PROJECT THESIS

NAO Meets GPT

Authors:
Meseret Milion*
Bosshard Fabian*

Supervisors:
Sadurski Marcin
Wyss Alexander

https://github.com/fabianbosshard/nao_meets_gpt

December 22, 2023

*Equal contribution.

Zurich University
of Applied Sciences

School of
Engineering

DECLARATION OF ORIGINALITY

Project Work at the School of Engineering

By submitting this project work, the undersigned student confirm that this work is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the project work have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:

Winterthur, December 2023

Signature:

F. Bosshard

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all project works submitted.

Zurich University
of Applied Sciences

Abstract

This project thesis explores the integration of advanced conversational AI, specifically ChatGPT, into the humanoid robot NAO, aiming to enhance human-robot interaction. The research is divided into two primary domains: theoretical exploration and software development. The theoretical study examines the origin and progression of humanoid robots and LLMs, providing a critical backdrop for practical application. This foundational understanding informs the subsequent software development phase, which focuses on bridging Python 2 and Python 3 environments. This bridge is crucial for enabling effective communication between NAO's SDK, which is limited to Python 2, and the Python 3 libraries necessary for ChatGPT, speech recognition, and other advanced functionalities.

The core challenge of integrating disparate technological frameworks was surmounted through a novel Flask-based web-API integration. This approach involved constructing a Python 3 client that dispatches HTTP requests to a Python 2 server, thereby seamlessly harnessing ChatGPT's capabilities and speech recognition libraries. A standout outcome of this integration is the elevation of NAO's communication abilities, transitioning from basic speech output to dynamic, user-engaged conversations.

One of the distinctive findings from this integration is NAO's significantly enhanced interactional fluidity, demonstrating marked improvements in both response accuracy and contextual awareness in conversations with users. This advancement not only signifies a leap in NAO's functionality but also opens new avenues for its application in diverse fields such as education, healthcare, and personal assistance. The insights gained from this project illuminate paths for future research, particularly in refining AI-driven communication in humanoid robotics, setting a benchmark for future developments in the field.

Acknowledgements

We would like to extend our sincere thanks to our supervisors, Marcin Sadurski and Alexander Wyss, for their guidance and insights throughout this project. Their expertise has been invaluable in shaping our research.

We also express our gratitude to our families for their support and understanding during this academic endeavor.

Special thanks to Institute of Mechatronic Systems and the entirety of its people. Their welcoming and supportive environment, alongside excellent working facilities, including a dedicated workspace, greatly enhanced our research experience. Their friendliness and the sense of community they fostered made a significant, positive impact on our work.

Contents

List of Abbreviations

ACE	Autonomous Cognitive Entity
AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
GPT	Generative Pre-trained Transformer
HMM	Hidden Markov Model
HRC	Human-Robot Collaboration
HRI	Human-Robot Interaction
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LLM	Large Language Model
NLP	Natural Language Processing
Pyro	Python Remote Objects
RLHF	Reinforcement Learning from Human Feedback
ROS	Robot Operating System
RPC	Remote Procedure Call
SDK	Software Development Kit
SFT	Supervised Fine-Tuning
URL	Uniform Resource Locator

1 Introduction

1.1 Motivation

NAO robots, developed by SoftBank Robotics, are widely recognised for their versatility and interactive capabilities. However, their current communication capabilities are limited, often restricted to pre-programmed responses and basic interaction patterns. This limitation hinders their potential in various interactive applications.

The integration of OpenAI's Generative Pre-trained Transformer (GPT) Application Programming Interface (API), a state-of-the-art Large Language Model (LLM), provides a groundbreaking opportunity to overcome these limitations. GPT's advanced algorithm enables a more natural and human-like flow of conversation, which could significantly improve the interactive experience with NAO robots.

This enhancement opens up a myriad of possibilities, from transforming educational methods by providing interactive, personalised learning experiences, to offering companionship and assistance to those in need. The ability of NAO robots to engage in meaningful conversations can revolutionise their use in various sectors.

This thesis aims to contribute to the fields of robotics and Artificial Intelligence (AI) by not only implementing advanced AI in a practical setting, but also by analysing the results of this integration. It explores how the combination of AI and robotics can create new possibilities and what limitations still exist.

The integration of GPT with NAO robots presents unique technical challenges, including ensuring real-time processing and response generation. This thesis will explore innovative solutions to these challenges and contribute to the advancement of robot communication technologies.

1.2 Topicality and Relevance

1.2.1 Current State of Robotics and AI

The recent advancements in Natural Language Processing (NLP) have led to the development of sophisticated LLMs such as Bidirectional Encoder Representations from Transformers (BERT), GPT-4, and Codex. These models have markedly revolutionized various applications [1]. The emergence of these LLMs opens a promising avenue for fostering an interactive, communicative, and robust framework in Human-Robot Collaboration (HRC) [2].

In the field of robotics, there has been a significant evolution from task-oriented machines to systems infused with enhanced intelligence and adaptability, a change largely attributed to AI integration. This evolution has been propelled by parallel developments in AI, especially in the areas of machine learning, neural networks, and NLP. As a result, the convergence of robotics

and AI technologies is leading to the creation of systems characterized by improved decision-making, learning capabilities, and the potential for human-like interactions [2].

1.2.2 Significance of AI Integration in Robotics

The integration of AI into robotics marks a crucial development in the field. AI confers upon robots the ability to process and learn from extensive data, facilitating real-time decision-making. This integration is pivotal due to:

- **Enhanced autonomy**: AI equips robots to function independently in unstructured environments, transcending their traditional operational confines [3].
- **Improved HRC**: AI-infused robots can better understand and respond to human behavior and emotions, broadening their utility in assistive roles across various sectors [2, 3].
- **Adaptability and learning**: AI enables robots to adapt to new tasks and environments, learning from interactions, which is essential across diverse applications, from manufacturing to space exploration [1].

1.2.3 Relevance in Today's Technological Context

The integration of GPT with the humanoid robot NAO, as investigated in this research, aligns with contemporary technological trends. This research significantly contributes to the field by:

- **Bridging human-machine communication gaps**: Enhancing NAO with GPT's conversational capabilities reduces the communication barrier between humans and robots, a crucial aspect in HRC.
- **Expanding applications in education and therapy**: The augmented interaction capabilities of NAO promise to revolutionize its utility in educational and therapeutic contexts, particularly in personalized interventions.
- **Pioneering future innovations**: This research exemplifies practical methodologies for robotic AI integration, illuminating potential challenges and paving the way for future innovations in the field.

In essence, the integration of advanced AI with humanoid robotics, exemplified in this study, reflects not only a response to current technological advancements but also acts as a harbinger for future breakthroughs, expanding the horizons of HRC.

1.3 Research Question

What are the methods and strategies involved in successfully integrating the GPT API with the humanoid robot NAO, and what technical challenges and practical considerations are encountered during this process? Furthermore, how does this integration enhance NAO's functionality, particularly in terms of communicative and interactive capabilities, and what potential does this advancement hold for similar applications in humanoid robotics?

1.4 Implementation

The complete source code developed for this project is openly accessible on GitHub at the following link: `https://github.com/fabianbosshard/nao_meets_gpt`

2 Foundations

2.1 Humanoid Robots

2.1.1 Evolution of Humanoid Robotics

Since our thesis is concerned with Human-Robot Interaction (HRI), we put the emphasize on robots that are primarily designed for the same purpose.

Robots by Waseda University in Japan

The journey of humanoid robotics, as we understand it today, began in the early 1970s with significant strides in creating robots resembling human form and behavior. In 1973, WABOT-1 (left photograph in Figure 1) was built, recognized as the first full-scale humanoid robot. This robot boasted capabilities for moving limbs, seeing, and conversing in Japanese. It is equipped with external receptors such as artificial ears, eyes and a mouth, can accurately measure the distance and direction of objects it seeks. Its bipedal locomotion is enabled by its lower limbs, allowing it to move with a human-like gait. Furthermore, the robot is adept at manipulating and transporting objects, a skill made possible by tactile sensors installed in its hands, which enhance its grip and handling capabilities. This innovation paved the way for future advancement in the field [4, 5].

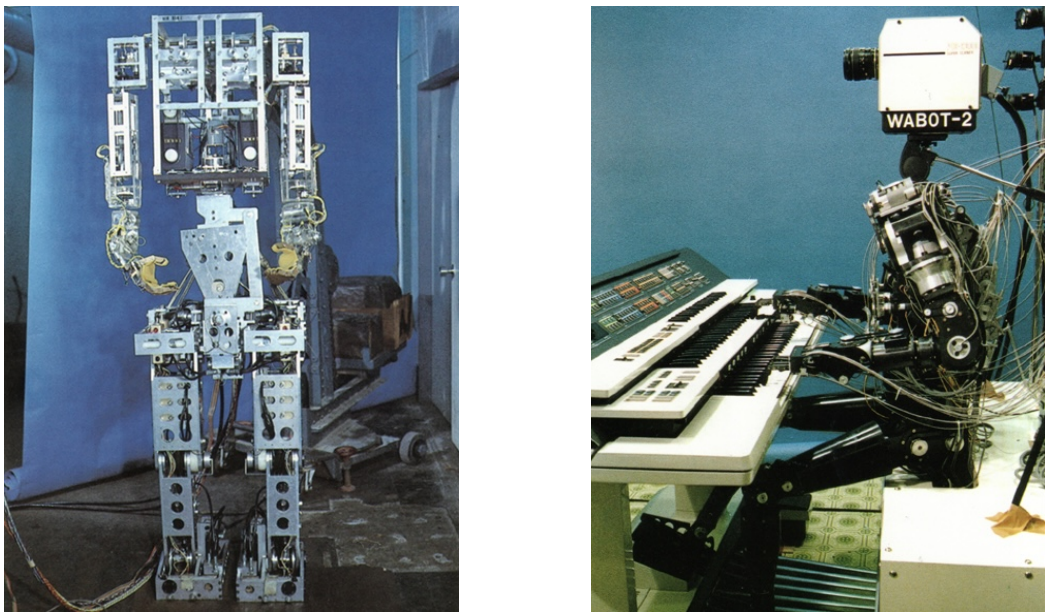


Figure 1: (left) WABOT-1, the first full-scale humanoid robot. (right) WABOT-2 demonstrating its capability to play the keyboard [4].

In 1984, an improved version followed with WABOT-2 (right photograph in Figure 1), a robot capable of performing complex tasks such as playing musical instruments – an endeavor requiring both intelligence as well as dexterity. Developed with a focus on natural and efficient information extraction from HRI, this system demonstrates how complex conversational dynamics can be effectively managed. It is composed of the following five integrated subsystems: limb control system, vision system, conversation system, singing voice tracking system, supervisory system. One key feature of WABOT-2 is its conversation control mechanism, which uses a state transition network (Figure 2). This network allows the robot to adapt its internal state according to speech input, ensuring relevant and accurate responses during conversations [5].

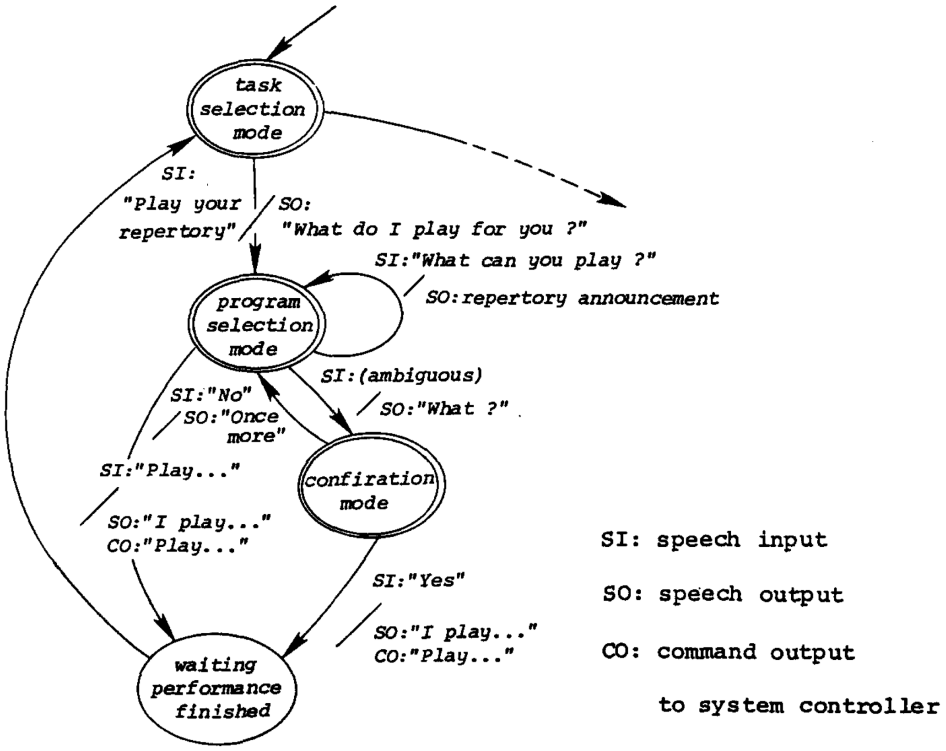


Figure 2: State transition network for conversation control of WABOT-2 [5].

In 1997, the humanoid robot Hadaly-2 was developed with the aim of facilitating smooth and natural communication between humans and robots, utilizing advanced AI techniques. It incorporated one-pass, time-synchronous algorithms, and relied on Hidden Markov Models (HMMs) for acoustic modeling of continuous speech. These algorithms allowed Hadaly-2 to recognize and respond to full sentences, rather than isolated words, enhancing interaction with users. Furthermore, its dialogue control module, integrating both speech and image recognition, enabled Hadaly-2 to interpret human instructions effectively (Figure 3). This module managed not just speech output, but also the robot's physical movements and responses, adapting dynamically to the context of the interaction. As with WABOT-2 (Figure 2), dialogue management is executed through state transitions, responding to auditory and visual cues based on the prevailing context. A notable advancement in Hadaly-2 was its ability to communicate through gestures and mimicry [4].

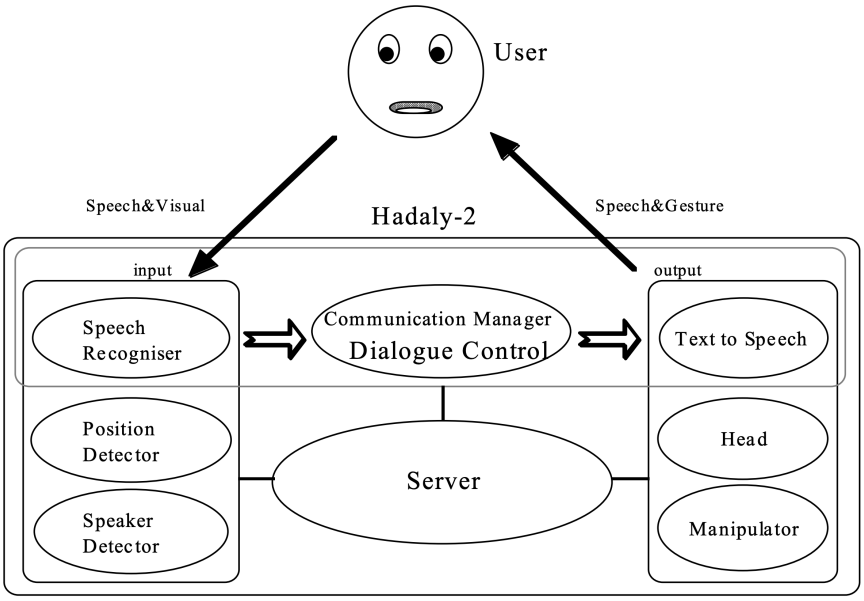


Figure 3: Module composition of Hadaly-2 [4].

ASIMO by Honda

ASIMO, unveiled by Honda in 2000, marked another milestone in humanoid robotics. Equipped with sophisticated sensors, ASIMO has the ability to recognize objects, gestures, and faces, enabling it to communicate interactively with humans as well as its environment [6].

NAO by Aldebaran Robotics

NAO was released in 2008 by Aldebaran Robotics. It is primarily used for research and education purposes. For more details, see 2.1.2.

Sophia by Hanson Robotics

Sophia, developed by Hanson Robotics in 2016 is known for its human-like appearance and advanced AI integration. Sophia features over 50 facial expressions, contributing to its lifelike interactions. Its software enables complex conversations, emotion recognition, and empathetic responses, making it highly effective in social interactions. Notably, Sophia was granted Saudi Arabian citizenship, a unique recognition for a robot [6].

Ameca by Engineered Arts

Ameca, created by Engineered Arts in 2021, stands as the most advanced humanoid robot in terms of realistic human expression to date. Its capability to accurately mimic human facial expressions heralds a new era in HRI. This advancement opens up possibilities for more empathetic and emotionally nuanced interactions with technology, setting a new standard in the realm of humanoid robotics and changing the way we perceive and engage with machines [6].

2.1.2 NAO

Overview

Figure 4: NAO [7].

NAO (Figure 4) is a humanoid robot, standing at a height of 58 cm and weighing 5.6 kg. Its design emphasizes a friendly and approachable appearance, with no visible metallic parts or sharp edges, featuring only smooth, rounded surfaces. The aim is to evoke an instant urge in users to hug the robot rather than fear it.

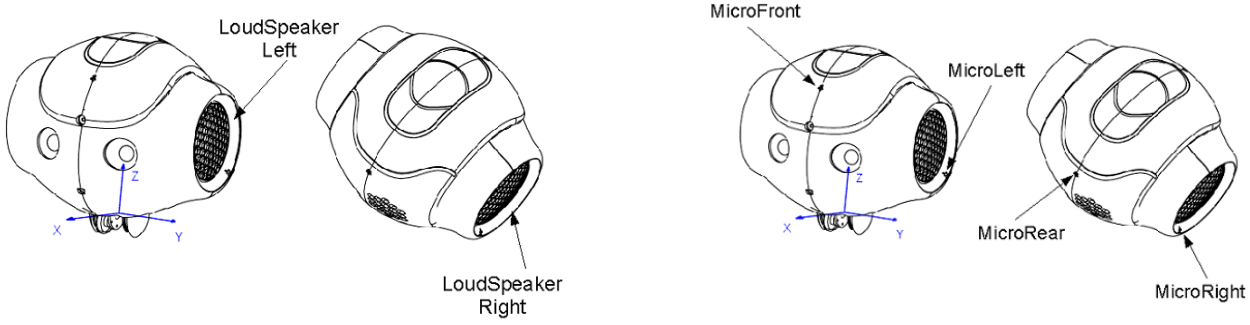
NAO possesses 25 degrees of freedom, including movements in the neck, arms, hands, and legs. The robot's pelvic joints are mechanically synchronized, preventing the trunk from rotating along the vertical axis when both legs support it.

NAO is programmed using the specialized NAOqi framework. The framework, known as NAOqiOS, is accessible in programming languages like Python and C++, as well as through Choregraphe, which offers a user-friendly graphical interface. It simplifies the development of applications, enabling users to program complex behaviors into the NAO robot without needing extensive programming expertise or deep knowledge of robotics [8, 9].

NAO can be connected to networks using either an Ethernet port located at the back of its head or wirelessly, which enables autonomous operation and remote control. This functionality is especially important when the robot is operating in real-world environments.

Speech Capabilities of NAO

The NAO humanoid robot is engineered with advanced speech capabilities, crucial for its role as a companion robot. It is equipped with an array of four directional microphones and two loudspeakers on its head (see Figure 5).



(a) Loudspeakers (b) Microphones

Figure 5: NAO audio hardware [10].

Speech synthesis in NAO strikes a meticulous balance between sounding distinctly robotic for clarity and sufficiently fluid to maintain comfort and approachability. The synthesis can be accompanied by synchronized gestures, reflecting the natural human tendency to complement verbal communication with non-verbal cues.

The `ALAudioDevice` module (Figure 6) is responsible for handling audio inputs and outputs. Therefore, any module designed to transmit sounds through NAO's speakers or to analyze sounds from NAO's microphones needs to interact with the `ALAudioDevice` module. For real-time audio processing, the initial step involves creating an "Audio in" module, which needs to subscribe to the `ALAudioDevice` module. The `ALAudioDevice` module will forward the input buffer through a callback function. It dispatches these buffers in a sequential manner; the input buffer is relayed to the first registered module, followed by the next one after the first completes its processing, and so on. The `ALAudioDevice` module consistently reads its input buffers at fixed time intervals

$$T = \frac{N}{f_s} \tag{1}$$

, which are determined by the buffer length N and the sampling rate f_s. To maintain real-time processing, the cumulative processing time of all registered modules must not surpass this interval T. Failing to do so will result in missing audio buffers, disrupting real-time processing [11].

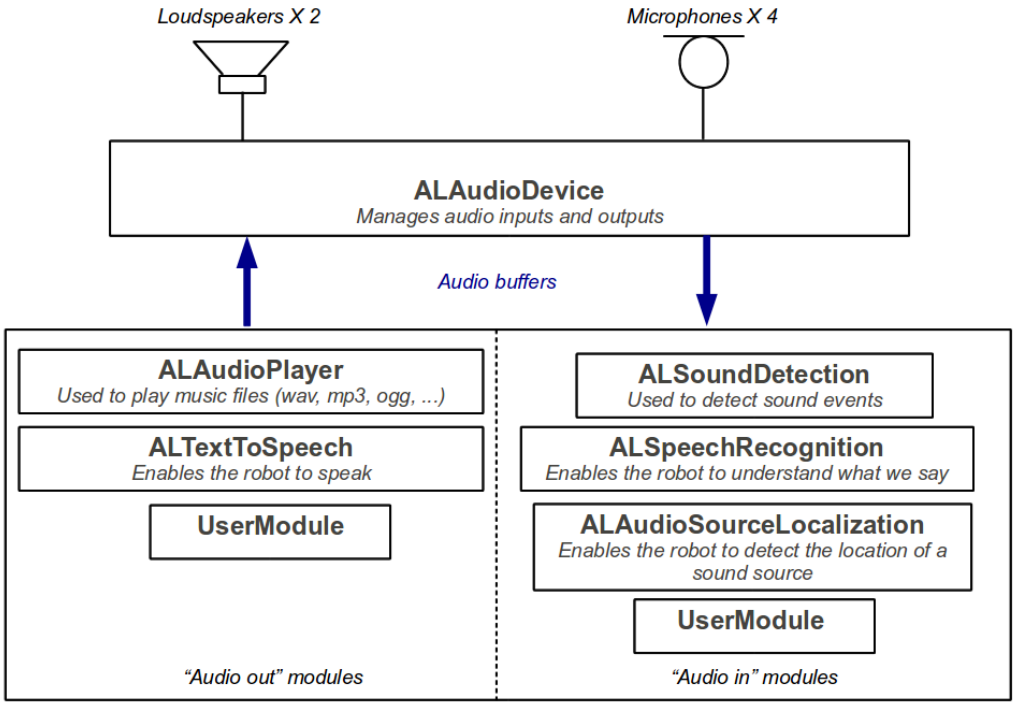


Figure 6: Architecture of the NAOqi audio modules [11].

2.2 Large Language Models

2.2.1 Architecture

LLMs are based on a kind of neural network known as a transformer. The original transformer architecture, illustrated in Figure 7, was initially proposed for tasks like machine translation in 2017. Unlike previous models, transformers exclusively utilize attention mechanisms and no recurrence or convolution [12]. These attention mechanisms enable the model to focus on specific parts of the input or output sequences, regardless of the distance between words or, to be more precise, tokens, which are just pieces of words. This feature allows the model to concentrate only on information that is relevant when generating subsequent tokens, even if they are spatially distant in a (possibly long) text [13].

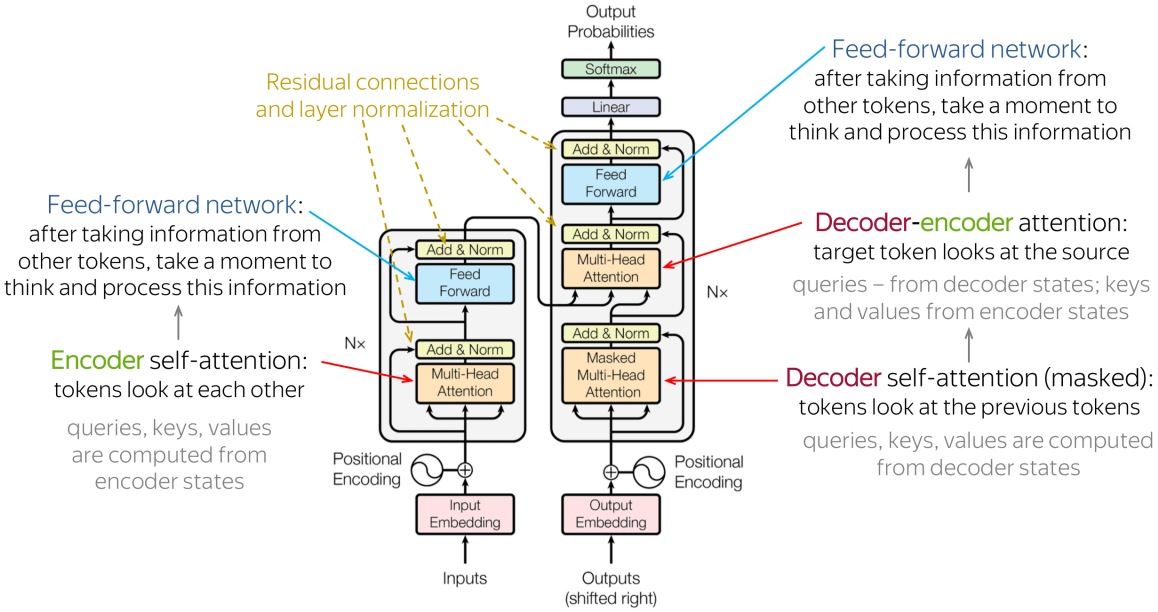


Figure 7: Original transformer architecture [14].

The full architecture of a modern transformer-based LLM integrates a myriad of algorithmic, numerical, and engineering techniques, which are extensive and beyond the scope of this project. However, we focus on elucidating the core building block, the attention mechanism.

The attention mechanism operates as follows: Given an input sequence of n tokens (i.e., a sentence), each token generates a trio of vectors: a query, a key, and a value. The query acts like a beacon, sending out a signal asking, "What information am I looking for?". The key, on the other hand, can be interpreted like a flag or label, declaring, "This is the information I represent!". The query and key vectors "collaborate" to compute attention weights, reflecting the relevance of each token's information. The value vectors carry the actual informational content of the tokens. For each token, a weighted sum of all value vectors is computed using these attention weights [15].

To determine these weights, we calculate the dot product of each query vector with all key vectors. A large dot product signifies a strong alignment between the query and a key, indicating the corresponding token's increased relevance to the query. These dot products are then normalized

and passed through a softmax function. While the exact reasons behind the use of this function, apart from its empirical performance, remain somewhat open to investigation, certain properties of the function are believed to play a key role. Specifically, it ensures all values in the attention matrix are non-negative, which helps the model to focus on positive correlations and ignore irrelevant negative features. Additionally, softmax introduces a non-linear reweighting mechanism that amplifies only the most relevant features [16].

In matrix notation, the attention mechanism can be expressed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{2}$$

Here, $\mathbf{Q} \in \mathbb{R}^{n \times d_q}$, $\mathbf{K} \in \mathbb{R}^{n \times d_k}$, and $\mathbf{V} \in \mathbb{R}^{n \times d_v}$ represent matrices of queries, keys, and values, respectively, corresponding to different tokens. The weights $\mathbf{W}_q \in \mathbb{R}^{d_q \times d}$, $\mathbf{W}_k \in \mathbb{R}^{d_k \times d}$, and $\mathbf{W}_v \in \mathbb{R}^{d_v \times d}$ that are used to calculate the query $\vec{q}_i \in \mathbb{R}^{d_q}$, key $\vec{k}_i \in \mathbb{R}^{d_k}$, and value $\vec{v}_i \in \mathbb{R}^{d_v}$ of an input token $\vec{x}_i \in \mathbb{R}^d$ are adjustable parameters that are learned during model training. The term $\mathbf{Q}\mathbf{K}^T$ is the matrix representation of all the dot products, denoting the degree of focus each token should have on every other token. Rows in the attention matrix represent the focus of a specific token on all others, while columns represent how much attention each token receives from the others. The attention mechanism, i.e. one attention-head, is visually depicted in Figure 8. These attention-heads are then stacked in parallel as well as in series, thus enabling a model to capture different aspects of language [12, 17].

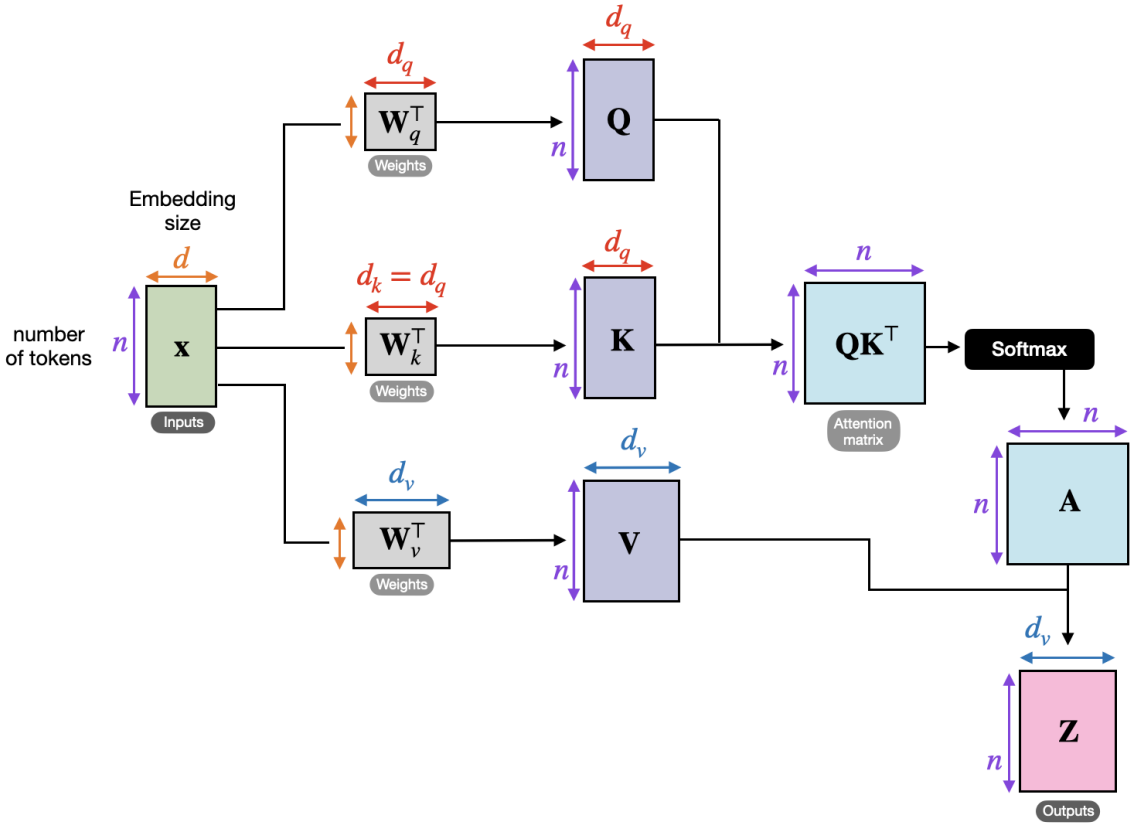


Figure 8: One attention head [17].

2.2.2 Training Stages Deployed for LLMs

The training of a state-of-the-art LLM is typically done in multiple stages, each with its unique purposes, challenges and requirements.

Unsupervised Pre-Training: Initially, LLMs undergo unsupervised generative pre-training. During this phase, the model is trained to predict the next token in a sequence of text. This foundational stage enables the model to learn the basic structure, syntax, and semantics of the language. However, at this stage, the model primarily functions as a text completion tool and lacks the refined abilities required for specific tasks. It might, for instance, generate various plausible continuations to a given prompt, sometimes even responding with a question, reflecting its yet unrefined understanding of user intent [18]. This initial limitation is humorously illustrated in Figure 9, which, while not a technical depiction, captures the essence of this early stage (and the subsequent stages) in a graphical matter.

Supervised Fine-Tuning (SFT): This stage serves as a pivotal transition for LLMs from functioning primarily as document completion tools to effectively answering questions and performing specific tasks. This process begins by assembling a dataset of high-quality LLM outputs, showcasing ideal model behavior. The LLM undergoes direct fine-tuning with this dataset, guided by the examples it should emulate, marking the 'supervised' aspect of the process. This stage enables the LLM to evolve beyond mere text completion, equipping it with the nuanced capability to generate accurate, context-aware responses, and perform targeted tasks like question answering, reflecting a significant advancement in its functional versatility [19].

Reinforcement Learning from Human Feedback (RLHF): In this final stage, the LLM undergoes refinement through human feedback. The focus is on aligning the model's outputs with human values and preferences, essential for applications requiring reliability, ethical sensitivity, and safety. A significant challenge here is to develop effective feedback mechanisms that ensure the model's responses are ethically aligned and intention-accurate without leading to overly cautious behavior for safe inputs. Achieving this balance is crucial for an LLM's practical and ethical effectiveness [20].

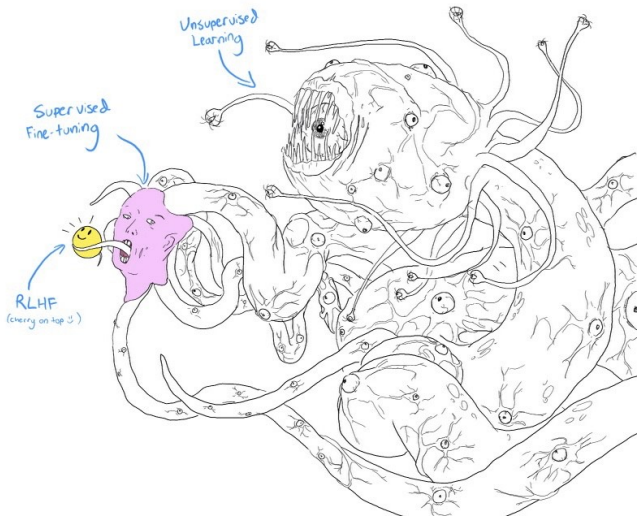


Figure 9: Training stages of LLMs [18].

2.2.3 Evolution of LLMs

One of the foundations for modern LLMs is the concept of word embeddings, wherein each token is represented as a vector $\vec{x}_i \in \mathbb{R}^d$ with embedding dimension d. While vector representations of words were proposed as early as 2003 [21], a notable breakthrough came with Word2Vec in 2013. This method significantly advanced NLP by efficiently generating vector representations that capture both semantic and syntactic aspects of words, as visualized in 10. Consequently, this allowed for concepts like word similarity or synonymy to be quantified in mathematical form [22]. This approach marked a major leap from previous, simpler word representation methods and continues to influence contemporary language models [23, p. 218–228].

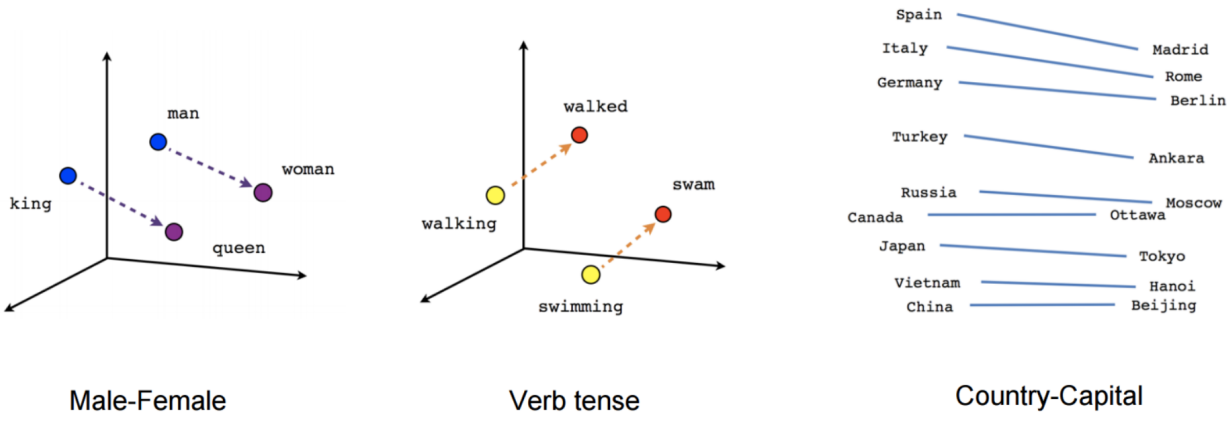


Figure 10: Semantic and syntactic relations captured by Word2Vec [24].

The transformer, as described in 2.2.1, marked another significant advancement in language modelling, because it was superior in performance to previous recurrent models, while at the same time requiring less time to train. The original architecture utilized both an encoder (left part in Figure 7) and a decoder (right part in Figure 7). This dual structure is useful for translating between languages, as it involves encoding the source language's meaning and reproducing it in the target language, while also considering the previously generated tokens in the target language [12, 15, 23].

However, for other applications like sentiment analysis, where the objective is to understand the meaning of a text without sequence generation, an encoder-only model such as BERT, developed by Google in 2018, can be highly effective. BERT is initially pre-trained on a large text corpus through self-supervision to learn general information about language patterns. Such encoder-only models utilize transfer learning: In the fine-tuning stage, BERT's parameters are adjusted, and an additional layer is added to tailor the model for the desired application, using a smaller, task-specific dataset [23, 25].

In the domain of text generation, the necessity of an encoder component is not universally applicable, as demonstrated by the architectural design of LLMs such as the GPT series. This series, encompassing GPT, GPT-2, GPT-3, and subsequent iterations, represents a pivotal development by OpenAI. They employ a decoder-only architecture. In a decoder, the attention mechanism is designed such that a token can only attend to preceding tokens, necessitating a masked attention matrix. As outlined in 2.2.2, these models are predominantly trained through

next-token-prediction on extensive text corpora, thereby learning to generate text that is both coherent and contextually relevant [26].

Particularly, GPT-3 stands out for its advanced language understanding and generation capabilities, operating effectively under paradigms of few-shot or zero-shot learning. This indicates its ability to perform diverse language tasks with minimal or no task-specific training data, leveraging its extensive pre-training. This approach marks a departure from traditional machine learning models that require extensive task-specific fine-tuning. The success of GPT-3 in this aspect underscores the potential of large-scale, decoder-based language models in NLP [27].

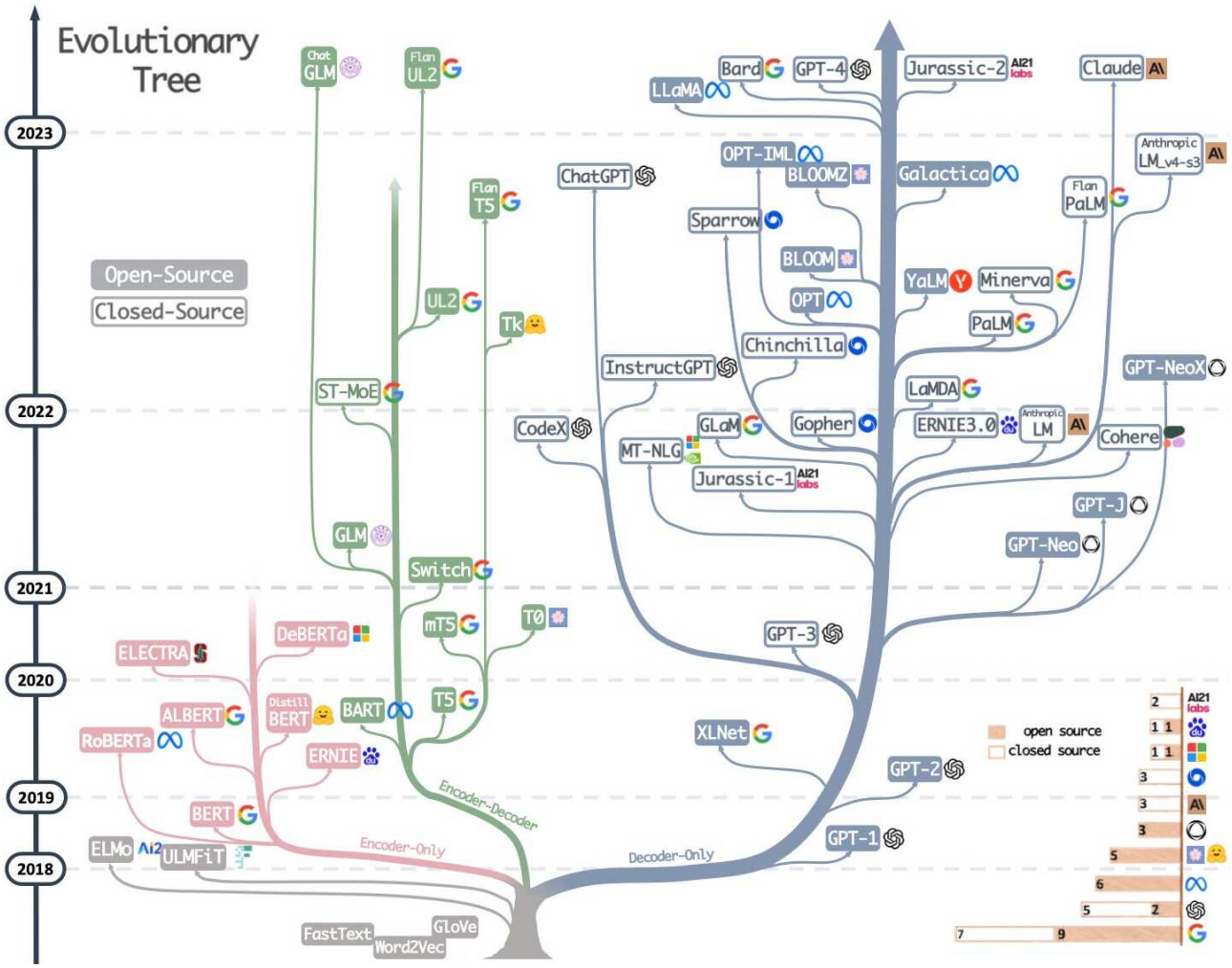


Figure 11: Evolution of LLMs [28].

The recent history of LLMs has been marked by significant contributions from major tech companies and rapid advancements in technology, as depicted in Figure 11.

2.3 Related Work

2.3.1 Advancing HRI with AI

Integrating advanced AI technologies in HRC has shown transfomative potential. One study [2], which is demonstrates this by controlling a 7-degree-of-freedom robot arm with the integration of ChatGPT. This approach highlights AI's role in enhancing interactive capabilities within robotics. The study investigates the application of ChatGPT as an intelligent assistant, capable of understanding task-specific contexts and controlling a robot arm through Robot Operating System (ROS) messages. As illustrated in Figure 12 it introduces an AI-driven workflow, RoboGPT, which translates human operators' spoken language into textual input for AI processing. Utilizing GPT-3.5 for decision-making, the system interprets and responds to the input, either engaging in bidirectional communication for clarification or directly controlling the robot. This interaction enhances transparency and reduces ambiguity in HRC. Once RoboGPT deems the information sufficient, it processes responses into ROS topics to trigger robotic actions.

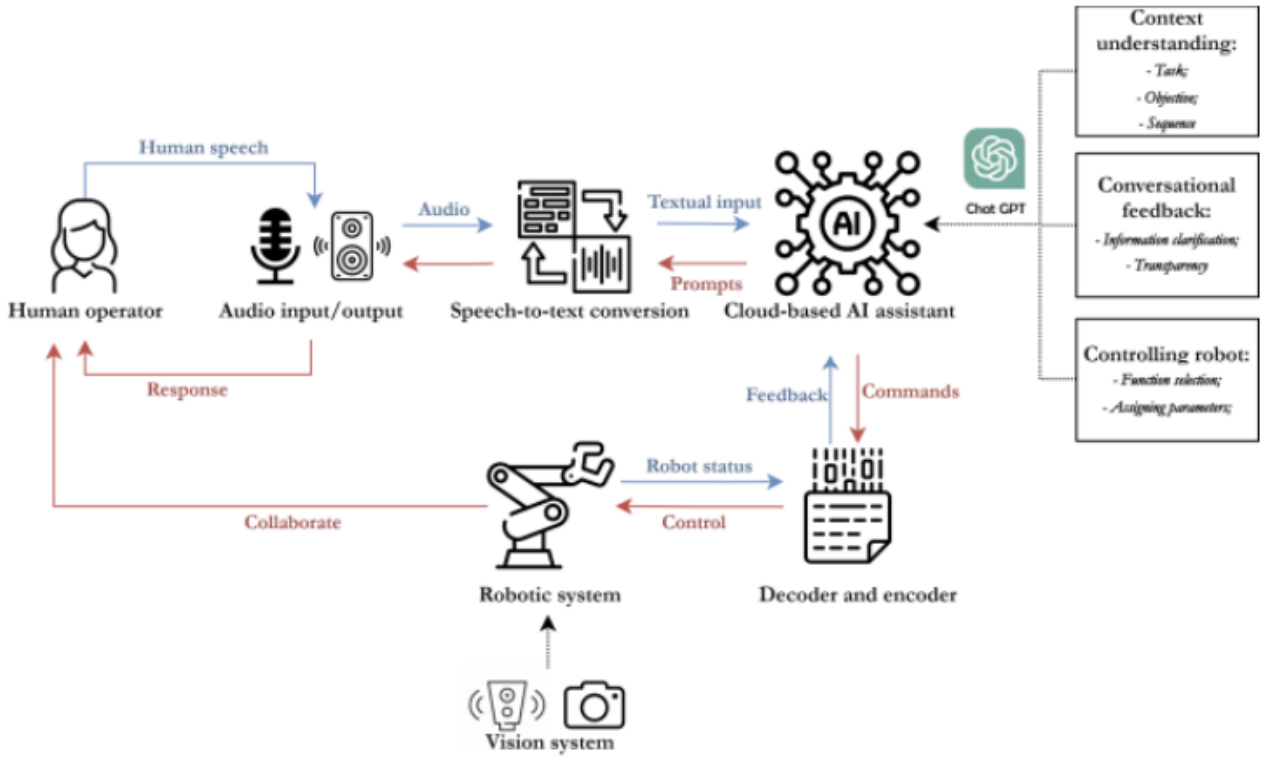


Figure 12: System workflow of RoboGPT [2].

Key findings from the study indicate that a ChatGPT-enabled robot assistant substantially improves task performance and trust levels compared to traditional control methods. This increase in trust is based on the system's competence in performing HRC tasks and its ability to remember previous interactions, like the locations for tool delivery, thereby fostering improved human-robot partnerships.

Working with a ChatGPT-enabled assistant was found to be less mentally taxing for participants, likely due to the system's autonomy and intelligence, which reduced the need for continu-

ous robot control. However, the study also highlights some limitations. The autonomous nature of the ChatGPT assistant, coupled with its understanding of scenarios, sometimes led to assertive behavior and decision-making challenges in instances of miscommunication. Moreover, the reliance on a text-based input/output format posed difficulties in effectively conveying complex real-world scenarios, such as object locations or robot status. To overcome these challenges, the study suggests future enhancements, including the integration of multi-modal understanding, like visual inputs, to refine interactions in HRC.

2.3.2 Comparative Analysis: NAO Robot and RoboGPT

This section describes the primary distinctions between the NAO humanoid robot, as utilized in this study, and the RoboGPT model referenced in related research.

NAO Robot: The NAO robot, a programmable humanoid standing 58 cm tall, boasts an array of functionalities including walking, speaking, and the ability to recognize objects and individuals. Equipped with a suite of sensors and actuators, it possesses a repertoire of predefined movements. In the context of this project, the integration of NAO with GPT primarily aims to augment its conversational capabilities, thereby enriching the naturalness and context-awareness of its interactions. Typically employed in education and research, NAO's design emphasizes versatility and generalist capabilities.

RoboGPT: In contrast, RoboGPT is tailored towards specialized tasks, characteristic of an industrial robot arm. Its design, featuring 7 degrees of freedom, allows for extensive motion range, crucial for precise object manipulation and specific movements in laboratory or industrial environments. A notable aspect of RoboGPT is the application of ChatGPT for interpreting commands and translating them into ROS messages, thus facilitating meticulous control over the robot arm's movements.

3 Methods

For a comprehensive understanding of the capabilities and evolution of humanoid robots and the integration of AI with humanoid robots, this research was divided into two main areas: Theoretical Research and Software Development Research.

3.1 Theoretical Research

The first phase of the research was to explore the evolution of humanoid robotics and the rise of LLMs. A literature review was conducted according to systematic review guidelines [29]. The literature review process was divided into three key stages, which are described in 3.1.1, 3.1.2 and 3.1.3.

3.1.1 Source Gathering and Search Strategy

The search process to identify relevant scientific studies was initiated in September 2023 and continued until mid-December. Trusted scientific databases and search engines such as Google Scholar, ScienceDirect, Web of Science and academic journals from publishers such as Elsevier, Springer, IEEE and ACM were used to compile potentially relevant literature. These databases were searched for English-language papers published up to 2023. The search included keywords such as 'evolution of humanoid robots', 'evolution of LLMs', 'AI and humanoid robots' and 'ChatGPT and humanoid robots'.

3.1.2 Evaluation and Screening

As illustrated in Figure 13 for the formal screening 302 potentially relevant studies were retrieved. A total of 237 studies were screened out due to their specific focus on the application of humanoid robots in fields such as medicine, agriculture, or ethical considerations of using humanoid robots and AI, which fell outside the scope of the research question. Inclusion or exclusion of remaining published papers was initially determined by assessing their abstracts, introductions, and keywords. The inclusion criteria focused on studies addressing the evolution of humanoid robots, the development of LLMs, and the exploration of the limitations and potential of these technologies. Furthermore, an additional 25 studies were excluded on account of duplication or inaccessibility of the full texts.

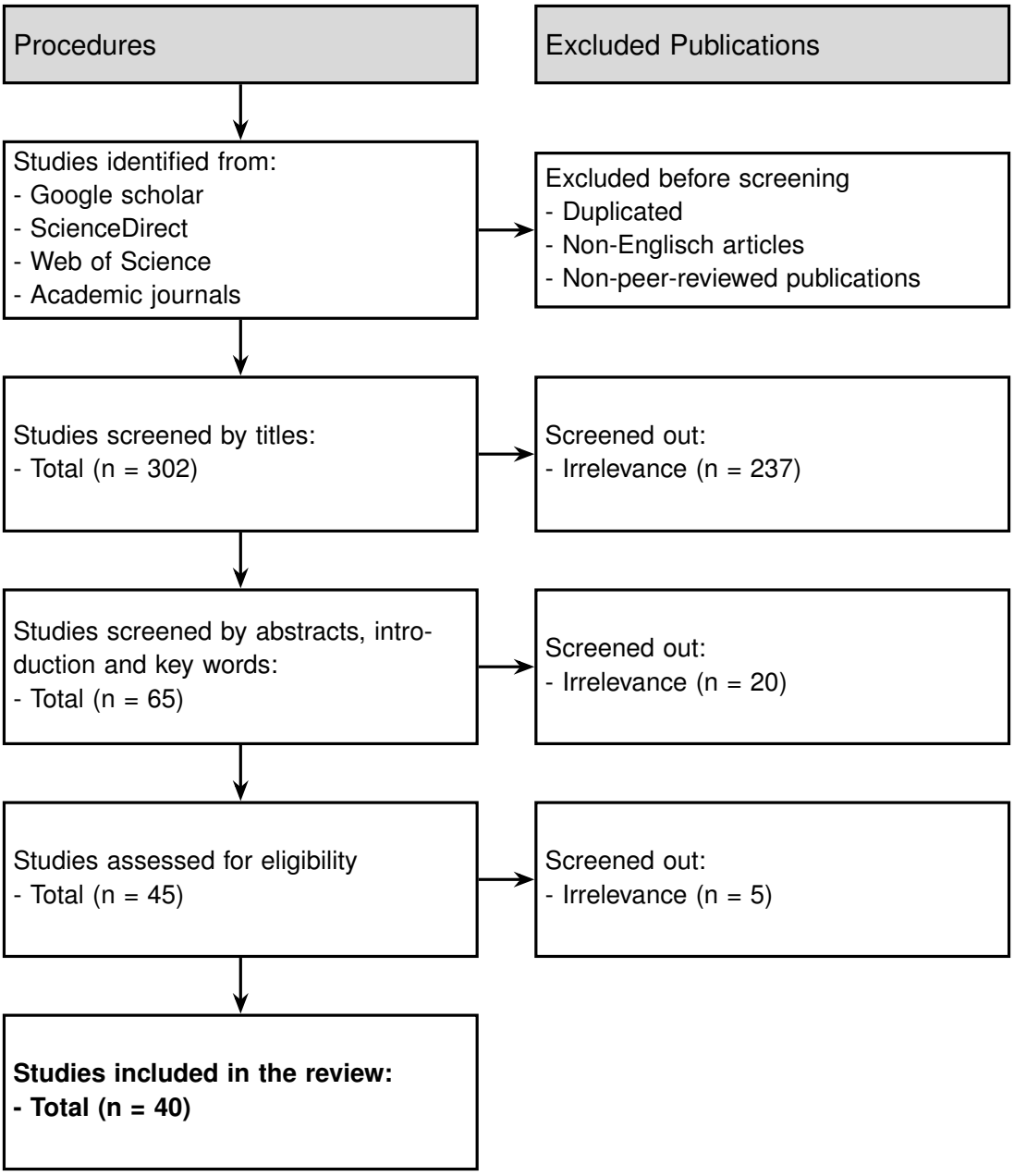


Figure 13: Literature review process [30].

3.1.3 Data Extraction and Synthesis

Following the careful selection process, 40 studies were deemed for depth data extraction. The extraction was systematically conducted to ensure that relevant information from each study was systematically conducted to ensure that relevant information from each study was captured comprehensively. This process involved distilling key findings, methodologies, results and conclusions. The extracted information was collated into a master document, which served as a foundation for comparative analysis and synthesis. The summarization aimed to distill the essence of each study, allowing for a clear overview of the collective insights gained.

3.2 Software Development Research

The second phase of the research centered on comprehending the software and hardware aspects of the NAO robot, a crucial component for the successful integration of ChatGPT. Insights into the maintenance and technical specifications of NAO were primarily gleaned from the Softbank Robotics documentation. The NAO documentation and NAOqi Developer Guide, proved particularly pertinent to this thesis. The NAO documentation is systematically divided into three main sections:

- **NAO - User Guide**: This section provides comprehensive guidelines on the initial setup, daily operation, configuration, and maintenance of the NAO robot, essential for ensuring its optimal functioning.
- **NAO - Developer Guide**: This guide offers an in-depth technical overview, covering aspects such as different model versions, detailed body specifications, supported programming languages, kinematic data, and an exhaustive list of actuators and sensors integral to the robot's operations.
- **ASK NAO Interface - V2 - User Guide**: Specifically designed to facilitate interaction with children with autism and to assist educators in classroom settings, this guide reflects the versatile application of NAO in educational and therapeutic environments.

The NAOqi Developer Guide elaborates on various branches that provide essential support for software development. For this thesis, the branches related to the Software Development Kits (SDKs), additional tutorials, and NAOqi APIs were of particular relevance. The SDKs segment delineates the range of supported programming languages, including Python, C++, Java, JavaScript, and the ROS Interface.

This wide array of programming options led to the selection of Python for this project, a decision driven by its simplicity and the extensive range of libraries available in its ecosystem. Python's user-friendly nature accelerates the development and prototyping process, making it an ideal choice, especially for beginners or in scenarios requiring rapid iteration of ideas. Moreover, Python's extensive and active community significantly enriches the development experience, providing an array of resources for learning and problem-solving, such as comprehensive documentation, diverse tutorials, and forums for active discussion.

However, it's noteworthy that Python presents certain limitations, notably its reliance on third-party libraries. This dependency can lead to version compatibility issues, especially in environments where installing additional packages is constrained. One of the initial challenges faced in this project involved addressing the compatibility discrepancies between the NAO SDK, confined to Python 2.7, and the newer libraries required for voice recognition and the GPT API, which are only compatible with Python 3. The strategies employed to resolve these issues are elaborated in 3.2.1 and 3.2.2.

3.2.1 Community-Sourced Python Solution

The initial phase entailed a comprehensive review of publicly accessible code, predominantly sourced from GitHub, a prominent code hosting platform. In this exploration, source code was identified that enabled communication between the NAO robot and ChatGPT. However, a detailed

examination of this code revealed limitations, particularly the absence of documentation beyond basic installation instructions. Furthermore, none of the identified software systems supported real-time audio streaming between the NAO robot and speech recognition engines. They either recorded audio for a fixed duration or did not utilize the NAO robot's microphones at all, opting instead for audio input from a laptop's microphone. Nevertheless, the exploration provided valuable insights into how other developers had approached similar challenges, offering potential starting points for further development.

3.2.2 Python's Web Application Gateway

To bridge the gap between Python 2.7 and Python 3, several options were evaluated:

1. **Pyro4:**
    - Python Remote Objects (Pyro) is a library facilitating advanced Remote Procedure Call (RPC) in Python.
    - Designed for creating distributed applications, Pyro enables Python objects to be invoked and managed across different machines within a network.
    - Pyro4, the specific version used, focuses on remote object management and network communication.
2. **Flask:**
    - Flask is a micro web framework for developing web applications.
    - It offers tools, libraries, and technologies for web application development, including features like routing, template rendering, and handling Hypertext Transfer Protocol (HTTP) requests and responses.
    - Renowned for its simplicity and extensibility, Flask is commonly used in smaller projects or as a foundational component in larger applications.
3. **FastAPI:**
    - FastAPI is a modern, high-performance web framework for building APIs with Python 3.8 or newer, utilizing standard Python type annotations.
    - It is noted for its speed and efficiency, including its ability to automatically create OpenAPI documentation.
    - Often employed in projects requiring rapid API development, FastAPI is particularly advantageous in contexts involving machine learning and data processing.

After careful consideration, Flask was selected to bridge Python 2 and Python 3 for the following reasons:

- **Simplicity and ease of use:** Flask's straightforward and minimalist design is ideal for creating a lightweight bridge between the two Python versions.
- **Flexibility:** Flask is extendable and allows for easy integration of various functionalities, ensuring compatibility with both Python versions.

- **Community support:** Flask has extensive community support and documentation, providing valuable resources for development and troubleshooting.
- **Alignment with project requirements:** Flask's capabilities are well-suited for the project's web-based interaction requirements, making it an optimal choice for integration.
- **Web-API integration:** it can created a small web-API with Flask which acts as an intermediary between the two Python versions. The Python 3 client can send HTTP requests to the Python 2 server to utilize the functionalities of ChatGPT and the speech recognition library.

The completion of the Software Development Research phase represents the culmination of efforts in addressing various challenges and exploring multiple solutions for effectively integrating ChatGPT with the NAO robot. The choice of Python as the programming language, coupled with resolving compatibility issues between different Python versions, were key developments in this phase. Utilizing Flask for web-API integration proved to be a strategic choice, enabling smooth communication between the NAO robot's hardware and ChatGPT's advanced functionalities.

4 Results

4.1 Overview

As outlined in Figure 14, the developed system presents an intricate interplay between a Python 3 client, a Python 2 server, OpenAI's GPT, and the NAO robot. Central to this system is the integration with OpenAI's GPT model, which plays a pivotal role in processing and generating human-like textual responses. The Python 3 client, `brain.py`, serves as the crucial link between the user's spoken input and the GPT model. It receives an audio stream from the Python 2 server, `body.py`, transcribes it to text, and sends it to the GPT API for response generation. This response is then relayed back to the Python 2 server, labeled as `body.py`, which in turn sends it to the NAO robot. The latter serves as the tangible interface of this system, recording speech from the user and playing back the answer from GPT through its speakers. Through this sophisticated setup, the system not only simulates human-like conversation but also enriches the user experience with the tangible, interactive capabilities of the NAO robot.

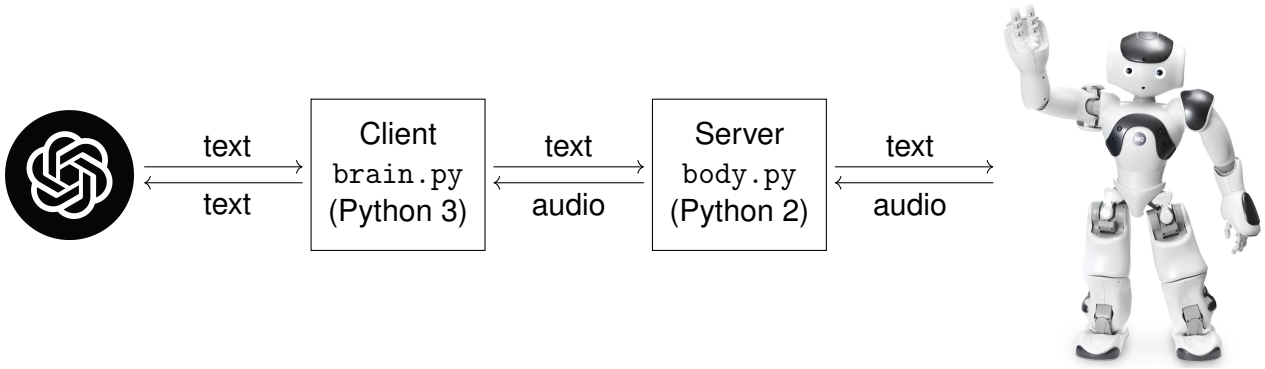


Figure 14: System interaction diagram [31, 32].

The rationale behind opting to transcribe audio in the Python 3 component arises from the deprecation of Python 2 and the limited availability of speech recognition libraries in Python 2. By doing so, the code is less prone to obsolescence, as it reduces dependency on libraries that may soon become unmaintained in Python 2. This strategic decision enhances the system's longevity and adaptability to evolving technological landscapes.

4.2 Python 2 Server

This section describes the implementation of the Python 2 server designed to interface with the NAO robot, enabling it to process and respond to audio inputs using GPT. The server is built using the Flask framework and integrates with NAO's Python SDK, NAOqi.

4.2.1 Audio Capture Module

A custom module, `AudioCaptureModule`, extends the `ALModule` class from the NAOqi framework to manage audio capture. It provides methods for starting and stopping audio capture, processing incoming audio data, and retrieving audio chunks. It is based on examples from the documentation [33, 34].

```
class AudioCaptureModule(ALModule):

    def __init__(self, name):
        ALModule.__init__(self, name)
        self.audio_device = ALProxy("ALAudioDevice", nao_IP, nao_port)
        self.is_listening = False
        self.buffers = []

    def start_listening(self):
        self.audio_device.setClientPreferences(self.getName(), 16000, 3, 0)
        self.audio_device.subscribe(self.getName())
        self.is_listening = True

    def stop_listening(self):
        self.audio_device.unsubscribe(self.getName())
        self.is_listening = False

    def processRemote(self, nbOfChannels, nbOfSamplesByChannel, timeStamp, ↙
↳ inputBuffer):
        if self.is_listening:
            self.buffers.append(inputBuffer)

    def get_audio_chunk(self):
        if self.buffers:
            return self.buffers.pop(0)
        else:
            return None
```

Codeblock 1: Audio capture module.

The `start_listening` function subscribes to the `audio_device` (see , indicating readiness to receive audio buffer, and sets the sampling rate to $f_s = 16$ kHz. Empirically, it was found out that each buffer contains $N = 1365$ samples and that each sample is represented by 16 bits. The size of each buffer is therefore $N \times 16$ bits $= 21840$ bits $= 2730$ bytes. The time interval between the

arrival of two consecutive buffers can therefore be calculated as follows:

$$T = \frac{N}{f_s} = \frac{1365}{16\ \text{kHz}} = 85.3125\ \text{ms} \tag{3}$$

Furthermore, the audio samples within each buffer adhere to little-endian encoding. In this format, the first byte of each pair constitutes the lower 8 bits, and the second byte constitutes the upper 8 bits of the 16-bit sample.

A critical aspect of the `AudioCaptureModule` is the `processRemote` function. This function serves as a callback mechanism and is invoked automatically whenever new audio data from NAO's microphones is available [35]. The function's design allows for immediate access and processing of audio data as it is received, making it essential for tasks that require rapid response, such as interactive communication. In the context of this project, this means the speech recognition engine on the client side can recognize when a person has stopped talking and immediately start transcribing the audio to text, leading to more natural and effective interactions.

4.2.2 Server Endpoints

In the developed Flask application, several endpoints are implemented to facilitate communication between the server and the NAO robot. These endpoints are the interface by which the `brain.py` client can interact with the `body.py` server. These endpoints allow for various operations such as initiating speech, starting and stopping audio capture, and retrieving recorded audio chunks from the robot.

```
@app.route('/get_audio_chunk', methods=['GET'])
def get_audio_chunk():
    audio_data = AudioCapture.get_audio_chunk()
    if audio_data is not None:
        return audio_data
    else:
        while audio_data is None:
            audio_data = AudioCapture.get_audio_chunk()
            time.sleep(sleep_time)
        return audio_data
```

Codeblock 2: Server endpoints.

The `/talk` endpoint allows the NAO robot to speak. When a POST request is sent to it, the server receives a JavaScript Object Notation (JSON) payload containing the message to be spoken. The message is then passed to the NAO robot's speech synthesis module. Utilizing *animated speech*, NAO performs gestures and movements aligned with the message's content, allowing for more natural interaction with the robot [36]. The `/start_listening` endpoint is used to command the NAO robot to start listening through its microphones. This function activates the audio capture capabilities of the robot. Conversely, the `/stop_listening` endpoint instructs the NAO robot to stop listening. It is used to deactivate the audio capture process. The `/get_audio_chunk` endpoint, shown in Codeblock 2 enables the retrieval of captured audio data from the robot. This endpoint is the working horse of the Flask server; it returns the audio chunk if available; otherwise, it waits until audio data is received. The `/get_server_buffer_length`

endpoint, as the name suggests, simply returns the current length of the audio buffer on the server side.

4.2.3 Python Broker

The functionality of the Python 2 server, designed to interface with the NAO robot, is facilitated through the implementation of a Python broker, specifically the `ALBroker` component from the NAOqi SDK. This broker serves as a communication mediator, essential for the interaction between the server and the robot's system.

```
try:
    pythonBroker = ALBroker("pythonBroker", "0.0.0.0", 0, nao_IP, nao_port)
    global AudioCapture
    AudioCapture = AudioCaptureModule("AudioCapture")
except RuntimeError:
    exit(1)
```

Codeblock 3: Initialization of Python broker.

In the code, the `ALBroker` is configured with the host address `0.0.0.0`, enabling it to listen on all network interfaces of the host machine. The usage of port `0` allows for dynamic port assignment by the operating system. The broker establishes a connection to the NAO robot using the provided `nao_IP` and `nao_port`, forming the primary communication channel between the server and the robot.

This broker plays a critical role in the instantiation and operation of the `AudioCaptureModule`. The module, which is pivotal for handling audio capture from the NAO robot, depends on the broker for communication with the robot's audio system. In instances where the broker initialization fails, indicated by a `RuntimeError`, the script is designed to output an error message and terminate. This response is a practical necessity, as the broker's functionality is integral to the server's ability to interact with the robot, particularly for audio data processing and management.

4.3 Python 3 Client

4.3.1 Audio Source Class

The `NaoAudioSource` class, a crucial component of the speech recognition system, is tailored for acquiring audio from the NAO robot. It inherits from `sr.AudioSource` provided by the `speech_recognition` library, which facilitates seamless integration with various speech recognition engines.

```
class NaoAudioSource(sr.AudioSource):

    def __init__(self, server_url=BODY_URL):
        self.server_url = server_url
        self.stream = None
        self.is_listening = False
        self.CHUNK = 1365
        self.SAMPLE_RATE = 16000
        self.SAMPLE_WIDTH = 2

    def __enter__(self):
        requests.post(f"{self.server_url}/start_listening")
        self.is_listening = True
        self.stream = NaoStream(self.audio_generator())
        return self

    def audio_generator(self):
        while self.is_listening:
            response = requests.get(f"{self.server_url}/get_audio_chunk")
            yield response.content
            {current_buffer_length = requests.get(f"{self.server_url}/get_server_buffer_length").json()["length"]
            correcting_factor = 1.0 / (1.0 + np.exp(current_buffer_length - np.pi))
            corrected_time_between_audio_chunks = time_between_audio_chunks * correcting_factor
            time.sleep(corrected_time_between_audio_chunks)

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.is_listening = False
        requests.post(f"{self.server_url}/stop_listening")
```

Codeblock 4: Speech recognition class.

Upon instantiation, the `NaoAudioSource` class initializes with a default server Uniform Resource Locator (URL). This URL points to the Flask server that communicates with the NAO robot. The class has attributes for managing the audio stream and listening state, along with constants for audio chunk size, sample rate, and sample width.

The class uses Python's context management protocol to handle the audio stream. The `__enter__` method starts the audio stream by sending a request to the server to begin listening

and initializes the `NaoStream` object. Conversely, the `__exit__` method stops the audio stream by signaling the server to stop listening. The core functionality of the class lies in its `audio_generator` method. This generator continuously requests audio chunks from the server while the robot is listening. It implements a dynamic wait system to adjust the time between audio chunk requests based on the server's buffer length. To account for the latency of other instructions, the time interval calculated in Equation 3 needs to be adjusted. This is done by multiplying it by a `correcting_factor`, which is calculated using a (shifted) sigmoid function:

$$\texttt{correcting\_factor} = \frac{1}{1 + e^{(\texttt{current\_buffer\_length} - \pi)}} \tag{4}$$

This function is chosen for its smooth, S-shaped curve (see Figure 15), making it ideal for gradual transitions between two values. It finds applications in various fields, ranging from astrophysics [37] to deep learning [38]. It provides a simple yet elegant method to dynamically adjust the time interval between consecutive server requests, ensuring efficient handling of incoming audio data and minimizing latency.

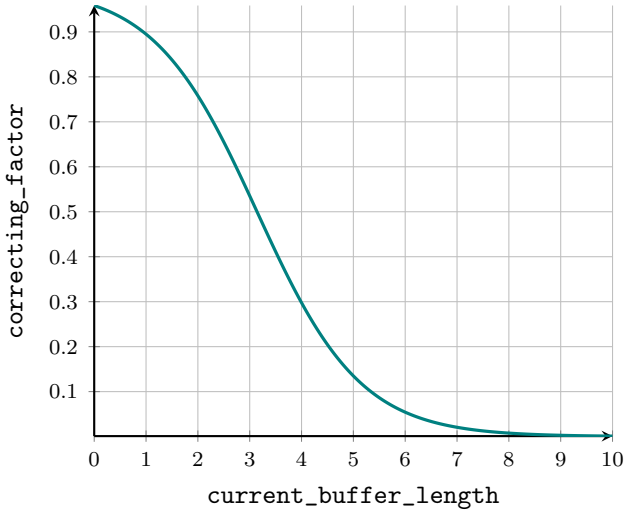


Figure 15: Correcting factor as a function of current buffer length.

4.3.2 Speech Recognition

Using a `with` statement to instantiate a `NaoAudioSource` object automatically invoking its `__enter__` method to initiate the audio stream. The `recognizer.listen` method from the `speech_recognition` library then captures audio as long as somebody is talking or until a timeout is reached, which is determined based on the energy of the audio [39]. The use of context management through the `with` statement guarantees that the `__exit__` method is called, stopping the audio stream and releasing any resources held by the `NaoAudioSource` object.

Once the audio data is captured, the `recognize_google` function of the `speech_recognition` library can be employed to convert this audio data into text.

4.3.3 Conversation Loop

```
conversation_context = [{"role": "system", "content": system_prompt}]
running = True
while running:
    user_message = get_user_text()
    conversation_context.append({"role": "user", "content": user_message})
    gpt_message = get_gpt_text(conversation_context)
    send_gpt_text_to_body(gpt_message)
    conversation_context.append({"role": "assistant", "content": gpt_message})
```

Codeblock 5: Conversation loop.

The conversation loop in the system facilitates a context-aware dialogue between the user and the robot. It starts by establishing the `conversation_context` with the system's initial prompt. A `while` loop, based on the `running` variable, allows ongoing communication until it's stopped.

Each iteration captures the user's spoken words as text via `get_user_text`, adding this to `conversation_context`. The `get_gpt_text` function then uses this context to produce an AI-generated response. This response is audibly relayed through `send_gpt_text_to_body`, and added to the `conversation_context` for a complete conversation history that will be used in the next iteration. Figure 16 visualizes the conversation loop.

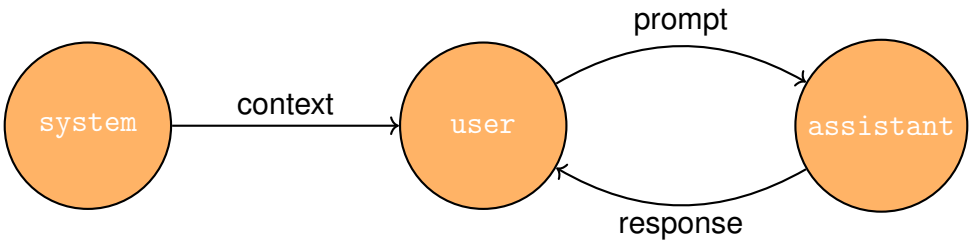


Figure 16: Conversation loop.

5 Discussion

5.1 Capabilites of the Developed System

In this project, the integration of transformer-based LLMs, specifically GPT-3.5 Turbo, with the NAO humanoid robot, has led to notable improvements in the robot's conversational capabilities. This integration marks a departure from the limitations of earlier NAO models and similar humanoid robots, which primarily relied on scripted interactions.

Previous versions of humanoid robots, including past iterations of NAO, often depended on predetermined responses, resulting in conversations that lacked depth and adaptability. The use of GPT-3.5 Turbo, however, enables more nuanced and context-aware dialogues, closely mirroring human conversational dynamics. This development enhances NAO's ability to interact in a manner that is more reflective of natural speech patterns, representing a step forward in the application of AI in humanoid robotics.

A key feature of this project is the effective use of NAO's inbuilt microphone and speakers, optimizing autonomous speech recognition and output capabilities. The system's ability to detect pauses in speech autonomously contributes to smoother conversational exchanges. Additionally, the improved response speed from GPT-3.5 Turbo aids in facilitating more fluid interactions, offering an improvement over the conversational functions of earlier humanoid robot models.

5.2 Technical Hurdles and Limitations

Throughout this research, several technical challenges were encountered, necessitating innovative solutions. A primary issue was the reliance of NAO's SDK on the outdated Python 2, posing significant integration challenges due to compatibility issues with modern software environments. This limitation hindered a seamless installation of necessary libraries, reflecting a broader challenge in maintaining forward compatibility with evolving technological standards.

The project also faced hurdles with inadequate documentation for critical implementations. For example, existing documentation often lacked clarity in the use of constants and parameters, requiring extensive investigation to comprehend and apply. This issue underscores the importance of comprehensive and clear documentation in complex technical projects.

Furthermore, the hardware limitations of NAO, particularly the microphone quality and fan noise, posed substantial obstacles for effective audio processing. To ensure effective communication, it was necessary to speak loudly and clearly, preferably directly towards the robot. Various methods were attempted to mitigate these issues, such as using the `reduce_noise` functionality of the `noisereduce` library [40] for post-processing as well as the `adjust_for_ambient_noise` function from the `speech_recognition` library [39] for ambient noise adjustment. Surprisingly, these attempts often resulted in lower performance of the speech-to-text engine compared to using the raw audio, even though the audio seemed clearer to human listeners.

Additionally, the project faced persistent challenges in integrating the developed code with Docker containers and Ubuntu systems. This issue added layers of complexity to the development process. The initial objective was to containerize the developed code using Docker, thereby facilitating its deployment across various operating systems. However, this goal could not be realized due to recurring integration errors and compatibility issues. The inability to achieve seamless Docker integration highlights an area for further investigation and potential refinement in future development phases.

5.3 Future Work

Looking towards future work, several promising enhancements are proposed.

The integration of visual input, possibly along with other sensory modalities, could play a key role in enriching the system's orientation and context-awareness capabilities. Such enhancements may pave the way for autonomous navigation within various environments. In this regard, additional functionalities from NAOqi for movement could be integrated.

An open-source LLM could even be fine-tuned for the purpose of controlling NAO. Research in this area has already been conducted on robots with 6 degrees of freedom: Pre-trained on vision-language data from the web, RT-2 demonstrates that transformer-based vision-language-action models can be fine-tuned on robotic data to perform a variety of manipulation tasks, such as picking up objects and opening drawers [41].

In the domain of HRI, further system enhancements are envisaged to facilitate more naturalistic human-like communication. Currently, when the system developed in this project is queried, it provides responses from the GPT model without the capability for interruption, regardless of the response length. This static response mechanism contrasts sharply with human conversational dynamics, where interruptions for clarifications, such as requesting a repetition of a missed word, are common. Enhancing the system to support interruptible and adaptive response generation could significantly improve the interactive experience, aligning more closely with the fluid nature of human communication. This would involve developing algorithms capable of detecting cues for interruption and appropriately adjusting the response flow, thereby emulating more closely the interactive patterns observed in human dialogue.

Incorporating a sophisticated cognitive architecture, like the Autonomous Cognitive Entity (ACE) framework, could significantly enhance the nuances of contextual comprehension and responsiveness in dynamic settings. ACE is a novel cognitive architecture designed to enable machines and software agents to operate more independently, leveraging the latest Generative AI technologies. The ACE framework consists of six distinct layers - Aspirational Layer, Global Strategy, Agent Model, Executive Function, Cognitive Control, and Task Prosecution - which collectively guide moral decision-making, strategic thinking, task execution, and adaptability. This model aims to formalize the structure of autonomous, agentic systems, enhancing their robustness and flexibility [42].

5.4 Conclusion

The successful integration of ChatGPT with the NAO humanoid robot marks a noteworthy advancement in enhancing robotic conversational abilities. Key achievements include overcoming technical challenges related to software compatibility, notably between different versions of Python, and employing Flask for effective system integration.

Despite facing hardware limitations and Docker integration challenges, the project successfully improved NAO's ability to conduct context-aware conversations. These advancements indicate potential for wider applications in HRI.

The integration offers a foundation for future enhancements in humanoid robotics, suggesting directions for incorporating advanced sensory and cognitive features.

Bibliography

[1] Sai Vemprala et al. “Chatgpt for robotics: Design principles and model abilities”. In: *Microsoft Auton. Syst. Robot. Res* 2 (2023), p. 20.

[2] Yang Ye, Hengxu You, and Jing Du. “Improved Trust in Human-Robot Collaboration With ChatGPT”. In: 11 (2023), pp. 55748–55754. ISSN: 2169-3536. URL: http://dx.doi.org/10.1109/access.2023.3282111.

[3] Edirlei Soares de Lima and Bruno Feijó. “Artificial Intelligence in Human-Robot Interaction”. In: Sept. 2019, pp. 187–199. ISBN: 978-3-319-96721-9.

[4] Shuji Hashimoto et al. “Humanoid Robots in Waseda University—Hadaly-2 and WABIAN”. In: *Auton. Robots* 12 (Jan. 2002), pp. 25–38. DOI: 10.1023/A:1013202723953.

[5] Ichiro Kato et al. “The robot musician ‘wabot-2’(waseda robot-2)”. In: *Robotics* 3.2 (1987), pp. 143–155.

[6] Antonio Zavarce. *Get to Know the Fascinating World of the 5 Most Famous Robots in the World*. Accessed: 2023-12-21. 2023. URL: https://inspenet.com/en/articulo/get-to-know-the-fascinating-world-of-the-5-most-famous-robots-in-the-world/.

[7] Aldebaran Robotics. *NAO*[6]. Accessed: 2023-12-22. 2023. URL: https://www.aldebaran.com/en/nao.

[8] Aida Amirova et al. “10 Years of Human-NAO Interaction Research: A Scoping Review”. In: *Frontiers in Robotics and AI* 8 (Nov. 2021). ISSN: 2296-9144. URL: http://dx.doi.org/10.3389/frobt.2021.744526.

[9] R. Gelin. “NAO”. In: *Humanoid Robotics: A Reference*. Springer Netherlands, Oct. 2018, pp. 147–168. URL: http://dx.doi.org/10.1007/978-94-007-6046-2_14.

[10] Aldebaran Robotics. *Hardware*. Accessed: 2023-12-21. URL: http://doc.aldebaran.com/1-14/family/body_type.html.

[11] Aldebaran Robotics. *NAO Key Feature Audio Signal Processing*. URL: https://www.generationrobots.com/media/NAO%20Next%20Gen/FeaturePaper(AudioSignalProcessing)%20(1).pdf.

[12] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). URL: http://arxiv.org/abs/1706.03762.

[13] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *ArXiv* 1409 (2014). URL: https://arxiv.org/abs/1409.0473v1.

[14] Lena Voita. *Sequence to Sequence (seq2seq) and Attention*. accessed 2023-12-16. 2023. URL: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html.

[15] Andrej Karpathy. *Let's build GPT: from scratch, in code, spelled out*. Accessed: 2023-12-16. 2023. URL: https://www.youtube.com/watch?v=kCc8FmEb1nY.

[16] Zhen Qin et al. *cosFormer: Rethinking Softmax in Attention*. 2022. arXiv: 2202.08791 [cs.CL].

[17] Sebastian Raschka. *Understanding Self-Attention from Scratch*. Accessed: 2023-12-16. 2023. URL: https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html.

[18] Huyen Chip. *RLHF: Reinforcement Learning from Human Feedback*. accessed 2023-12-16. 2023. URL: https://huyenchip.com/2023/05/02/rlhf.html#mathematical_formulation_sft.

[19] Chunting Zhou et al. *LIMA: Less Is More for Alignment*. 2023. arXiv: 2305.11206 [cs.CL].

[20] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[21] Yoshua Bengio et al. "A Neural Probabilistic Language Model". In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 1137–1155. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=944919.944966.

[22] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[23] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023. URL: http://udlbook.com.

[24] Murat Aydogan. "COMPARISON OF WORD EMBEDDING METHODS FOR TURKISH SENTIMENT CLASSIFICATION". In: Dec. 2020, pp. 67–86. ISBN: 978-625-7319-20-1.

[25] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: 2018. URL: https://arxiv.org/abs/1810.04805.

[26] Gokul Yenduri et al. "Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions". In: *ArXiv* abs/2305.10435 (2023). URL: https://api.semanticscholar.org/CorpusID:258762263.

[27] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[28] Abi Aryan and Andrew McMahon. *Introduction to Large Language Models*. accessed 2023-12-16. 2023. URL: https://blog.sylphai.com/introduction-to-large-language-models.

[29] Hayrol Azril Mohamed Shaffril, Samsul Farid Samsuddin, and Asnarulkhadi Abu Samah. "The ABC of systematic literature review: the basic methodological guidance for beginners". In: *Quality & Quantity* 55.4 (Oct. 2020), pp. 1319–1346. ISSN: 1573-7845. URL: http://dx.doi.org/10.1007/s11135-020-01059-6.

[30] Sehwan Chung et al. "Comparing natural language processing (NLP) applications in construction and computer science using preferred reporting items for systematic reviews (PRISMA)". In: *Automation in Construction* 154 (2023), p. 105020. ISSN: 0926-5805. URL: https://www.sciencedirect.com/science/article/pii/S0926580523002807.

[31] Freelogopng. Accessed: 2023-12-01. URL: https://freelogopng.com/image/861.

[32] HiClipart. Accessed: 2023-12-01. URL: https://www.hiclipart.com/free-transparent-background-png-clipart-jrpbr.

[33] Aldebaran Robotics. *NAOqi ALAudioDevice - Process microphone signals*. Accessed: 2023-11-28. 2018. URL: http://doc.aldebaran.com/2-8/dev/python/examples/audio/audio_soundprocessing.html.

[34] Carnegie Mellon University. *Making a Python module - Reacting to events*. Accessed: 2023-12-08. 2011. URL: https://www.cs.cmu.edu/~cga/nao/doc/reference-documentation/dev/python/reacting_to_events.html#python-reacting-to-events.

[35] Edgar Lopez-Caudana et al. "Classification of materials by acoustic signal processing in real time for NAO robots". In: *International Journal of Advanced Robotic Systems* 14.4 (2017). URL: https://doi.org/10.1177/1729881417714996.

[36] Aldebaran Robotics. *NAOqi ALAnimatedSpeech*. Accessed: 2023-12-13. 2018. URL: http://doc.aldebaran.com/2-8/naoqi/audio/alanimatedspeech.html.

[37] Sergio Torres-Arzayus et al. *Evaluating a Sigmoid Dark Energy Model to Explain the Hubble Tension*. 2023. arXiv: 2311.05510 [astro-ph.CO].

[38] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. 2022. arXiv: 2109.14545 [cs.LG].

[39] Anthony Zhang et al. *Speech Recognition*. Version 3.8. 2017. URL: https://github.com/Uberi/speech_recognition#readme.

[40] Tim Sainburg. *timsainb/noisereduce: v1.0*. Version db94fe2. June 2019. URL: https://doi.org/10.5281/zenodo.3243139.

[41] Anthony Brohan et al. *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control*. 2023. arXiv: 2307.15818 [cs.RO].

[42] David Shapiro et al. "Conceptual Framework for Autonomous Cognitive Entities". en. In: (2023). URL: https://rgdoi.net/10.13140/RG.2.2.14161.30569.