Guardian of the Aisles: Enhancing Customer Experience in Shopping Malls with Pepper

Group 04: Agostino Cardamone [2276], **Chiara Ferraioli** [2169], **Asja Antonucci** [2437], **Viktor Dobrev** [2331]

[1]Dept. of Information Eng., Electrical Eng. and Applied Mathematics - University of Salerno, Italy

Abstract This project introduces an AI-assistant robotic system designed for shopping malls, with **Pepper** at its core, acting as both a guardian and an interactive assistant. Pepper utilises a connected camera and a video analytics system that detects individuals and identifies their attributes, such as gender and accessories. Instead of processing this data autonomously, Pepper retrieves the analysed results from a database, allowing it to engage with visitors through spoken interaction. This enhances the customer experience by providing a seamless interface to the mall's sophisticated data processing system.

Beyond real-time visitor insights, Pepper also has access to information regarding the **Artificial Vision Contest 2025**, specifically details about the participating groups. These details include group identification numbers, lists of team members, as well as their scores and final rankings. By integrating both surveillance-based insights and competition-related data, Pepper delivers an enriched and interactive experience within the shopping mall environment.

For correspondence:

a.cardamone7@studenti.unisa.it (AC); c.ferraioli30@studenti.unisa.it (CF); a.antonucci5@studenti.unisa.it (AA); v.dobrev@studenti.unisa.it (VD)

1 Introduction

As artificial intelligence and robotics continue to evolve, their applications in public spaces are becoming increasingly sophisticated. Shopping malls, in particular, can benefit from AI-driven systems that enhance customer interactions, improve security, and provide valuable insights into visitor behaviour. This project explores the implementation of an intelligent robotic assistant, **Pepper**, designed to act as both a guardian and a point of interaction within the mall environment. The goal is to integrate Pepper into a broader AI-powered infrastructure, allowing it to assist visitors and provide contextual information based on real-time data.

An essential element of this system is its connection to a video analytics network, which continuously monitors the mall. Instead of processing raw footage independently, Pepper accesses a structured database containing pre-analysed information about individuals detected in the shopping centre. This enables it to identify the presence of people, retrieve attributes such as gender and accessories, and determine whether a visitor has passed by or stopped in front of specific locations, along with the frequency of such occurrences. Leveraging this information, Pepper can perform various tasks, such as helping visitors locate someone based on provided descriptions or estimating the number of people matching specific characteristics within the mall.

Beyond assisting shoppers, Pepper is also integrated with the **Artificial Vision Contest 2025**, a competition focused on advancements in computer vision technologies. The system has access to detailed records of the participating teams, including their identification numbers, member lists, performance scores, and final rankings. This allows Pepper to re-

spond to contest-related inquiries, such as revealing the top three teams, identifying the winners, or providing information about specific groups and their rankings.

The following sections will delve into the technical framework, system architecture, and implementation details of the proposed solution.

2 ROS Based Architecture

The robotic system is built using the **Robot Operating System (ROS)**, a widely used middleware that facilitates modular and scalable robotic applications.

To optimise system performance, the architecture integrates ROS's two primary communication paradigms: **topics** for continuous, asynchronous data exchange and **services** for synchronous request-response interactions. This combination enables efficient real-time processing while maintaining operational flexibility, ensuring reliable task execution, facilitating debugging, and allowing modular components to be easily reused or extended for different applications.

The following sections outline the ROS nodes implemented in this system, describing them individually or as part of broader functional modules, depending on their role within the architecture.

2.1 Video Analytics Module

The **Video Analytics Module** handles video capture, face detection, and visualisation within the system. It consists of three nodes: the **Image Publisher Node**, which streams video frames from Pepper's camera or an external source; **the Face Detector Node**, which identifies and tracks faces in the frames; and the **Visualization Node**, which overlays detection results for real-time monitoring.

The following sections describe each node's implementation and role in the system.

2.1.1 Image Publisher Node

The **Image Publisher Node** is responsible for capturing and broadcasting video frames, acting as a ROS **publisher** that streams visual data to the `/image_feed` topic. This enables any subscribing nodes to access the video feed in real time, ensuring that a visual input can be processed without creating direct dependencies between components.

To support diverse operational scenarios, the node is designed to handle video input from both **Pepper's onboard camera** and **external cameras**. When using Pepper's camera, it interfaces with **NAOqi's ALVideoDevice** service to capture RGB frames, while for external cameras, it relies on **OpenCV's VideoCapture**. This approach allows seamless adaptation to different hardware configurations while maintaining a unified video publishing mechanism.

Captured frames are converted into **ROS-compatible Image** messages using the **CvBridge** library before being published. The node also provides configurable parameters for resolution and frame rate, enabling a balance between computational efficiency and image quality based on system requirements.

To enhance reliability and resource efficiency, the node includes mechanisms for managing hardware usage. It can unsubscribe from the camera when no active subscribers are present and properly release resources upon termination, preventing unnecessary hardware allocation and improving overall system stability.

2.1.2 Face Detector Node

The **Face Detector Node** is responsible for analysing video frames to identify and localise faces within the scene. Operating as a **subscriber** to the `/image_feed` topic, it continuously receives video frames from the **Image Publisher Node** and processes them to detect faces. The detection results, including **bounding box coordinates**, **confidence scores**, and **assigned face IDs**, are then published to the dedicated `/face_detection` topic. This modular design allows other nodes to access face detection data without being directly linked to the detection process.

The Face Detection is implemented using a **pre-trained OpenCV Deep Neural Network (DNN) model**, selected for its efficiency and compatibility with ROS. The face detection

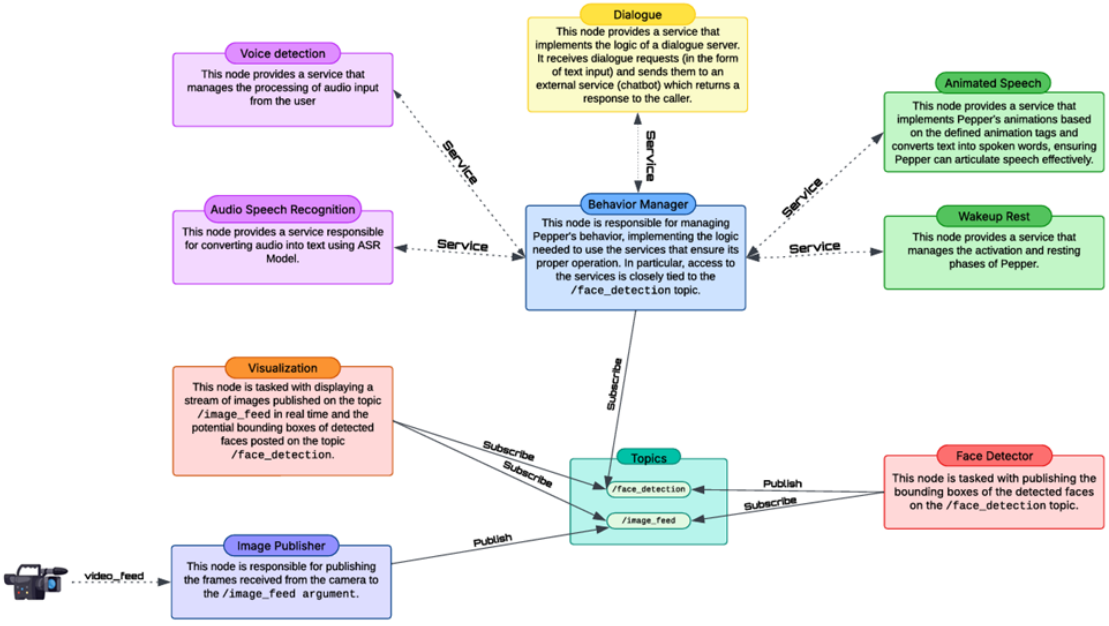

Figure 1: Design of the ROS-based architecture

pipeline begins with the initialisation of the model, which involves loading two files stored locally:

- `opencv_face_detector.pbtxt`: A configuration file defining the structure of the neural network, including layer connections and architecture details.
- `opencv_face_detector_uint8.pb`: A pre-trained model file containing the learned weights for face detection, enabling the network to recognise faces in input images.

To process video frames, the node converts the incoming **ROS Image** messages into an **OpenCV-compatible** format using **CvBridge**. Each frame undergoes preprocessing, including **input resizing**, **mean subtraction**, and **colour channel reordering**, before being fed to the DNN model to obtain the detection results.

For each detected face, the system assigns a unique face ID, enabling tracking across multiple frames, and a confidence score. Only detections with a confidence score above a predefined threshold are considered valid. The results are then formatted as **Detection2D ROS** messages, which include bounding box coordinates, confidence scores, and assigned ID. These messages are aggregated into a **Detection2DArray** message and published in real time to the `/face_detection` topic.

To maintain stable operation in multi-threaded ROS environments, the node implements thread safety mechanisms, preventing race conditions when handling image data.

To ensure that the implemented face detection component met the system's real-time performance requirements, we conducted a comparative evaluation of existing face detection models, analyzing their accuracy, computational efficiency, and ease of integration with ROS. We referenced the study "Comparative Evaluation of Face Detection Algorithms" [1], which compared **OpenCV**'s DNN-based detector and **Dlib**'s CNN-based alternative. The study found that while Dlib's model achieved a higher accuracy of 91.4% compared to OpenCV's 89.6%, it required significantly more computational resources, with an inference time of 62 ms per image on a CPU versus OpenCV's 22 ms per image. Given our system's constraint of maintaining real-time performance on standard CPUs, the additional computational overhead of Dlib's model made it impractical for our application.

Further validation of OpenCV's suitability for real-time deployment was supported by the research "A Comprehensive Review of Face Detection Technologies" [2], which assessed the viability of these models for embedded environments. The findings confirmed that OpenCV's DNN model maintains efficient processing without the need for GPU acceleration, making it an ideal choice for computationally constrained systems. This aspect was particularly relevant for our design, as our objective was to develop a flexible and scalable system without relying on specialized hardware. Moreover, OpenCV's native integration with ROS simplified implementation and long-term maintainability, whereas Dlib's model would have introduced additional dependencies and complexities in deployment.

Based on these considerations, we selected OpenCV's DNN-based face detector as the optimal solution for balancing accuracy, efficiency, and real-time processing capabilities, ensuring seamless operation within our robotic framework.

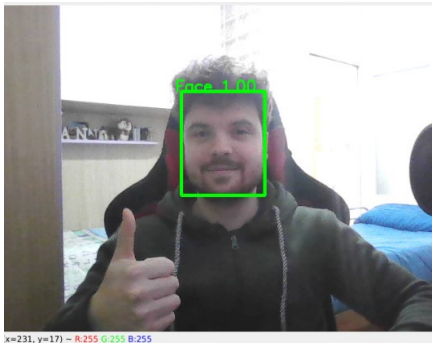

Figure 2: Example of Face Detection

2.1.3 Visualization Node

The **Visualization Node** provides real-time visual feedback by displaying video frames from the `/image_feed` topic with overlaid detection results from the `/face_detection` topic. It allows direct observation of detected faces by rendering bounding boxes and confidence scores on the video stream.

The node subscribes to `/image_feed` to receive video frames as **ROS Image** messages, which are converted into an **OpenCV-compatible** format using **CvBridge** for rendering. Simultaneously, it subscribes to `/face_detection` to retrieve detection data, including bounding box coordinates, confidence scores, and assigned face IDs. These coordinates, originally provided in a normalised format, are scaled to match the resolution of the video frames to ensure accurate placement. Once processed, the annotated frames are displayed in real time using **OpenCV's imshow** functionality.

By visually presenting detection results, the node facilitates quick assessment of face detection performance, helping to identify potential misdetections and variations due to different camera setups or lighting conditions. To ensure stable operation in a multi-threaded ROS environment, it incorporates a thread safety mechanism, preventing race conditions when handling simultaneous callbacks from the image and detection topics.

2.2 Speech To Text Module

The **Speech-to-Text Module** is responsible for capturing and transcribing spoken input, enabling voice-based interaction within the system. It consists of two nodes: the **Voice Detection Node**, which detects and records speech when requested, and the **Audio Speech Recognition (ASR) Node**, which processes the recorded audio and converts it into text using **OpenAI's Whisper** model via **Azure OpenAI**.

By organizing these tasks into distinct but interconnected nodes, the module ensures efficient audio processing, high transcription accuracy, and seamless integration with other system components. The following sections describe each node's implementation and role in detail.

2.2.1 Voice Detection Node

The **Voice Detection Node** is responsible for detecting and capturing spoken input using either the **built-in microphone** of a laptop or an **external microphone** connected to Pepper. Unlike a continuous audio stream, this node activates the microphone only when explicitly requested, ensuring that only relevant speech data is recorded and processed. This design helps filter out unnecessary background noise and optimises resource usage.

To provide this functionality, the node implements a custom **ROS service**, called `voice_detection_service`,

which allows other components to request voice input synchronously. When the service is invoked, the node activates the microphone and listens for speech input. If necessary, it can first adjust for ambient noise to improve recording quality. The process includes a timeout mechanism, ensuring that if no speech is detected within a predefined period, the operation is terminated to prevent unnecessary processing. The captured audio is recorded in **raw format** as a sequence of data chunks and returned to the requesting node for further processing.

To ensure stable operation in a multi-threaded ROS environment, the node includes a thread safety mechanism, preventing conflicts when handling simultaneous requests.

2.2.2 Audio Speech Recognition Node

The **Audio Speech Recognition (ASR) Node** is responsible for transcribing spoken input into text, enabling seamless natural language interaction. To provide this functionality, the node implements the **ROS service** `asr_service`, which allows other system components to request speech-to-text transcription synchronously. This approach ensures real-time responses while keeping the speech recognition process independent from other modules that rely on its output.

Assessing various ASR models to determine which one would work best for the system was a significant step in the design process. Several cloud-based solutions were evaluated, each with different strengths and limitations. The assessment focused on three factors: **accuracy**, **multilingual support**, and **computational efficiency**, ensuring that the chosen model would be reliable in real-world applications.

At first, **Google Speech-to-Text** appeared to be a convenient choice, mainly due to its free usage credits, which made it ideal for initial testing and development. However, as testing progressed, it became evident that the service struggled with strong accents and fast speech, leading to inconsistent performance. Its **Word Error Rate (WER)**, which ranged between 16.51% and 20.63%, further confirmed these limitations [3]. These issues reduced its suitability for handling diverse user interactions, prompting the search for a more robust alternative.

Amazon Transcribe showed some improvement in terms of accuracy, achieving a **WER** of approximately 12.9% [3]. While this marked a step up from Google's API, it still fell short of the system's needs, particularly in scenarios requiring high precision and adaptability. Similarly, **Microsoft Azure Speech-to-Text** delivered **WER** results between 15% and 20%, making it comparable to Google in terms of accuracy. Additionally, Azure's service required manual configuration for multilingual input, limiting its flexibility and adding complexity to the integration process.

Another explored option was **Rasa's ASR capabilities**. Unlike the other solutions, Rasa does not offer a native speech recognition system but instead relies on external ASR services such as **Deepgram** [4]. While this approach provides some flexibility, it also introduces dependencies that limit direct control over the ASR process. Given that the system required a more self-sufficient and high-performance solution, Rasa's ASR integration was ultimately ruled out.

After evaluating all available options, **OpenAI's Whisper** model, accessed through **Azure OpenAI**'s API, emerged as the most effective choice. It consistently delivered the lowest **WER**, with a median of 8.06%, significantly outperforming all other tested solutions [5]. Additionally, independent research [6] confirmed Whisper's reliability across noisy environments, diverse accents, and multilingual speech, making it particularly well-suited for real-world applications. Unlike Google's API, which required explicit language configuration, Whisper's ability to dynamically adapt to multiple languages further reinforced its advantage over competing ASR models.

Another major decision regarding the implementation of the node was whether to run the ASR model **locally** on the external laptop or process it in the **cloud**. Initially, executing the model locally seemed like a viable option, as it would reduce dependence on external services and eliminate potential network-related delays. However, high-performance ASR models, such as Whisper, require significant computational resources, which exceeded the capabilities of the available hardware.

Given these limitations, a cloud-based approach was ulti-

mately chosen to ensure scalability and efficiency. Offloading processing to Azure OpenAI not only provided access to more powerful computational resources but also allowed the system to benefit from continuous model updates and improvements. This approach ensures that the ASR functionality remains optimized and adaptable, without being restricted by local hardware constraints.

With the choice of Whisper and cloud-based processing established, the next step was implementing the ASR node to efficiently handle audio input and interact with the **Azure OpenAI** for transcription. The ASR node processes **raw audio data** received through the ROS service request. The audio is converted into a byte array and saved as a temporary **WAV** file, formatted according to Whisper's specifications. This file is sent to **Azure OpenAI** via a **RESTful request**, where the Whisper model processes it and returns the transcribed text in **JSON** format. The node extracts the transcription from the API response and provides it to the requesting component.

Since the ASR node operates within a real-time system, it was also essential to implement mechanisms that ensure robustness and stability, even in cases of unexpected errors. To ensure reliability, the node includes error-handling mechanisms that manage invalid audio inputs and API failures, preventing system disruptions.

2.3 Additional Core Nodes

In addition to the structured modules, the system includes a set of nodes that manage aspects of Pepper's interactions, dialogue processing, and state transitions. These nodes handle core functionalities independently while ensuring seamless integration with other components, allowing the robot to operate efficiently and adapt dynamically to user interactions.

The following sections describe the **Animated Speech Node**, responsible for synchronising spoken output with contextual gestures; the **WakeUp Rest Node**, which manages Pepper's transitions between active and resting states; the **Dialogue Node**, which handles natural language processing and response generation; and the **Behaviour Manager Node**, which coordinates user interactions by processing sensory input and generating responses.

2.3.1 Animated Speech Node

The **Animated Speech Node** enables Pepper to produce spoken responses while synchronising them with appropriate animations, enhancing the quality of human-robot interactions. By integrating **text-to-speech (TTS)** with contextual gestures, it ensures that verbal communication is naturally accompanied by expressive movements, making interactions more dynamic and engaging. The node leverages **NAOqi's ALAnimatedSpeech** API, which processes the text input for speech synthesis and determines suitable animations to align with the content, reinforcing Pepper's ability to convey information in a more intuitive and human-like manner.

The node is implemented as a **ROS service** named `animated_speech_service`, which receives requests containing a speech string and processes them asynchronously. Upon receiving a request, it forwards the input to the ALAnimatedSpeech API, which executes both the speech synthesis and the corresponding animations. This design allows the system to delegate the complexity of speech-gesture coordination to a dedicated service, ensuring seamless and natural synchronisation without additional processing overhead.

To maintain responsiveness, the node is structured to handle multiple requests efficiently, preventing blocking operations that could disrupt the flow of interactions. This is particularly relevant in scenarios requiring continuous engagement, where rapid execution of consecutive speech outputs is necessary. Additionally, error-handling mechanisms are incorporated to monitor speech synthesis and animation execution, logging potential failures and ensuring the system can recover from disruptions without affecting the overall performance.

2.3.2 WakeUp Rest Node

The **WakeUp Rest Node** manages Pepper's transitions between active and resting states, ensuring smooth posture adjustments and optimised energy consumption. It provides two dedicated **ROS services**: `wakeup_service`, which powers on the robot's motors and adjusts its posture for interaction, and

`rest_service`, which places the robot in a stable resting position, reducing power usage while keeping it ready for reactivation. These state transitions are essential for preserving Pepper's mechanical integrity and adapting its operational status based on activity requirements.

To execute these commands, the node interfaces with **NAOqi's ALMotion** and **ALRobotPosture** APIs, ensuring that movements are performed safely and efficiently. Additionally, this implementation integrates the **ALBasicAwareness** API, allowing Pepper to detect and respond to its surroundings. When the wake-up command is issued, this service is activated, enabling the robot to track faces, detect motion, and dynamically adjust its focus to human presence, enhancing engagement from the moment it becomes operational. Conversely, when transitioning to the resting state, the awareness service is deactivated to conserve energy and prevent unnecessary processing.

To improve reliability, the node incorporates a refined error-handling mechanism that monitors potential failures during state transitions. If an issue arises, motion and posture services are reset, and the operation is retried, ensuring that temporary disruptions do not affect the robot's ability to switch states.

2.3.3 Dialogue Node

The **Dialogue Node** is responsible for managing text-based conversational interactions, enabling Pepper to generate natural language responses based on user input. It is implemented as a **ROS service**, named `dialogue_server`, which processes incoming text queries and returns AI-generated responses. This node allows Pepper to engage in meaningful exchanges without relying on predefined dialogue flows.

Upon receiving a service request, the node forwards the provided text to a chatbot implemented using **OpenAI's Assistant API**, accessed through **Azure OpenAI**. The assistant processes the input and generates a response, which is then returned in textual form to the requesting component. The node is designed to function asynchronously, minimising response latency and maintaining real-time conversational fluidity.

To improve reliability, error-handling mechanisms have been implemented to manage potential failures in communication with the cloud-based service. In cases where a response is delayed or an issue arises, the system ensures graceful recovery without affecting other ongoing operations.

2.3.4 Behaviour Manager Node

The **Behaviour Manager Node** acts as the central controller for Pepper's interactive capabilities, coordinating multiple subsystems to ensure fluid and context-aware interactions. Its primary function is to manage the robot's engagement with users by integrating sensory input, processing speech, generating responses, and triggering appropriate actions in a structured manner.

A fundamental aspect of its design is the ability to dynamically adapt Pepper's behaviour based on user presence. By subscribing to the `/face_detection` topic, the node continuously monitors for detected faces, enabling it to regulate interaction flow accordingly. When a user is present, it initiates the voice detection process to acquire speech input, which is subsequently processed through automatic speech recognition to produce a textual representation. The generated text is then passed to the dialogue system for response generation before being forwarded to the animated speech service, which handles both text-to-speech synthesis and corresponding gestural output. If no user is detected for a predefined period, the node resets the interaction state to prevent unnecessary resource consumption and ensure system efficiency.

To maintain low-latency performance, the node employs persistent ROS services for high-frequency operations such as speech recognition, voice detection, and dialogue processing. This design choice mitigates the overhead associated with repeated initialisation and enhances real-time responsiveness. Core execution is managed within a structured event-driven loop, which continuously evaluates system state, orchestrates service invocations, and ensures synchronised access to shared resources to prevent concurrency-related issues.

3 Chatbot

The **chatbot** integrated within the **Dialogue Node** was developed to equip Pepper with advanced conversational capabilities, allowing it to handle both open-ended dialogues and structured queries relevant to its environment. The system was specifically designed to process requests related to the shopping mall, enabling Pepper to determine the number of people present and identify individuals based on specific visual attributes. Additionally, it needed to provide information about an competition, including details on the number of participating groups, their composition, rankings, and scores. These requirements made it essential to implement a solution capable of understanding both general language queries and structured, task-oriented questions.

During the evaluation phase, different approaches were considered to identify the most effective way to implement the chatbot. An initial assessment focused on **Rasa**, an open-source **NLU framework** that offered a high degree of flexibility in structuring NLU pipelines. The ability to define intent recognition, entity extraction, and dialogue management explicitly allowed for precise control over the chatbot's behaviour. However, this same level of customisation introduced significant challenges, particularly in terms of data requirements and adaptability.

The model relied on a **supervised learning approach**, meaning that for each **intent**, it required an extensive set of manually defined training examples to ensure accurate classification. This dependency on predefined data made generalisation difficult, as even small variations in phrasing could lead to misclassification if those specific formulations had not been explicitly included in the dataset. The same issue extended to **entity** recognition, where the system required an exhaustive list of possible values and synonym mappings to function effectively. Despite efforts to provide a diverse and structured dataset, it frequently struggled to correctly classify entities, particularly when they appeared in less common contexts or deviated slightly from the provided examples. This limitation became even more evident in cases where the intent remained the same but the entity value changed. Even with an **identical sentence structure**, the model sometimes failed to correctly associate the input with the intended action, requiring additional data augmentation to improve classification accuracy. The necessity of continuously expanding the dataset to account for such inconsistencies significantly increased the effort required to maintain reliable performance.

Beyond these limitations in understanding natural language, Rasa also required explicit state tracking to manage conversation memory. Unlike end-to-end neural network models, which handle context retention implicitly, Rasa demanded a structured dialogue flow where previous exchanges had to be manually tracked and incorporated into decision-making processes. This made multi-turn conversations harder to manage, as ensuring continuity required additional engineering effort. The rule-based nature of the system further constrained its ability to handle dynamic interactions, often resulting in rigid conversational flows that struggled with unexpected queries. The need for extensive manual configuration ultimately made Rasa less suitable for the project's requirements.

Given these constraints, **OpenAI's Assistant API** emerged as the more effective alternative. Unlike Rasa, which relies on predefined patterns and extensive manual configuration, the Assistant API is built on a **pre-trained transformer-based language model (gpt-4o)** capable of dynamically interpreting user input. This approach eliminated the need for manually defining intents and responses, significantly reducing development overhead while ensuring higher adaptability. An appreciated advantage of this solution was its built-in ability to retain conversational memory, allowing Pepper to handle multi-turn interactions naturally without requiring explicit state management. This ensured that user exchanges remained coherent and contextually relevant across multiple requests.

The Assistant API was also chosen over **OpenAI's Completion API** due to its ability to process structured information in addition to general conversation. To ensure that Pepper could provide accurate responses to queries related to the shopping mall and the competition, the chatbot was equipped with **knowledge files** containing structured project-specific data. These files allowed the system to retrieve real-time statistics,

ensuring that responses were always up to date. Additionally, the chatbot was integrated with **OpenAI's Code Interpreter** tool, which enabled it to process structured numerical data and perform analytical reasoning. This made it possible for Pepper to answer queries requiring computations, such as retrieving competition rankings based on scoring metrics or analysing real-time occupancy levels in the shopping mall.

To align the chatbot with the system's specific requirements, it was configured with **instructional guidelines** that defined its role, behaviour, and response style. These guidelines ensured that the assistant would consistently refer to itself as Pepper, maintain a natural and engaging communication style, and generate responses that prioritised clarity and relevance. While configuring an AI assistant with such guidelines is a standard feature of OpenAI's Assistant API, it was essential in ensuring that Pepper's responses matched the expectations set for the project.

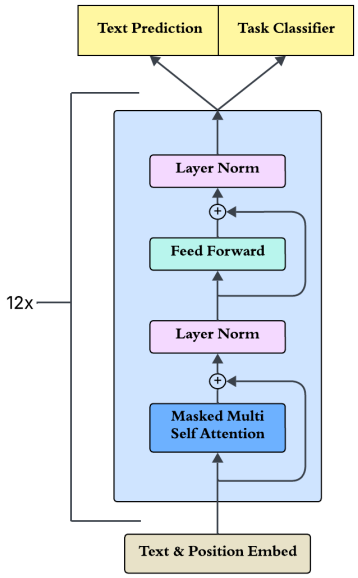

Figure 3: GPT Transformer Architecture

3.1 NLU in GPT-models

Unlike traditional dialogue systems that rely on predefined intent classification and rule-based dialogue management, the chatbot integrated into the **Dialogue Node** utilises a **transformer-based GPT model** to process user input in a more flexible and adaptive manner. Instead of following a rigid **Natural Language Understanding (NLU) pipeline**, where processing is divided into discrete stages such as intent recognition and entity extraction, the GPT-based system interprets input thoroughly.

At the core of GPT's ability to process language is the **Transformer architecture**, which enables the model to analyse text as a whole rather than processing words sequentially. The first step in this process is **tokenisation**, where the input text is broken down into smaller units, called **tokens**, using **Byte-Pair Encoding (BPE)**. Each token is then mapped to an **embedding**, a numerical representation that captures its meaning in relation to other words. Since transformers do not process text in a strictly sequential manner, positional encodings are added to ensure that the model retains information about word order. These enriched representations are then passed through multiple layers of **self-attention** and feedforward networks, which refine the contextual understanding of the text.

The **self-attention mechanism** is a feature that allows GPT to evaluate how words relate to one another across the entire input. By assigning different levels of importance to each token depending on its relevance to the overall meaning, the model can capture complex dependencies between words. Additionally, multi-headed attention enables GPT to focus on multiple aspects of a sentence simultaneously, such as syntactic structure, semantic meaning, and named entities. Unlike conventional NLU architectures, which require predefined labels for intents and entities, GPT processes language in a more fluid manner, adjusting dynamically to the user's input without relying on manually defined categories.

As the conversation progresses, the model maintains **contextual memory**, tracking the history of interactions to ensure that responses remain relevant and coherent. Unlike rule-based dialogue systems, which require explicit state management, GPT naturally incorporates previous exchanges into its reasoning, allowing for more dynamic and context-aware interactions.

Response generation follows an **autoregressive approach**, meaning the model predicts each word step by step

based on the preceding tokens. A feedforward network computes logits for the next possible words, and a softmax function determines the most likely choice at each step. This iterative process continues until a full response is generated, ensuring grammatical correctness and logical consistency.

4 Test

This section reports on the qualitative and quantitative tests conducted to evaluate the system's functionality and performance. The qualitative assessment aimed to demonstrate the system's behaviour in real-world scenarios, providing examples in the form of video recordings. These examples illustrate the interaction between the robot and users, showcasing the functionalities of the implemented ROS nodes. The quantitative evaluation focused on collecting and analysing relevant performance data to assess the system's effectiveness and reliability. The results from these tests offer valuable insights into the system's capabilities and potential areas for improvement.

4.1 Video Analytics Test

Qualitative tests for the **Video Analytics Module** were conducted to verify the correct operation of the component under real-world conditions and ensure its adaptability across different hardware configurations. The module was tested using two setups, managed via dedicated **ROS launch files**:

- With **Pepper's onboard camera** (`face_detector.xml`), leveraging its integrated vision system for video capture.
- With an **external camera** (`face_detector_nopepper.xml`), ensuring the system's flexibility beyond Pepper's built-in hardware.

These configurations allowed seamless switching between setups while maintaining the same detection and visualisation processes. The tests aimed to evaluate the robustness and stability of the module when processing video streams in different environments, considering variations in lighting conditions, user positioning, and camera angles.

An important aspect of these tests was the real-time verification of the face detection process, made possible through the **Visualization Node**. By overlaying bounding boxes and confidence scores on the video feed, users could immediately assess whether detected faces were correctly identified and tracked. This facilitated the identification of potential limitations, such as false positives, missed detections, or performance inconsistencies between the two camera configurations.

While the qualitative evaluation provided insights into the module's real-world performance, a quantitative analysis was conducted to objectively assess its accuracy and efficiency. This analysis leveraged the **FDDB (Face Detection Data Set and Benchmark)**, which provided a structured framework for assessing detection accuracy and mitigating real-world variability.

The evaluation was based on standard classification metrics, including **accuracy**, **precision**, **recall**, and **F1-score**, computed globally across the dataset. Additionally, a **Receiver Operating Characteristic (ROC)** curve was generated to illustrate the relationship between the true positive rate and the false positive rate across different confidence thresholds, with the **Area Under the Curve (AUC)** serving as an aggregate measure of the model's discriminative capability. The results demonstrated an accuracy of 72.2%, a precision of 86.6%, a recall of 81.3%, and an F1-score of 83.9%, indicating a well-balanced performance in terms of false positives and false negatives. The AUC value of 0.81 further confirmed the model's ability to distinguish faces from background elements. These results confirm that the module achieves a good balance between precision and recall, making it reliable for real-time face detection across diverse scenarios.

Performance was also evaluated in terms of computational efficiency, with an **average inference time** of 0.0085 seconds per image, ensuring that the system meets real-time processing requirements. The structured dataset evaluation provided a clear benchmark for detection performance, complementing the qualitative tests conducted in real-world conditions.

4.2 Speech To Text Test

The qualitative evaluation of the **Speech-to-Text (S2T) Module** was conducted to assess its ability to accurately transcribe spoken input in different operational contexts. The module was tested in two distinct configurations, each launched using a dedicated **ROS launch file**:

- **Integrated with Pepper** (`s2t.xml`): The module operated as part of the full system, interacting with other components. This setup allowed evaluation of its performance within a real-world scenario where speech recognition influenced the robot's behavior.
- **Standalone operation** (`s2t_nopepper.xml`): The module was tested in isolation, independent of Pepper, to analyze its core transcription performance without external interactions.

In both setups, the **Voice Detection Node** captured audio from an external microphone, while the ASR Node performed transcription using **OpenAI's Whisper model**, deployed via **Azure OpenAI**. Testing conditions included variations in speech clarity, accents, and articulation styles, ensuring exposure to a diverse range of inputs. The transcriptions were monitored in real-time through the module's logs, allowing for immediate evaluation of accuracy and responsiveness.

To complement these qualitative observations, we refer to the quantitative evaluation provided in OpenAI's official report on **Whisper** [5], which analyzes the model's accuracy and its limitations under various speech conditions. The report highlights several aspects of Whisper's performance:

- **Clearly articulated speech** is generally transcribed with high accuracy.
- **Accent variations and informal language structures** increase recognition errors, with notable misinterpretations of colloquial phrases.
- **Proper names and regional terms** are frequently replaced with phonetically similar alternatives, a known limitation of Whisper due to its reliance on probabilistic word inference.

The **Word Error Rate (WER) metric** calculated quantifies these effects, showing higher error rates in cases involving rapid speech, accent-heavy pronunciation, or informal sentence structures. While the model remains effective for general speech recognition, these findings suggest that additional fine-tuning or adaptation would be beneficial for applications involving diverse linguistic inputs.

4.3 Animated Speech Test

The qualitative evaluation of the **Animated Speech Node** aimed to verify its ability to synchronize spoken output with corresponding animations, ensuring natural and fluid interactions. The test was conducted using a dedicated script that sent structured speech requests to the `animated_speech_service`, with Pepper actively connected to execute both speech and gestures.

The script contained predefined test messages designed to assess synchronization between speech and movement. For example, one such message, "Hello! How are you today?", was structured to trigger both a verbal greeting and a waving gesture. By analyzing the execution, the test evaluated whether the spoken output matched the expected text, whether the animation was performed at the correct moment, and whether the overall interaction appeared smooth and natural.

4.4 Chatbot Test

The evaluation of the chatbot integrated within the **Dialogue Node** was conducted through both qualitative and quantitative testing to assess its ability to handle queries effectively, maintain conversational coherence, and adhere to the predefined interaction guidelines. The chatbot was tested in two distinct configurations:

- **Full system integration**: Where interactions occurred through Pepper, incorporating the complete pipeline of speech recognition, transcription, and response generation.

- **Direct terminal interaction**: Where the chatbot was accessed via text input, isolating its conversational capabilities from the additional speech processing components.

The qualitative assessment aimed to evaluate both the accuracy of the chatbot's responses and its adherence to the defined behavioural guidelines. A series of structured conversations were conducted, covering a diverse range of queries relevant to the system's domain. The questions tested various aspects of the chatbot's knowledge, including:

- Counting individuals in the shopping mall based on specific attributes.
- Locating lost individuals within the environment.
- Providing details on the competition, such as identifying the winner, listing group members, checking whether a specific individual participated, and retrieving rankings and scores.

The test sessions were designed to simulate realistic user interactions, with questions asked in random sequences to challenge the chatbot's ability to handle context shifts and topic changes. This evaluation was essential to determine whether the chatbot could maintain relevant information across multi-turn conversations.

In addition to factual correctness, the chatbot's language style and compliance with predefined behavioural constraints were assessed. It was verified that the chatbot consistently identified itself as "Pepper" maintained a polite and cooperative tone, and provided concise yet informative responses.

The qualitative evaluation uncovered that while the chatbot correctly answered the vast majority of queries, it exhibited a tendency to **hallucinate** information, occasionally generating incorrect or entirely fabricated responses. This phenomenon is a well-documented limitation of large language models, as also reported in OpenAI's GPT-4o technical report [7]. The issue was further amplified by the fact that the Assistant API used for chatbot deployment remains in beta stage, implying that its behaviour is still subject to refinement.

To complement the qualitative assessment, a quantitative evaluation was conducted to systematically measure the chatbot's performance. Given the concern regarding its tendency to hallucinate, the **DeepEval** library was utilised to compute objective evaluation metrics based on a structured dataset.

Three metrics were selected for assessment:

- **Factual Accuracy**: Measures whether the chatbot's responses contain **correct and verifiable information**, by comparing them against a reference ground truth.
- **Response Relevance**: Evaluates how closely the chatbot's answers match the intent of the immediate user query, ensuring that the response remains on-topic. This metric employs **embedding-based similarity** techniques to assess semantic alignment between the question and the generated answer.
- **Conversational Relevance**: Extends the concept of response relevance to **multi-turn interactions**, verifying that the chatbot maintains **contextual continuity** and does not produce responses that deviate from the established dialogue flow.

The evaluation dataset consisted of 50 structured queries, reflecting the range of information accessible to the chatbot. The results were as follows:

- **Factual Accuracy: 0.92** - Indicating that, despite occasional hallucinations, the chatbot provides correct information in the majority of cases, mitigating initial concerns regarding its reliability.
- **Response Relevance: 0.53** - This lower score is attributed to the chatbot's tendency to provide additional contextual information beyond the explicit query. While this behaviour aligns with its intended role as a helpful assistant, it was detected as off-topic by the evaluation metric.
- **Conversational Relevance: 0.62** - The chatbot demonstrated moderate capability in maintaining conversation flow, benefiting from its ability to retain contextual

knowledge, though still affected by the extraneous information observed in single-turn responses.

4.5 Integration Test

The purpose of the **Integration Test** was to verify the end-to-end functionality of the complete system, ensuring that individual modules operated in coordination and that the overall execution flow conformed to the intended interaction model. This evaluation focused on validating the communication among components, the correctness of event sequencing, and the system's responsiveness to real-time user inputs.

The test scenario simulated a full interaction session with a user. The sequence was initiated when a person approached Pepper and greeted it. At this stage, the system relied on the **Video Analytics Module** to detect the presence of a face within Pepper's field of view. Upon successful detection, Pepper was expected to respond verbally to the greeting, initiating the interaction.

The user then engaged in a dialogue with Pepper by asking various questions. These included inquiries about the shopping mall, such as the number of people present with specific attributes or the whereabouts of a missing individual. Other queries focused on the **Artificial Vision Contest**, covering topics such as the final rankings, podium positions, group membership, participant verification, and individual scores.

The **Speech-to-Text Module** transcribed the user's spoken input, which was subsequently processed by the **Dialogue Node** and the integrated chatbot to generate appropriate responses. Pepper delivered the responses using the **Animated Speech Node**, which was responsible for synchronising verbal output with contextual gestures—such as a wave when greeting or a nod during informative answers.

The session concluded when the user said goodbye. Pepper, upon recognizing the farewell through its speech and vision subsystems, responded accordingly with a verbal and animated goodbye.

This test validated the system's ability to handle natural, multi-turn interactions involving visual recognition, real-time speech transcription, contextual response generation, and synchronised animated output. It demonstrated that the modules—when integrated—effectively maintained consistency, timing, and coherence throughout the full interaction flow, confirming the system's readiness for deployment in realistic environments.

References

[1] Ahmed Yamout, Ahmed Abdelmawgood, Ebraam Sadick, and Mohamed Naguib. Comparative evaluation of face detection algorithms. pages 64–71, 2020.

[2] Abhishek Tiwari, Suhail Manzoor, Jiya Sehgal, and Ashutosh Mishra. A comprehensive review of face detection technologies. 1:1–6, 2024.

[3] Gladia. Openai whisper vs google speech-to-text vs amazon transcribe, 2024.

[4] Rasa Technologies. Speech integrations in rasa pro, 2024.

[5] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. *OpenAI Technical Report*, 2022.

[6] Calbert Graham and Nathan Roll. Evaluating openai's whisper asr: Performance analysis across diverse accents and speaker traits. February 2024.

[7] OpenAI. Gpt-4o system card, 2024.