall, the successful resolution of these challenges highlights the feasibility and potential of integrating Pepper into various technological frameworks, paving the way for future advancements in human-robot interaction.

6 CONCLUSION

The successful integration of advanced language models into the Pepper robot, coupled with the implementation of Whisper Automatic Speech Recognition (ASR) and the evaluation of multiple Language Models (LLMs), is a noteworthy achievement in advancing the conversational capabilities of robotics. Employing a systematic approach, this research effectively addressed the challenges of integrating cutting-edge language technologies with robotic systems.

The evaluation of LLMs, emphasizing response time, performance quality, and specific features, has contributed valuable insights to the field and resulted in the identification of gpt4all-falcon-q4_0.gguf as a versatile and cost-effective solution for seamless integration with the Pepper robot. This optimal balance between response speed, feature-rich performance, and cost-effectiveness positions gpt4all-falcon-q4_0.gguf as a main component, paving the way for enhanced HRIs.

Moreover, the successful integration of Whisper ASR into the Pepper robot has introduced a transformative dimension to human-machine interaction. The inclusion of ASR served as a powerful tool, simplifying HRI, and facilitating more natural and responsive exchanges. With the evaluations of the 4 whisper models, finding the optimal option to be distil whisper has contributed to a great value of insight. Optimizing model configurations, considering device specifications, and selecting the appropriate ASR are essential for enhancing transcription efficiency in audio processing tasks, with Distil Whisper emerging as a well-rounded and dependable solution for ASR needs.

In addressing the research questions posed at the outset of this study:

1. How well can the Pepper robot transcribe spoken audio?

   Through the utilization of Distil Whisper ASR model, the Pepper robot demonstrates commendable transcription capabilities. The evaluation of

various Whisper models highlighted the effectiveness of Distil Whisper, offering high accuracy and competitive speed in transcribing spoken audio.

2. Are the responses from the Pepper robot human-like enough?

The responses generated by the Pepper robot, utilizing gpt4all-falcon-q4_0.gguf, language models, are deemed sufficiently human-like for users to understand and grasp the intended meaning. While certain nuances may still be refined, the responses contribute to a meaningful interaction experience.

3. Is the latency generating the responses from the Pepper robot good enough for natural feeling conversation?

The latency in generating responses from the Pepper robot, facilitated by the integration of Language Models and ASR, is considered acceptable for achieving a natural feeling conversation. While improvements in response time and model efficiency are desirable, the implemented system demonstrates feasibility for engaging interactions.

Reflecting on the accomplishments of this thesis, it becomes evident that the fusion of language models and advanced speech recognition not only enhances the robot's ability to comprehend and respond to human interactions but also paves the way for more natural, intuitive, and meaningful exchanges. The promising results of this study suggest a bright future for integrating cutting-edge technologies in robotics, opening new avenues for collaboration and cooperation between humans and robots in various domains.

In conclusion, the successful implementation of advanced language models into the Pepper Robot and enhanced speech recognition capabilities marks a significant stride forward in the realm of HRI. The positive outcomes of this integration underscore the potential for this technology to be a powerful tool in facilitating seamless communication between humans and robots. The demonstrated effectiveness of the implemented system lays a strong foundation for further refinement and optimization, suggesting that with a few strategic

modifications, the Pepper Robot, which is equipped with advanced language models, can evolve into a highly valuable and versatile asset for HRI applications.

6.1 Future Work

Future work for this thesis could focus on several key areas to further enhance the capabilities of the Pepper robot in human-robot interaction (HRI). One avenue for exploration involves refining the integration of language models and speech recognition systems, aiming to improve accuracy, response quality, and the naturalness of interactions. This could entail experimenting with different pre-trained models, fine-tuning them for specific applications, and exploring novel approaches for integrating advanced AI technologies into robotic platforms. Another area of interest is reducing latency and enabling real-time interactions between users and the Pepper robot. This could involve optimizing algorithms, leveraging parallel processing techniques, and exploring hardware acceleration options to enhance processing speed and responsiveness during interactions. Overall, future research endeavors could further enhance the Pepper robot's capabilities and broaden its applicability in various real-world contexts, ultimately advancing the field of human-robot interaction.

REFERENCES

Akash Takyar. (2023, July 21). Natural Language Processing: A comprehensive overview. LeewayHertz - AI Development Company. https://www.leewayhertz.com/natural-language-processing/

Anand, Y., Nussbaum, Z., Treat, A., Miller, A., Guo, R., Schmidt, B., Community, Gpt., Duderstadt, B., & Mulyar, A. (2023). GPT4All: An Ecosystem of Open Source Compressed Language Models. ArXiv.org. https://arxiv.org/abs/2311.04931

Armin Norouzi, (2023). *The brief history of Large Language Models: A Journey from ELIZA to GPT-4 and Google Bard*. Medium; Level Up Coding. https://levelup.gitconnected.com/the-brief-history-of-large-language-models-a-journey-from-eliza-to-gpt-4-and-google-bard-167c614af5af

Billing, E., Rosén, J., & Lamb, M. (2023). *Language Models for Human-Robot Interaction*. https://www.diva-portal.org/smash/get/diva2:1744270/FULLTEXT01.pdf

Callie, Y., Kim, C., Lee, B., & Mutlu. (2024). Understanding Large-Language Model (LLM)-powered Human-Robot Interaction. https://doi.org/10.1145/3610977.3634966

Canonico, M., & De Russis, L. (n.d.). *A Comparison and Critique of Natural Language Understanding Tools*. Retrieved August 29, 2023, from https://core.ac.uk/download/pdf/226262881.pdf

Dale, R., Moisl, H., & Somers, H. (Eds.). (2000). Handbook of Natural Language Processing (1st ed.). CRC Press. https://doi.org/10.1201/9780824746346

Dass, R. (2018). *The essential guide to how NLP works - Riti Dass - Medium*. Medium; Medium. https://medium.com/@ritidass29/the-essential-guide-to-how-nlp-works-4d3bb23faf76

Devansh. (2023). *What are the Different Types of Transformers in AI - MLearning.ai - Medium*. Medium; MLearning.ai. https://medium.com/mlearning-ai/what-are-the-different-types-of-transformers-in-ai-5085275664e8

Dhaduk Hiren . (2023, September 1). The Complete Guide to Google Bard: Features, Benefits, and Use Cases. Simform - Product Engineering Company. https://www.simform.com/blog/the-complete-guide-to-google-bard/

Fred, O. (2022). What is Natural Language Processing (NLP)? | Nicholas Idoko. Nicholas Idoko. https://nicholasidoko.com/blog/2022/11/11/what-is-natural-language-processing-nlp/

FutureBeeAI. (2023, March 20). What is a Language Model: Introduction, Use Cases. FutureBeeAI; FutureBeeAI. https://www.futurebeeai.com/blog/what-is-a-language-model

Gillis, A. S. (2023). *natural language understanding (NLU)*. Enterprise AI; TechTarget. https://www.techtarget.com/searchenterpriseai/definition/natural-language-understanding-NLU

Guide to Natural Language Understanding (NLU) in 2023. (2023). AIMultiple. https://research.aimultiple.com/nlu/

Gupta, S. (2023, August 25). How Does NLP Change Tech/Computer Interactions? Parangat.com. https://parangat.com/blog/nlp-change-tech-interactions/

Hadi, M. U., Al-Tashi, Q., Qureshi, R., Shah, A., Muneer, A., Irfan, M., Zafar, A., Shaikh, M., Akhtar, N., Wu, J., & Mirjalili, S. (2023). *Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects*. https://doi.org/10.36227/techrxiv.23589741.v1

History and present of Natural Language Processing "Deep Talk." (2020). Deep-Talk.ai. https://www.deep-talk.ai/post/history-and-present-of-natural-language-processing#:~:text=The%20beginnings%20(1950%E2%80%931970),and%20understanding%20of%20natural%20language.

Hutchins, J. (2004). *The first public demonstration of machine translation: the Georgetown-IBM system, 7th January 1954*. ResearchGate. https://www.researchgate.net/publication/253470063_The_first_public_demonstration_of_machine_translation_the_Georgetown-IBM_system_7th_January_1954

Juang, B., & Rabiner, L. (2004). *Automatic Speech Recognition -A Brief History of the Technology Development*. https://web.ece.ucsb.edu/Faculty/Rabiner/ece259/Reprints/354_LALI-ASRHistory-final-10-8.pdf

Katikapalli Subramanyam Kalyan. (2024). A survey of GPT-3 family large language models including ChatGPT and GPT-4. Natural Language Processing. https://doi.org/10.1016/j.nlp.2023.100048

Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C., & Carlini, N. (2021). *Deduplicating Training Data Makes Language Models Better*. ArXiv.org. https://arxiv.org/abs/2107.06499

Lenci, A. (2023). Understanding Natural Language Understanding Systems. A Critical Analysis. https://arxiv.org/ftp/arxiv/papers/2303/2303.04229.pdf

Liddy, E. (2001). *Natural Language Processing Natural Language Processing Natural Language Processing 1*. https://surface.syr.edu/cgi/viewcontent.cgi?article=1043&context=istpub

Lin, T., Wang, Y., Liu, X., & Qiu, X. (2021). *A Survey of Transformers*. ArXiv.org. https://arxiv.org/abs/2106.04554

Lipenkova, J. (2022). *Choosing the right language model for your NLP use case*. Medium; Towards Data Science. https://towardsdatascience.com/choosing-the-right-language-model-for-your-nlp-use-case-1288ef3c4929

Liu, Z., Roberts, R., Madhu Lal-Nag, Chen, X., Huang, R., & Tong, W. (2021). AI-based language models powering drug discovery and development. *Drug Discovery Today*, *26*(11), 2593–2607. https://doi.org/10.1016/j.drudis.2021.06.009

Lutkevich, B., & Burns, E. (2023). *natural language processing (NLP)*. Enterprise AI; TechTarget. https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP

Mandour, A. (2023). *GPT-3.5 model architecture*. OpenGenus IQ: Computing Expertise & Legacy; OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/gpt-3-5-model/#:~:text=GPT%2D3.5%20was%20developed%20in,with%20multi%2Dhead%20attention%20blocks.

Mishra, D., Guillermo Arroyo Romero, Pande, A., Bhavana Nachenahalli Bhuthegowda, Dimitrios Chaskopoulos, & Shrestha, B. (2023). An Exploration of the Pepper Robot's Capabilities: Unveiling Its Potential. https://doi.org/10.3390/app14010110

Muhammad Owais. (2023, May 23). Classic NLP is dead — Next Generatio of Language Processing is Here. Medium; Artificial Intelligence in Plain English. https://ai.plainenglish.io/classic-nlp-is-dead-next-generation-of-language-processing-is-here-ce2d0dcf7725

Natural Language Processing (NLP): 7 Key Techniques. (2021). MonkeyLearn Blog. https://monkeylearn.com/blog/natural-language-processing-techniques/

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. (2022). *Training language models to follow instructions with human feedback*. https://arxiv.org/pdf/2203.02155.pdf

Pandey, A. K., & Gelin, R. (2018). A Mass-Produced Sociable Humanoid Robot: Pepper: The First Machine of Its Kind. *IEEE Robotics & Automation Magazine*, *25*(3), 40–48. https://doi.org/10.1109/mra.2018.2833157

Paramita (Guha) Ghosh. (2022). *The Fundamentals of Natural Language Processing and Natural Language Generation - DATAVERSITY*. DATAVERSITY. https://www.dataversity.net/fundamentals-natural-language-processing-natural-language-generation/

Park, S., & Lee, J. (2021). *Finetuning Pretrained Transformers into Variational Autoencoders*. ArXiv.org. https://arxiv.org/abs/2108.02446

Samant, R. M., Mrinal Bachute, Shilpa Gite, & Kotecha, K. (2022). Framework for Deep Learning-Based Language Models Using Multi-Task Learning in Natural Language Understanding: A Systematic Literature Review and Future Directions. *IEEE Access*, *10*, 17078–17097. https://doi.org/10.1109/access.2022.3149798

Samyuktha jadagi. (2023, July 29). AI and Natural Language Generation (NLG): Transforming Data into Human-Friendly Content. Medium; Medium. https://medium.com/@samyukthajadagi0/ai-and-natural-language-generation-nlg-transforming-data-into-human-friendly-content-34a9e873be31

Simplilearn. (2022). *What is Natural Language Understanding How Does it Work?* Simplilearn.com; Simplilearn. https://www.simplilearn.com/natural-language-understanding-article#:~:text=Natural%20Language%20Understanding%20(NLU)%20is,assistants%2C%20and%20automated%20translation%20services.

Standford Online. (2023). Stanford XCS224U: NLU I Intro & Evolution of Natural Language Understanding, Pt. 1 I Spring 2023 [Video]. YouTube. https://www.youtube.com/watch?v=K_Dh0Sxujuc&list=PLoROMvodv4rOwvldxftJTmoR3kRcWkJBp

Tech Gumptions. (2023, October 14). *Natural Language Processing(NLP) Pipeline - Tech Gumptions - Medium*. Medium; Medium. https://medium.com/@tech-gumptions/natural-language-processing-nlp-pipeline-e766d832a1e5

Tomáš Zemčík. (2019, October 22). A Brief History of Chatbots. ResearchGate; DEStech Publications. https://www.researchgate.net/publication/336734161_A_Brief_History_of_Chatbots

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł., & Polosukhin, I. (2017). *Attention Is All You Need*. https://arxiv.org/pdf/1706.03762.pdf

Wagh, A. (2023). *What's new in GPT-4: Architecture and Capabilities | Medium*. Medium; Medium. https://medium.com/@amol-wagh/whats-new-in-gpt-4-an-overview-of-the-gpt-4-architecture-and-capabilities-of-next-generation-ai-900c445d5ffe

Wang, C., Wu, A., & Pino, J. (2020). *CoVoST 2 and Massively Multilingual Speech-to-Text Translation*. https://arxiv.org/pdf/2007.10310.pdf

Weizenbaum, J. (1966). ELIZA---a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, *9*(1), 36–45. https://doi.org/10.1145/365153.365168\

What is a large language model (LLM)? (2022). Cloudflare. https://www.cloudflare.com/learning/ai/what-is-large-language-model/

What is Natural Language Processing? | IBM. (n.d.). Ibm.com. Retrieved October 1, 2023, from https://www.ibm.com/topics/natural-language-processing

What is NLP? - Natural Language Processing Explained - AWS. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/what-is/nlp/

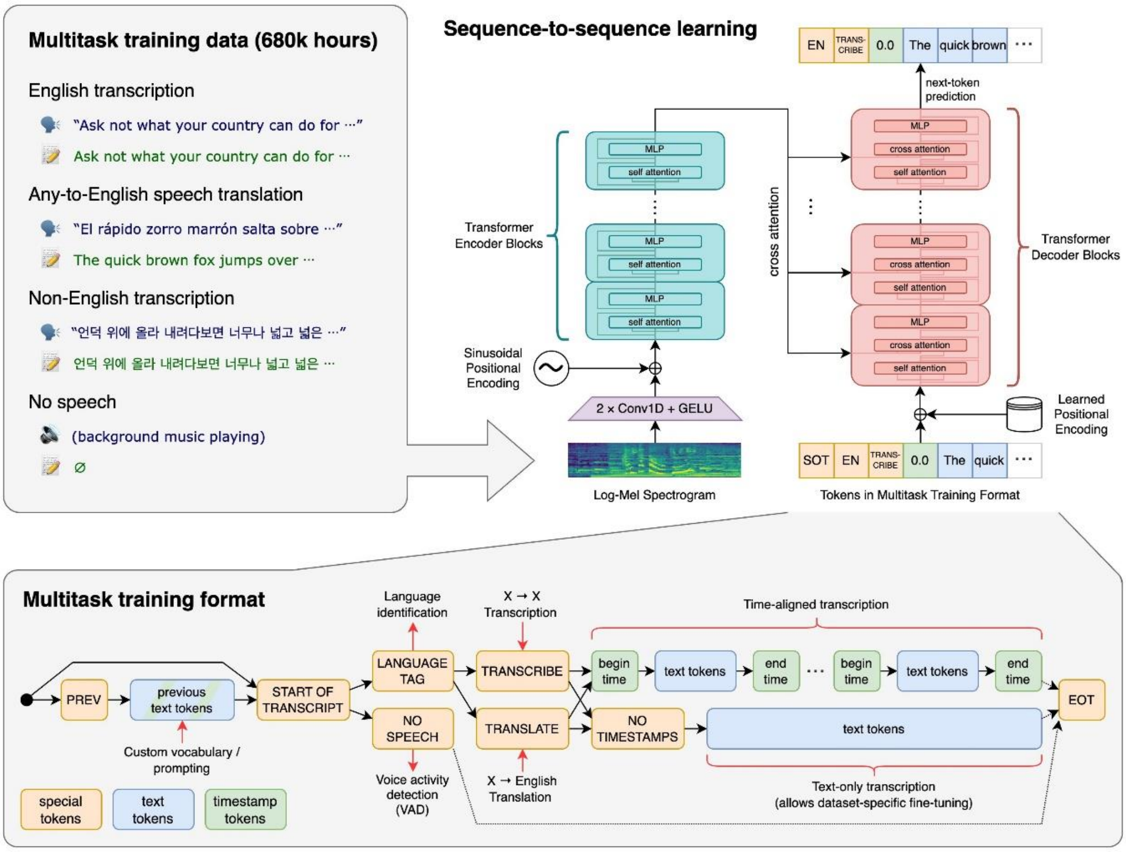
Ye, J., Chen, X., Xu, N., Zu, C., Shao, Z., Liu, S., Cui, Y., Zhou, Z., Gong, C., Shen, Y., Zhou, J., Chen, S., Gui, T., Zhang, Q., & Huang, X. (2023). *A Comprehensive Capability Analysis of GPT-3 and GPT-3.5 Series Models*. https://arxiv.org/ftp/arxiv/papers/2303/2303.10420.pdf

Abideen Zain ul. (2023). Autoregressive Models for Natural Language Processing. Medium; Medium. https://medium.com/@zaiinn440/autoregressive-models-for-natural-language-processing-b95e5f933e1f

Zhang, C., Chen, J., Li, J., Peng, Y., & Mao, Z. (2023). Large language models for human-robot interaction: A review. Biomimetic Intelligence and Robotics, 3(4), 100131–100131. https://doi.org/10.1016/j.birob.2023.100131

Zhao, W., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., & Liu, P. (n.d.). *A Survey of Large Language Models*. https://arxiv.org/pdf/2303.18223.pdf

APPENDIX 1


Multitask training data (680k hours)
English transcription
"Ask not what your country can do for ⋯"
Ask not what your country can do for ⋯
Any-to-English speech translation
"El rápido zorro marrón salta sobre ⋯"
The quick brown fox jumps over ⋯
Non-English transcription
"언덕 위에 올라 내려다보면 너무나 넓고 넓은 ⋯"
언덕 위에 올라 내려다보면 너무나 넓고 넓은 ⋯
No speech
(background music playing)
∅
Sequence-to-sequence learning
EN TRANS-CRIBE 0.0 The quick brown ⋯
next-token prediction
MLP
cross attention
self attention
Transformer Encoder Blocks
Transformer Decoder Blocks
cross attention
Sinusoidal Positional Encoding
2 x Conv1D + GELU
Log-Mel Spectrogram
Learned Positional Encoding
SOT EN TRANS-CRIBE 0.0 The quick ⋯
Tokens in Multitask Training Format
Multitask training format
Language identification
X → X Transcription
Time-aligned transcription
PREV
previous text tokens
START OF TRANSCRIPT
LANGUAGE TAG
NO SPEECH
TRANSCRIBE
TRANSLATE
begin time
text tokens
end time
NO TIMESTAMPS
EOT
Custom vocabulary / prompting
Voice activity detection (VAD)
X → English Translation
Text-only transcription (allows dataset-specific fine-tuning)
special tokens
text tokens
timestamp tokens

APPENDIX 2

Illustration of the relationship between NLP, NLU, NLG, and LLM

Term	Task	Description	Relationship to Other Terms
Natural Language Pro-cessing (NLP)	Processing and under-standing hu-man lan-guage	NLP encompasses a wide range of techniques for manipulating and un-derstanding human lan-guage, including natural language understanding (NLU) and natural lan-guage generation (NLG).	NLU and NLG are both sub-fields of NLP.
Natural Language Under-standing (NLU)	Extracting meaning from human language	NLU systems analyze hu-man language to extract its meaning, identifying keywords, phrases, and relationships between concepts.	NLU takes in-put from hu-man language and provides output that can be understood by machines.
Natural Language Genera-tion (NLG)	Creating hu-man-like lan-guage	NLG systems generate human-like language, such as text, code, or speech, from structured data or other forms of in-put.	NLG takes in-put from ma-chines and pro-vides output that can be un-derstood by hu-mans.
Large Language Model (LLM)	A neural net-work trained on a mas-sive dataset of text and code	LLMs are trained on mas-sive amounts of text and code, enabling them to perform a wide range of tasks, including NLU, NLG, and question an-swering.	LLMs can be used to im-prove the per-formance of NLU and NLG systems.

APPENDIX 3

GITHUB LINK: https://github.com/Raneem-AbdeHafez/Enhancing_Pep-perRobot_LLM.git