Diploma Project

DEVELOPMENT OF AN AI AVATAR CHATBOT FOR CONTROLLED DO-MAIN INTERACTIONS

Maria Chrysanthou

University of Cyprus
Department of Computer Science

Department of Computer Science

May 2026

UNIVERSITY OF CYPRUS

Faculty of Pure and Applied Sciences

Department of Computer Science

DEVELOPMENT OF AN AI AVATAR CHATBOT FOR CONTROLLED DOMAIN INTERACTIONS

Maria Chrysanthou

Supervisors

Dr. Yiorgos Chrysanthou

Dr. Kleanthis Neokleous

The Thesis was submitted in partial fulfillment of the requirements for obtaining the Computer Science degree of the Department of Computer Science of the University of Cyprus

May 2026

I declare that this dissertation and any accompanying software are my own original work, conducted in full compliance with the University of Cyprus Code of Conduct for Research (https://www.ucy.ac.cy/legislation/kodikas-deontologias-kai-politikes/), and with full accountability on my part for the accuracy, integrity, and validity of all content.

Acknowledgements

I would first like to express my sincere gratitude to Dr. Kleanthis Neokleous for his guidance, support, and valuable feedback throughout this thesis. His advice and continuous assistance during the development and evaluation of the project were greatly appreciated and helped shape the final outcome of this work.

I would also like to thank Dr. Yiorgos Chrysanthou and the Department of Computer Science at the University of Cyprus for the opportunity to carry out this thesis and for the academic knowledge and support provided throughout my studies.

In addition, I would like to thank everyone who participated in the evaluation process and provided valuable feedback during the testing of the system.

Finally, I would especially like to thank my family and friends for their patience, encouragement, and support throughout this journey.

ABSTRACT

Conversational AI systems are becoming increasingly common in everyday environments, however many existing assistants still rely mainly on text or heavily scripted dialogue. This thesis presents Lana, a voice-driven embodied conversational assistant designed for controlled-domain interactions, with a hospital information environment used as the primary use case. The system combines a Unity frontend with a FastAPI backend, allowing users to ask questions through speech or text and receive answers through both audio playback and an animated 3D avatar. The implemented architecture follows a retrieval-first approach built around a structured hospital knowledge base from official public information. It uses a hybrid retrieval pipeline which combines semantic embeddings and BM25 lexical scoring in order to provide grounded and consistent responses, while large language models are used only for constrained tasks such as query routing, follow-up interpretation, and refining responses. Speech interaction is supported through speech-to-text and text-to-speech processing, while avatar lip synchronization and basic animations are used to make the interaction more natural. The prototype was evaluated with 18 participants using hospital-based scenarios and a user questionnaire. Participants were able to complete most tasks successfully and responded positively to how easy the system was to use, its responsiveness, and the avatar interaction. At the same time, the evaluation revealed some practical limitations, particularly in the way it handled certain follow-up questions, the speech recognition when under less ideal conditions, and the limited scope of the current knowledge base. These findings helped identify areas that could be improved in future versions of the system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

A&E	Accident and Emergency
AI	Artificial Intelligence
API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
BM25	Best Matching 25
ECA	Embodied Conversational Agent
FAQ	Frequently Asked Question
HTTP	Hypertext Transfer Protocol
ICU	Intensive Care Unit
IDF	Inverse Document Frequency
JSON	JavaScript Object Notation
LLM	Large Language Model
NLP	Natural Language Processing
OVR	Oculus Virtual Reality
PCM	Pulse-Code Modulation
RAG	Retrieval-Augmented Generation
REST	Representational State Transfer
SHSO	State Health Services Organisation
STT	Speech-to-Text
TTS	Text-to-Speech
UI	User Interface
URL	Uniform Resource Locator
VAD	Voice Activity Detection
WebRTC	Web Real-Time Communication

1 Introduction

1.1 Motivation

A good conversational system should feel less like "using software" and more like speaking to someone who can actually help. In the last few years, conversational AI has gone from a novelty to something people use daily. For instance voice assistants handle quick tasks, and chatbots answer questions on websites and services [1]. These systems are becoming more common, which brings more expectations from users such as fast or natural interactions, closer to how humans normally communicate.

At the same time many conversational systems still rely mainly on text. This can easily become a problem for users who are multitasking, or have difficulty reading and typing, since text-based interaction can be slow and it demands visual attention. It also tends to feel less "present", even when the content is correct, since an interface that is entirely text-based can feel impersonal, which is not ideal in settings where people expect an experience similar to asking a real assistant.

This is where embodied, multimodal conversational agents become interesting. When conversation is accompanied by a visible avatar, users naturally expect more human-like behavior such as gaze, facial expressions, and gesture, which normally support speech in face-to-face communication [2]. On top of that, spoken interaction is highly sensitive to timing. Delays or even small pauses can make the system feel unresponsive and affect user performance, which makes low-latency interaction a practical requirement for a natural, responsive experience [3].

Finally, the motivation for this work is not limited to one specific use case. Public-facing assistants appear in many contexts like hospital receptions [1], museum guides [3], university helpdesks, and information kiosks, where the core need is similar. Answer common questions clearly and consistently, with a natural interaction style. Instead of building a separate system from scratch for each domain, it is more practical to develop an adaptive framework that can be used in multiple domain by changing its controlled knowledge source, while keeping the same interaction pipeline.

1.2 Problem Statement

While chatbots and voice assistants are widely used today, many still don't feel like a "real" interaction. In practice, most systems are either good at answering questions but remain disembodied (text-only or voice-only), or they use an avatar, but conversation is heavily scripted and limited. This results in a gap between simply receiving an answer and having a natural

exchange with an assistant that feels present [4].

This can be even more challenging when speech and embodiment are combined together. A human like avatar creates expectations for human-like behavior, including basic non-verbal cues such as gaze, facial signals, and gestures which also support speech in face-to-face communication [2]. At the same time, these systems can negatively affect the user experience if the timing is not right, which highlights even more the need for natural and fast responses [3].

The problem addressed in this thesis is how to design and implement a stable, low-latency, voice driven embodied conversational agent, while also remaining domain adaptive through a controlled environment (e.g. hospital, museum, university helpdesk etc.)

1.3 Study Objectives

To address the problem described in the previous section, this thesis sets a number of concrete objectives that guide both the system development and the evaluation.

The specific objectives of the study are:

- **Develop an end-to-end interaction loop:** Support the full conversational cycle, from speech input to response generation and speech output, including synchronized avatar speaking behaviour.
- **Enable real-time interaction:** Integrate Speech-to-Text for user input and Text-to-Speech for system output, with emphasis on responsiveness and stability.
- **Design a modular architecture:** Structure the system in a way that separates the main components (frontend, backend, speech modules, reasoning modules) to support maintainability and future expansion.
- **Integrate a domain-adaptive system:** Allow the system to be configured for multiple different settings (e.g. hospital, museum, university helpdesk) by using a controlled knowledge source, without the need for an entirely new pipeline.
- **Provide basic embodiment features:** Implement embodiment elements such as talking/idle animations, lip-sync, and simple presence cues, aiming for functional and realistic interactions rather than visual complexity.
- **Evaluate functionality and practical limitations:** Test the system through structured interaction scenarios and technical evaluation (e.g. response flow and timing constraints), and document limitations and possible improvements.

1.4 Overview of the Thesis Structure

This thesis is organized into 6 chapters, which are designed to clearly present the background, the system design, the implementation, and finally the evaluation of the work.

Chapter 2 covers the background and related work. This provides the reader with an overview of what conversational agents, speech based interaction and embodied/multi-modal systems are, and explains how this project fits within that area. Chapter 3 presents the overall architecture and design of the system. It explains the organization of the main components, how the frontend and backend communicate, and explains certain design decisions.

Chapter 4 focuses on implementation. It goes into the practical side of the project and describes how the different parts were built and connected to support the full interaction loop. Chapter 5 is using scenarios and technical analysis to perform an accurate evaluation of the system, and it also discusses limitations and constraints.

Finally, Chapter 6 provides the conclusion and future improvements. Additional technical information (such as usage instructions and code) is included in the Appendices.

2 Background and Related Work

2.1 Conversational Agents

Conversational agents are systems that talk to people using natural language, either through text or speech [5]. Their goal is to support natural interaction by providing information, assisting with tasks, or helping users make decisions. Over time, these systems evolved from simple rule-based chatbots into more advanced architectures that use machine learning and natural language processing [5, 6].

Earlier versions of conversational systems were mainly rule based, meaning they followed set patterns and had scripted responses [5]. These systems had limited flexibility as they could only respond to inputs that matched specific rules. Retrieval-based systems were introduced in order to address this limitation, where the system selects the most appropriate response from a predefined dataset based on similarity measures. While they are more robust than rule-based approaches, these systems are still constrained by the coverage of their knowledge base [7].

Generative conversational agents using large language models (LLMs) have gained popularity in recent years [6]. They are able to generate responses dynamically, which makes interactions more flexible and natural. However fully generative systems have certain drawbacks such as lack of control, inaccuracies and even hallucinations, which can be a problem for applications that value safety, or are domain-specific [8].

In this project, a hybrid approach is used, which combines retrieval-based methods with controlled use of language models. The system mainly relies on a structured knowledge base and retrieval pipeline to keep responses accurate and consistent, while LLMs are used selectively in fallback cases or to support the conversation flow [9].

2.2 Large Language Models

Large Language Models (LLMs) are deep learning models trained on large amounts of text data to understand and generate natural language. Nowadays, most LLMs are based on the Transformer architecture which uses self-attention methods to analyze the relationships between words in a sequence [10].

The behaviour of LLMs is heavily influenced by prompt engineering. Through careful structuring of prompts, it is possible to influence the behaviour of the model and constraint its output [11]. In this system, prompts are used to guide the model toward domain-specific behaviour, and generate outputs that are specifically catered on hospital information services.

However beyond the general limitations mentioned above, a critical problem that occurs while working with LLMs is controlling them within the context of a certain domain. Since these models are trained on diverse data sets, there is no guarantee that the responses will always stay focused on the intended topic. Therefore it is necessary to ensure the results are relevant and appropriate for the given application [9].

These challenges are clearly reflected in the design of this system, and affect the way it works. Relying on external LLM APIs introduces potential issues related to performance and availability. This became evident during testing, so fallback mechanisms were added to maintain system reliability. For this reason, the system prioritizes retrieval-based responses using its knowledge base, and instead uses LLM only when it is necessary.

2.3 Speech Interaction

Speech interaction is a critical component of modern conversational systems, that enables a more natural user experience [4].

2.3.1 Speech-to-Text (STT)

Speech-to-text systems convert spoken language into text. In this thesis, STT is implemented using a neural model that is capable of real-time transcription [12]. Audio input is captured and processed from the user's microphone, and then passed to the backend for transcription. The implementation includes mechanisms that handle real-world constraints such as silence detection and recording limits. For instance, silence detection automatically pauses recording when silence is detected after speech, which improves the response time and avoids unnecessary processing.

2.3.2 Text-to-Speech (TTS)

Text-to-Speech systems convert textual responses into spoken output. This project uses a neural TTS provider capable of generating high-quality speech with configurable voice parameters such as pitch, rate, and volume [13]. The generated audio is transmitted as PCM (Pulse-code modulation) data and played in the Unity frontend. The system also verifies that audio is not silent or corrupted before playback, which helps prevent playback errors.

2.3.3 Real-time Constraints

A key challenge in speech-based systems is maintaining low latency. Delays in STT, backend processing, or TTS generation can make conversations feel slow or unresponsive [3]. The sys-

tem reduces these delays through asynchronous processing, early stopping in audio recordings, as well as limiting unecessary network calls, in order to reduce the response time of the LLM.

2.4 Embodied Conversational Agents

Embodied Conversational Agents (ECAs) represent the development of traditional chatbots, with added elements of a physical presence, in the form of a virtual avatar [2, 5]. This adds visual and behavioural elements to the interaction, such as facial expressions, gestures, and lip synchronization [5].

In this project, the frontend is implemented in Unity and includes an animated avatar that is able to respond to user interaction. The system uses lip sync techniques in order to align mouth movements with speech. In particular, the application makes use of viseme-based animation, which maps phoneme data into facial blend shapes.

Additional features include lip-sync processing by using audio input to detect visemes and a head nodding feature triggered during interactions. Although limited, there is also gesture and talking animations based on responses, in order to avoid repetitive patterns.

Together, these features help the perception of the system feel like a "present" conversational agent, rather than a simple scripted bot [5]. However, embodiment can also bring challenges, such as users expecting human-like timing and behavior, which means a mismatch in speech and animation could negatively impact the conversation [3].

2.5 Related Work

Over the years, different types of conversational systems have been developed, each focusing on a specific part of human interaction with technology, such as text, speech, or visuals.

Text-based conversational systems have been significantly improved due to the development of large language models. With the introduction of transformer architectures by Vaswani et al. [10], it became possible to design conversational systems capable of producing responses based on contextual awareness. A more recent version of conversational systems is Retrieval-Augmented Generation (RAG) by Lewis et al. [9]. The idea behind RAG is to enable the integration of data from external sources into a language model. These conversational systems can be viewed as disembodied since they solely use texts in order to generate a response.

Voice assistants represent another category of conversational systems. As the name suggests, these conversational agents are designed to let users interact with them through speech. However, studies such as Völkel et al. [4], show that voice assistants today are predominantly task-

oriented and lack support for general or domain-specific conversation. In addition, these conversational agents are disembodied by nature.

Embodied conversational agents (ECAs) have been studied as a way to introduce visual presence into human-computer interaction. According to Cassell [2] and Yang et al. [5], these agents use speech combined with non-verbal behaviours including gaze, facial expressions, and gestures, to increase user engagement. Nevertheless, many of these systems utilize only scripted or constrained dialogues, which reduces their flexibility in real-world applications.

In domain-specific areas, conversational agents have been applied in fields such as healthcare. Based on the findings of Laranjo et al. [1], healthcare chatbots are typically designed for specific tasks such as symptom checking or patient guidance, which lack in terms of safety and reliability.

2.6 Research Gap

From the discussion above, it is clear that current conversational systems focus on specific conversational aspects without providing a complete solution [1, 4–6, 9]. While text-based systems offer strong linguistic abilities, they lack embodiment, and voice systems can interact through speech, but are usually limited to executing tasks. On the other hand, embodied agents improve user engagement but require scripted or restricted dialogue. Moreover, generative conversational agents introduce flexibility, but they also create concerns about their reliability and grounding, especially in domain-specific scenarios [8, 9].

As a result, there is limited work that combines real-time speech interaction, controlled use of large language models, retrieval-based grounding, and embodied interaction within a single system. The aim of this thesis is to address this gap.

3 System Architecture and Design

3.1 System Overview

The system is designed as an end-to-end conversational pipeline capable of voice and text interaction through embodied interface. Input from the user is captured in the Unity frontend [14] and processed by a FastAPI backend [15], which delivers a validated response through speech and avatar behaviour.

The current implementation targets hospital information services and uses a controlled knowledge base as the main source of responses. The system operates in a closed domain where all outputs are grounded in predefined data and validated before being returned to the user.

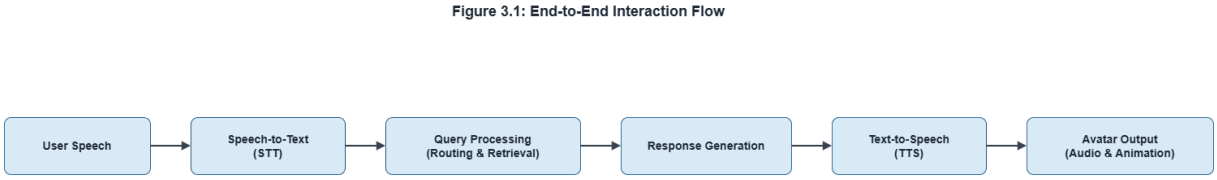


Figure 3.1: End-to-end interaction flow of the system.

3.2 System Architecture

The system consists of three components, a Unity frontend [14], a FastAPI backend [15] and additional services. The frontend handles user interaction including text input, audio capture and playback, and avatar visualization. It sends requests to the backend and renders responses through synchronized audio and animation. The backend exposes the main endpoints, including `/ask`, `/stt`, and `/tts`, and manages the processing pipeline. It handles query interpretation, retrieval, response selection, and session state. The additional services include speech recognition, speech synthesis, the structured knowledge base, retrieval indices, and external language model APIs used for constrained tasks. Communication between the frontend and backend is handled through HTTP requests.

3.3 Backend Processing Pipeline

The backend processes user input through the /ask endpoint. Each request is handled within a multi-stage pipeline consisting of routing, context handling, retrieval, and response validation.

Figure 3.2: System Architecture

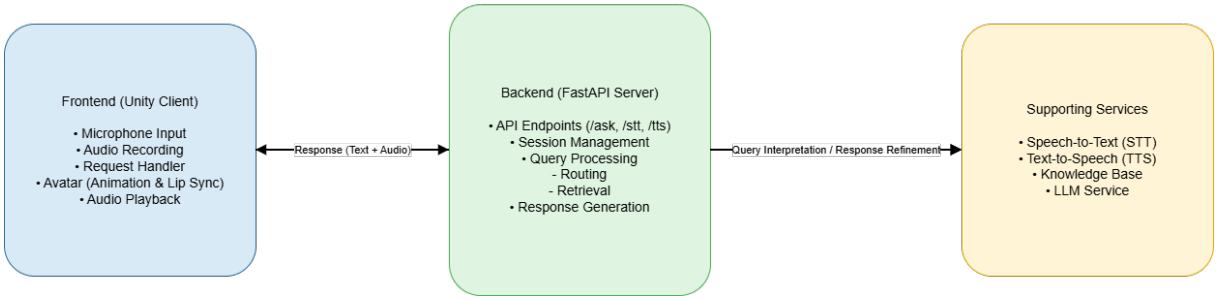


Figure 3.2: System architecture showing frontend, backend, and supporting services.

The routing module classifies the query into predefined categories (e.g. domain query, follow-up, social interaction) and extracts metadata such as query type and referenced entity. This determines whether the request is forwarded to retrieval or handled with predefined responses.

Session state is used for follow-up queries to rewrite the input before retrieval. The system stores minimal context per session and it gives priority to information mainly in the current query.

Candidate responses are retrieved from the indexed knowledge base and are ranked using a hybrid scoring mechanism which combines semantic similarity and lexical matching. The final response is selected after validating against query type and entity constraints. If there are not any valid candidates found, the system returns a fallback response.

Figure 3.3: Backend Processing Pipeline

Figure 3.3: Backend processing pipeline.

3.4 Data and Knowledge Flow

The system uses a structured knowledge base as its main source of information. Before runtime, the knowledge base is converted into retrieval artifacts, including canonical answers, FAQ-style query items, BM25 statistics [16], dense embeddings [7], and alias mappings. During query processing, the retrieval engine matches the user input against these indexed items. The purpose of this design is to separate factual content from query variations, so that different user phrasings can still map to the same canonical answer.

3.5 Speech and Avatar Interaction

Voice input is captured in Unity [14] and sent to the /stt endpoint. The backend applies voice activity detection [17] to remove silence and processes the filtered audio using a Whisper-based transcription model [12]. The resulting text is passed to the main processing pipeline. After a response is selected, it is sent to the /tts endpoint, where speech is generated and returned to the frontend. The audio is played through the avatar using Unity's audio system [18].

Lip synchronization is implemented using viseme-based animation driven by the audio signal. Playback events trigger speaking states and basic avatar behaviour, which helps align the audio and visual output. The interface includes a chat window, text input, and voice controls, with visual feedback for user interaction.

3.6 Design Decisions and Trade-offs

The system is designed to follow a retrieval-first approach where all responses remain grounded in predefined data. Language models are used only for constrained tasks such as routing and refining responses. Hybrid retrieval was selected to balance flexible wording and match with domain-specific terms [7][16]. Communication is REST-based, since it provides simplicity and stability. While streaming approaches could reduce latency, they were not required for the current implementation. Speech processing is split between local transcription and external services for synthesis and LLM tasks, which improves performance without overly complicating the system. The frontend uses minimal animations to maintain a consistent and professional appearance, focusing on lip synchronization and basic motions rather than complex behaviour.

4 Implementation

4.1 Development Tools and Technologies

4.1.1 Python, FastAPI and Uvicorn

For the backend, Python was used for the server side logic, retrieval pipeline, speech processing and integration with external AI services. FastAPI [15] was used to expose REST endpoints such as `/ask`, `/stt`, `/tts`, and `/health`, allowing the Unity frontend to access backend services for querying, speech recognition, and speech synthesis. During development, the FastAPI application was executed using Uvicorn [19], which serves as the ASGI server responsible for running the backend locally.

4.1.2 Unity and C#

Unity game engine [14] was used for building the frontend, while the logic of interactions was written using C#. Unity was chosen due to its capabilities to perform real-time 3D rendering, audio input/output, control animations, and integrating avatars in one environment. It was also used for hospital reception scene rendering, user inputs collection, display of the chat window, audio synthesis playback, and controlling avatar movement. The C# scripts were used for implementing microphone input recording, sending and receiving data from the backend, playing back audio, handling events, lips syncing, and avatar animation.

4.1.3 Speech Processing Tools

Speech interaction was implemented using both speech-to-text and text-to-speech technologies. Speech recognition was done through the `faster-whisper` library [20], which serves as an efficient implementation of the Whisper transcription model [12]. Voice activity detection uses WebRTC VAD [17], which enables the detection and removal of silent or irrelevant portions of the recording before transcribing them. For speech synthesis, the system uses an Edge Neural TTS provider [21], which generates spoken responses from the selected text answer.

4.1.4 Retrieval and NLP Libraries

The retrieval component uses a combination of semantic and lexical techniques. Semantic retrieval is supported by `sentence-transformers`, specifically the `all-mpnet-base-v2` model

[22], which is used to generate dense embeddings for query matching. Lexical retrieval is implemented with a custom BM25-based scoring mechanism [16], using precomputed BM25 statistics generated during indexing. NumPy is used for numerical operations, including vector similarity computation and score combination. These tools allow the system to rank candidate answers by combining semantic similarity with exact keyword matching.

4.1.5 Language Model Providers

External language model providers are used selectively in the system. Groq and OpenRouter are integrated for constrained tasks such as query routing, entity extraction, follow-up interpretation, and response refinement. The language model is not used as the main factual source of answers, and instead factual responses are retrieved from the controlled knowledge base. This design allows the system to benefit from the flexibility of LLMs while limiting the risk of unsupported responses or hallucinations [8].

4.1.6 Avatar and Animation Tools

The embodied frontend uses a Ready Player Me avatar [23] imported into Unity as a rigged 3D model with blend shape support. Lip synchronization is implemented using OVR LipSync [24], which processes audio frames and produces viseme values that are mapped to the avatar's facial blend shapes. Unity's Animator system is used to control basic animation states, including idle and speaking behaviour. Mixamo animations were also tested during development, but the final implementation uses minimal animations to maintain a professional and non-distracting appearance that suits a hospital information assistant.

4.1.7 Development and Testing Tools

Development was carried out using PyCharm for backend implementation and the Unity Editor for frontend development and testing. The backend was tested through direct endpoint calls, backend logs, and runtime debugging, while the Unity side was tested using the Unity console and visual inspection of audio playback, microphone input, UI behaviour, and avatar synchronization. GitHub was used for version control and project organization. Overleaf was used for preparing the written thesis document in LaTeX format. This type of tools section follows the pattern found in UCY project theses, where the development technologies are listed and explained before the implementation details.

4.2 Backend Implementation

The backend is implemented with FastAPI [15] and manages the interaction pipeline, which includes routing, retrieval, speech services, controlled LLM usage, and response validation.

4.2.1 API Design and Endpoints

The backend exposes a set of HTTP endpoints which are implemented in the main server module.

The `/ask` endpoint is the main entry point for handling user queries. The endpoint accepts text input and processes it through routing, retrieval, and returning structured responses with relevant metadata.

The `/stt` endpoint is responsible for converting audio input into text output by receiving audio data from Unity and providing the corresponding transcription.

The `/tts` endpoint converts the text response into audio data through text-to-speech synthesis.

The `/health` endpoint provides a light way to monitor the system's performance during development.

The following snippet illustrates how a user query is initially handled by the system:

```
@app.post("/ask")
def ask(payload: AskReq):
query = (payload.question or "").strip()
lang = normalize_lang(None, query)
state = get_session_state(payload.session_id)

routed = route_user_turn(
    query=query,
    session_state=state,
    openrouter_key=LLM_AVAILABLE_KEY,
)

route = routed["route"]
query_type = routed.get("query_type")
```

This highlights how the system processes the user query, recognizes the language, retrieves the session state, and passes it for routing.

4.2.2 Query Processing and Routing

The `/ask` endpoint begins with query interpretation and routing. In implementation, the router component classifies the input into categories such as domain query, follow-up question, social interaction, emergency query, or out-of-scope request. It also extracts fields such as query type and subject text, which are then used by the retrieval and validation stages. Routing uses a constrained LLM call, where the model is prompted to return structured output which is parsed and validated before the values are used by the backend.

The routing module returns a structured JSON object containing the route type, query classification, and extracted subject.

```
content = llm_chat(system, user, task="router")
parsed = _extract_first_json_object(content)

route = parsed.get("route")
query_type = parsed.get("query_type")
subject_text = parsed.get("subject_text")
```

4.2.3 Retrieval Engine Implementation

For queries specific to certain domains, the system uses a hybrid retrieval engine over the structured knowledge base. Semantic retrieval uses dense embeddings created by the sentence transformer model [22], while lexical retrieval uses BM25 scoring [16] for calculating exact term matches, including department names and service-related queries.

After normalization, both scores are combined into a single ranking score.

```
dense = self.q_emb @ qv
dense_norm = (dense + 1.0) / 2.0

lexical = np.array([
self.bm25_score(q_tokens, it.get("q_tokens", []))
for it in self.faq_items
], dtype=np.float32)

hybrid = 0.65 * dense_norm + 0.35 * lexical_norm
```

The final score is computed as a weighted combination of semantic and lexical components. After computing the base hybrid score, the implementation also applies small boosts for aligning departments, previous session departments, compatibility with query-type, and matching

language, which prioritizes candidates that are semantically similar, and also suitable for the current context.

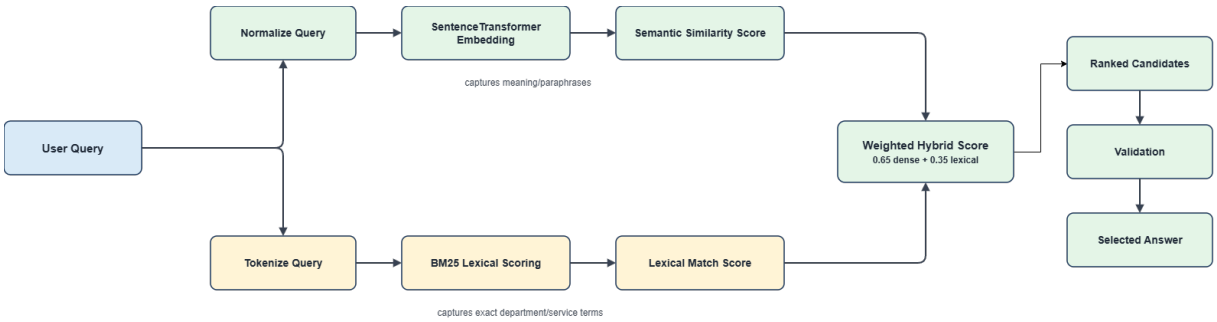


Figure 4.1: Hybrid retrieval scoring process combining semantic similarity and BM25 lexical matching before response validation.

4.2.4 Response Selection and Validation

After retrieval, the candidate responses are evaluated before returning a final answer. The system checks relevance, compatibility with query type, and alignment with extracted entities. If no candidate satisfies the required conditions, the backend returns a controlled fallback or clarification response.

A confidence-based filtering mechanism is applied:

```
if best["hybrid"] < self.hybrid_min_low:
return "low"

gap = best["rerank"] - second["rerank"]

if gap >= 0.35:
return "high"
```

These thresholds determine if the response is reliable or if it requires fallback handling. The selected responses can optionally be rephrased using a constrained language model.

4.2.5 Session Management and Context Handling

To support multi-turn interaction, the backend maintains a light session state per user. This allows the system to interpret follow-up queries based on previous context, without storing unnecessary dialogue history, as it is updated after each interaction. The backend keeps recent information such as the last department, last query type, last answer identifier, and last resolved query. When a follow-up is detected, this state can be used to rewrite or interpret the new query, while explicit information in the current input remains the priority.

4.2.6 Error Handling and Reliability Mechanisms

The backend includes fallback mechanisms for invalid input, low confidence retrieval, and external API failure. The LLM providers are considered non-critical parts of the system, since if one provider fails, the system attempts the next available LLM provider or continues the process without LLM output.

```
if task == "router":
groq_model = "llama-3.3-70b-versatile"
else:
groq_model = "llama-3.1-8b-instant"

try:
response = groq.chat(system, user, model=groq_model, timeout=10)
if response:
return response
except Exception:
pass

try:
response = openrouter.chat(system, user, timeout=10)
if response:
return response
except Exception:
pass

return None
```

These fallback strategies help keep the system operational even when dependencies are unavailable. In addition, there is validation of the intermediate output at each stage to prevent error propagation.

4.3 Knowledge Base Construction and Indexing

The system uses a structured, answer-first knowledge base with precomputed retrieval artifacts. The knowledge base was constructed from publicly available information on the official Nicosia General Hospital website [25], specifically the clinics and departments pages provided by the State Health Services Organisation (SHSO). Only information available from the official source was included, and fields that were not provided were left empty rather than completed manually.

This approach ensured that the system's answers remained grounded in an identifiable public source.

4.3.1 Knowledge Base Structure

The knowledge base is stored in a structured JSON form (`kb_canonical.json`), containing information about hospital departments. Department information includes multilingual labels, aliases, and a set of factual entries.

```
{
"id": "accident_emergency_department",
"department_en": "Accident & Emergency Department",
"aliases_en": ["A&E", "Emergency Department"],
"facts": [
{
"fact_id": "aed_overview",
"answer_en": "The Accident and Emergency Department..."
}
]
}
```

Facts are represented as answers in a canonical form. Metadata for each fact such as tags (e.g. overview, services, process) are later used to align queries with answer types.

4.3.2 Answer Bank Construction

An answer bank is created based on the canonical knowledge base. Each answer is assigned a unique identifier and annotated by additional metadata, such as language, department, scope, and types of queries.

```
{
"answer_id": "ans_196957ff3bee",
"canonical_answer": "The Accident and Emergency Department...",
"lang": "en",
"department": "accident_emergency_department",
"tags": ["overview", "hours"]
}
```

The use of unique answer identifiers allows multiple query variations to map to the same canonical response.

Answer IDs are generated using a hash of the answer content and metadata:

```
def answer_id_from_text(answer, lang, department, entity_name):
raw = f"{lang}::{department or
    ''}::{normalize(entity_name)}::{normalize(answer)}"
return "ans_" + sha1(raw.encode()).hexdigest()[:12]
```

This guarantees stable identifiers even if the knowledge base is rebuilt.

4.3.3 Query Expansion and FAQ Item Generation

In order to support flexible user input, the system generates multiple instances of each query (FAQs) by paraphrasing or using multilingual equivalents, which correspond to one single canonical response.

```
{
"item_id": "item_00000",
"q": "What is Accident & Emergency Department?",
"q_tokens": ["what", "is", "accident", "&", "emergency"]
}
```

Each FAQ item is linked to a corresponding `answer_id`, allowing different query variations to resolve to the same answer. Each generated query item is also normalized and tokenized during preprocessing, so that lexical and semantic retrieval use a consistent representation.

4.3.4 Lexical Indexing

For lexical matching, the system computes BM25 statistics [16] over all query items. The statistics include IDF (inverse document frequency) scores and average document length.

```
{
"avgdl": 7.34,
"idf": {
"accident": 4.58,
"emergency": 4.14
}
}
```

They are then used during runtime to calculate BM25 scores for query-document matching, which allows the system to prioritize exact keyword matches, particularly for department names and specific services.

4.3.5 Semantic Embedding Generation

In parallel with lexical indexing, the system generates dense vector embeddings for all query items based on a pre-trained SentenceTransformer model. The embedding model used is sentence-transformers/all-mpnet-base-v2, which was selected because of its ability in semantic similarity tasks.

The generated embedding matrix is stored as a NumPy array in `q_emb.npy` and loaded by the backend at runtime. This enables fast similarity computation between user queries and stored items using vector operations.

4.3.6 Department Alias Mapping

To improve entity recognition, The system builds an alias mapping between user-facing names and internal department identifiers. This includes abbreviations, alternative names, and Greek variants.

```
{
"a&e": "accident_emergency_department",
"emergency": "accident_emergency_department",
"taep": "accident_emergency_department"
}
```

This mapping allows the system to resolve user queries even when informal or abbreviated terms are used, in order to maintain consistent matching across different input forms.

4.3.7 Index Build Process

Preprocessing is performed offline through the file `build_index.py`. The script loads the canonical knowledge base, generates answer and query items, computes BM25 statistics, creates embeddings, and stores the final runtime artifacts.

The final output consists of the precomputed files, like `faq_items.json`, `answer_bank.json`, `bm25_stats.json`, `q_emb.npy`.

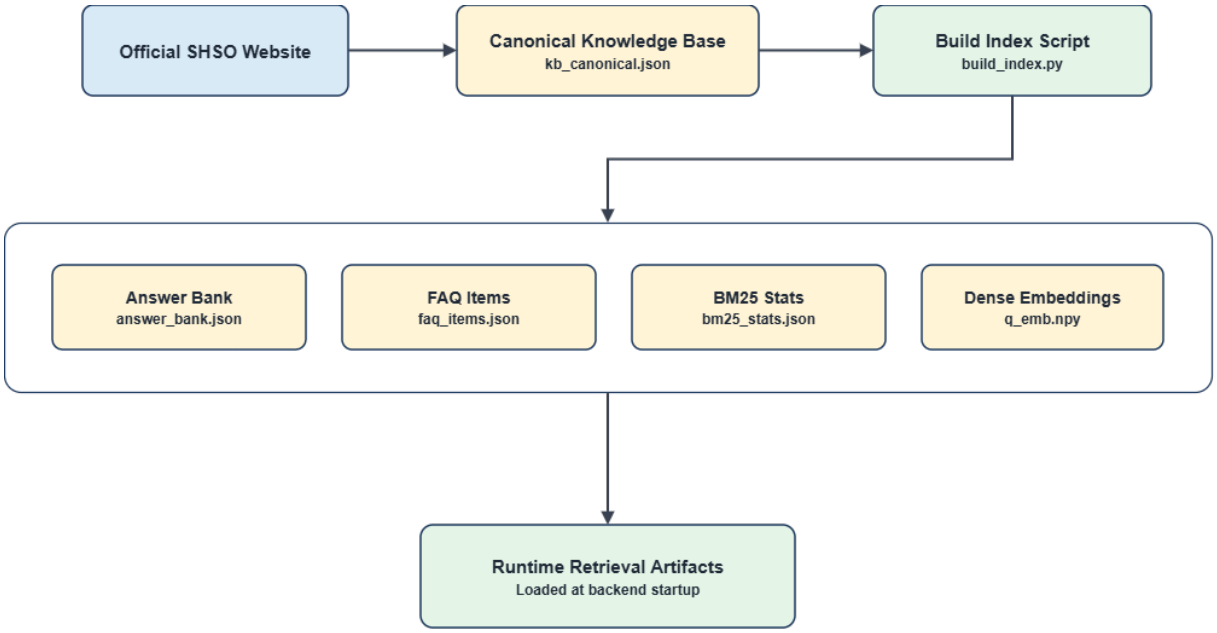


Figure 4.2: Offline index build process used to generate the retrieval artifacts loaded by the backend at runtime.

A build summary records the scale of the index:

```
{
"n_items": 2491,
"n_answers": 765,
"embedding_shape": [2491, 768]
}
```

4.3.8 Design Considerations

The indexing pipeline separates canonical answers from query variations. This keeps the knowledge base consistent while still allowing flexibility in user input. Since embeddings and BM25 statistics are precomputed, runtime retrieval does not require heavy computing. This approach also supports adaptabilty, as new departments or facts can be added to the knowledge base and re-indexed without modifying the core retrieval pipeline.

4.4 Speech Processing Implementation

Speech-to-Text (STT) and Text-to-Speech (TTS) are incorporated into the request/response pipeline used by text inputs. The audio is captured in Unity and processed by the backend to generate a speech response that can be played through the avatar.

4.4.1 Overview of the Speech Pipeline

Speech interaction follows a structured pipeline where audio input is captured in Unity, processed in the FastAPI backend, and returned as synthesized audio for playback. Intermediate processing is performed in text form, allowing integration with query-processing.

Figure 4.3 summarises the speech processing flow used by the system.

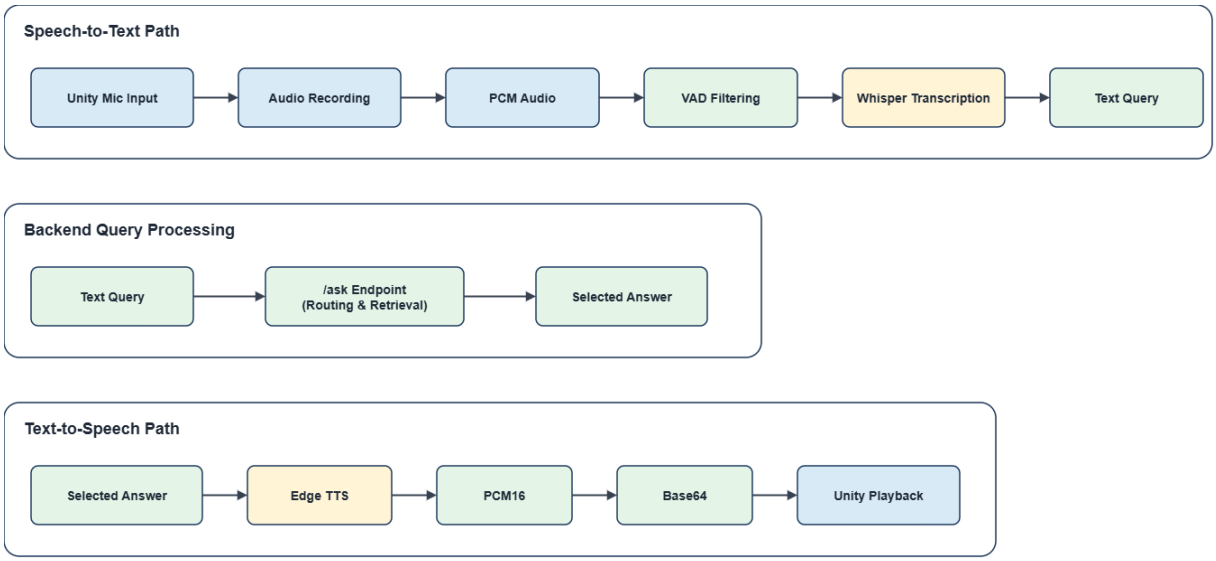


Figure 4.3: Speech processing implementation flow showing the speech-to-text, backend query processing, and text-to-speech stages.

4.4.2 Speech-to-Text (STT) Processing

Whisper [12] is utilized on the backend for speech recognition, and is accessible through the `faster-whisper` library [20]. The `/stt` endpoint receives raw PCM audio from the Unity client and performs preprocessing before transcription. The backend uses the Whisper base model with int8 inference, which reduces processing time while still providing sufficient transcription quality for the prototype.

To improve transcription accuracy and reduce noise, the system applies Voice Activity Detection using the WebRTC VAD library [17]. The audio is divided into frames, and frames containing speech are stored:

```
vad = webrtcvad.Vad(2)
if vad.is_speech(frame.tobytes(), sr):
voiced_frames.append(frame)
```

Padding is applied before and after detected speech segments to avoid clipping the beginning or end of spoken input. In the implementation, short padding windows are added before and after

the detected speech region before the filtered audio is passed to Whisper.

The filtered audio is then passed to Whisper for transcription:

```
segments, _ = model.transcribe(speech_f32, beam_size=1)
text = " ".join(seg.text.strip() for seg in segments)
```

This approach removes silence and background noise before transcription. The final output is a clean text string, which is forwarded to the main `/ask` endpoint for processing.

4.4.3 Text-to-Speech (TTS) Processing

TTS processing is handled on the backend through the Edge Neural TTS provider, which generates natural sounding speech from text responses. The `/tts` endpoint receives the response text and produces audio in PCM format.

The voice is selected automatically depending on the detected language. The system uses el-GR-AthinaNeural for Greek responses and en-US-AriaNeural for English responses. After synthesis, the generated audio is decoded, converted into PCM16 format, encoded as base64, and returned to the Unity frontend.

4.4.4 Frontend Audio Handling and Playback

In the Unity environment, audio input is captured through the computer microphone and stored as an AudioClip [26]. The recorded audio is then converted into raw waveform format and transmitted to the backend.

```
clip = Microphone.Start(null, false, maxDuration, sampleRate);
```

After the synthesized response is received, Unity reconstructs the audio and plays it through an AudioSource [18].

4.4.5 Avatar Synchronization and Lip Sync

To enhance user experience, the system synchronizes speech output with avatar animation. This is achieved using viseme based lip synchronization, where audio features are mapped to facial blend shapes in real time.

```
ovrLipSync.ProcessFrame(audioData, frame);
```

The resulting viseme values are applied to the avatar's mesh [27]:

```
skinnedMeshRenderer.SetBlendShapeWeight(index, value);
```

Speaking animation is still triggered during playback, but the audio-driven viseme processing is what connects the generated speech to visible mouth movement.

4.4.6 Latency and Real-Time Constraints

Latency in speech interaction can degrade the user experience [3]. To address this, audio is captured locally to minimise input delay, while Voice Activity Detection (VAD) removes silence and reduces unnecessary processing. Transcription is performed using a light Whisper model with int8 inference to improve speed, and responses are generated through a retrieval-based approach, avoiding the overhead of fully generative models. On the output side, the use of streaming text-to-speech reduces the delay before playback begins. While latency is still impacted by model processing and network delay, it is reduced by limiting the complexity of models being used.

4.4.7 Design Considerations and Trade-offs

The chosen approach was request-response communication between client and server rather than real-time streaming since it is easier to implement. This makes development and debugging much better, especially because there is no complexity with partial outputs that should be synchronized. Even though streaming could have reduced latency even further, it would also introduce additional challenges and error cases, and was therefore not considered necessary for this prototype.

4.5 Unity Frontend and Avatar Implementation

The frontend implementation was developed with the Unity game engine and it manages user interaction, audio input/output, and avatar visualization. It communicates with the backend through HTTP requests, and all visual rendering and interactions are processed locally.

4.5.1 Scene Design and Environment Setup

The scene uses a hospital reception area to match the target use case. A pre-built hospital environment asset [28] was integrated into the Unity scene and modified to fit the application requirements. The original environment asset was larger than required, containing multiple rooms and assets irrelevant to the interaction scenario. To address this, the environment was selectively reduced to a single reception-like space, reducing unnecessary geometry and adjusting scene scale to ensure that the avatar and interface elements remain clearly visible.

The final layout includes the avatar, a hospital background, and a minimal interaction interface. As shown in Figure 4.4, the interface presents the avatar with a chat panel, input field, microphone button, and send button.

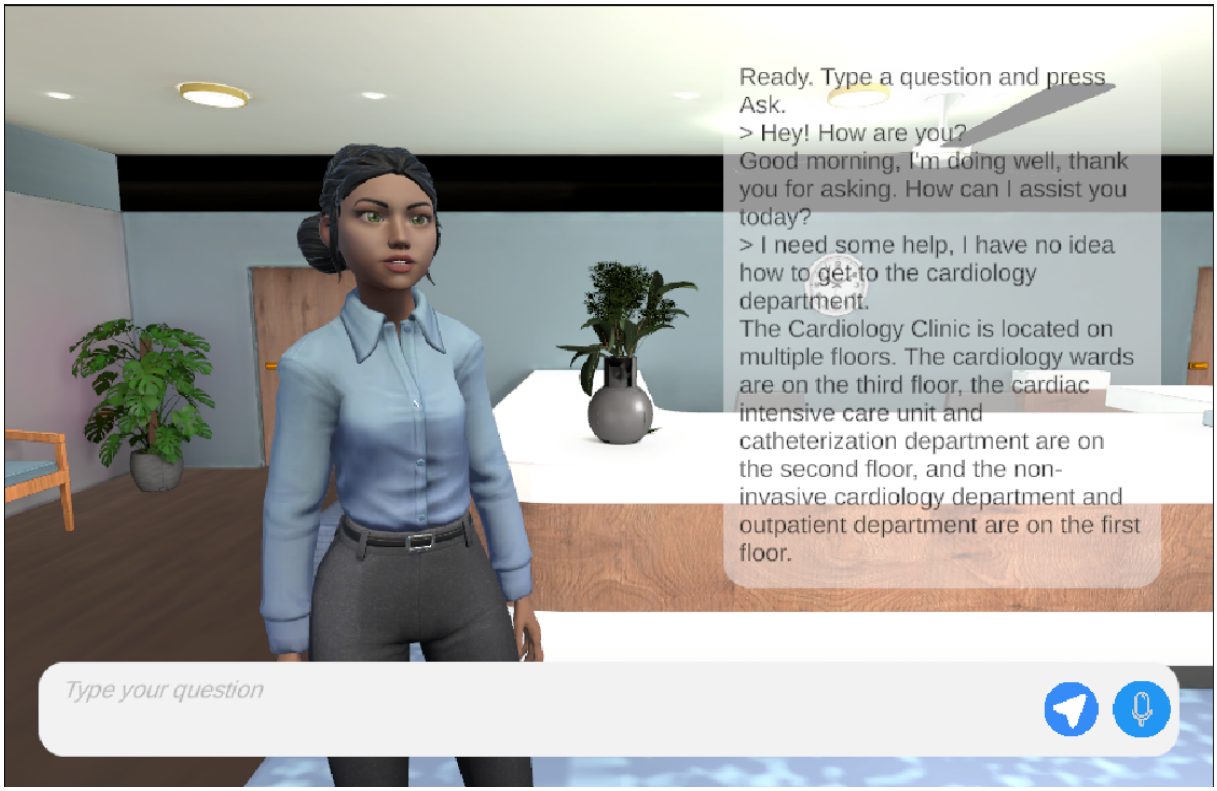


Figure 4.4: Unity frontend showing the embodied assistant, hospital reception environment, chat panel, text input field, send button, and microphone control.

4.5.2 Avatar Integration and Customization

The avatar was created using the Ready Player Me platform [23] and then imported into Unity as a rigged 3D model with the support of blend shapes. It has been customised to represent a neutral and professional assistant that aligns with the hospital environment use case. To support animation and lip synchronization, the avatar was configured with appropriate blend shape mappings and integrated into the Unity Animator system. During development, more expressive full-body animations from Mixamo Assets were tested, but they appeared distracting in this context. The final implementation therefore uses a smaller animation set based on idle breathing, subtle head movement, and speech-driven facial animation, in order to maintain functional realism and avoid unnatural behaviour.

4.5.3 User Interface Design and Interaction Elements

The user interface consists of an overlay within the Unity scene, supporting both text and speech interaction. The interface features a scrollable chat window for past conversations, a text input field, a send button, a microphone button, as well as visual feedback that offers better usability. Interactive elements such as the send and microphone buttons change states when pressed, while the microphone button shows a dynamically expanding circular animation of the detected sound levels during recording.

4.5.4 Input Handling and Backend Communication

User input is handled through both text and voice, which are connected before being processed by the backend. Voice input is captured using Unity's Microphone API [26], where the audio is converted into a transferable format and sent to the backend `/stt` endpoint for transcription. Text input is sent directly to the `/ask` endpoint. The frontend coordinates communication with multiple backend endpoints, including speech-to-text, query processing, and text-to-speech, where the requests are performed asynchronously in order to prevent blocking the network communication.

Figure 4.5: Example end-to-end interaction showing a follow-up query handled through the Unity frontend and answered using the backend retrieval pipeline.

Figure 4.5 shows an example where the user asks for contact information and then uses a follow-up query for another department, demonstrating that frontend interaction, backend context handling, retrieval, and response playback operate as a complete pipeline.

4.5.5 Audio Playback and Avatar Synchronization

Audio playback is managed through a dedicated component that reconstructs received audio data into a Unity AudioClip and plays it through an AudioSource [18]. Playback events are used to trigger avatar behaviour, ensuring synchronization between speech and animation. Lip synchronization is implemented using the OVR LipSync system [24], which processes audio frames to generate viseme data. These viseme values are mapped to the avatar's blend shapes, producing real-time mouth movement aligned with the audio output, while animation drivers control the speaking state and head movement.

Figure 4.6: Avatar lip synchronization during speech playback, showing different mouth positions generated from audio-driven viseme animation.

4.5.6 Event-Driven Interaction Model

The frontend uses an event-driven architecture where shared events coordinate the main stages of the interaction, including recording, backend requests, response reception, and audio playback. This allows components such as the voice recorder, request handler, chat interface, audio player, and animation controller to work separately while still staying synchronized during the conversation.

4.5.7 Design Considerations and Trade-offs

Although a 3D environment and avatar increase the quality of the interaction process, they also introduce synchronization and performance issues. For this reason, the final frontend keeps the interface simple and limits animation complexity. Together with the event-based structure, this helped keep the system stable and responsive.

4.6 Integration Challenges and Limitations

Integrating the Unity frontend, FastAPI backend, retrieval engine, speech modules, external LLM providers, and avatar animation system introduced practical challenges around reliability, latency, synchronization, and debugging. These issues mainly came from connecting the components into one complete interaction loop, rather than from the individual modules only.

One challenge was LLM-based routing, especially for short, informal, or ambiguous queries. To reduce this, the router was constrained to return structured JSON output, which is parsed and validated before use, while provider fallback logic allows the system to continue if one LLM provider fails. Latency was also an issue because a full interaction includes audio capture, transcription, routing, retrieval, optional LLM processing, TTS generation, and playback. The system reduces this delay through VAD filtering, a light Whisper model, precomputed retrieval artifacts, and a retrieval based design, but response time is still affected by model processing and network delay [3]. Avatar synchronization also required adjustments, since early versions showed mismatch between audio playback, lip movement, and animation states. This was improved by using event-driven playback triggers and audio-driven lip synchronization. Finally, ambiguous or queries with typos remained difficult, so confidence thresholds and fallback responses were used to avoid returning hospital information that was not supported.

Debugging was handled by separating the main stages into different endpoints, including `/stt`, `/ask`, `/tts`, and `/health`, and by using backend logs together with Unity console output. Figure 4.7 shows the backend debug log for the same follow-up interaction shown earlier in Figure 4.5. The log shows the received query, routed follow-up interpretation, resolved query, retrieved candidates, and selected response before speech output is generated.

Figure 4.7: Backend log output for the follow-up interaction shown in Figure 4.5, showing query routing, retrieval candidates, response selection, and text-to-speech execution.

All in all, the final system handles these issues through constrained LLM usage, fallback mechanisms, precomputed retrieval, validation layers, and frontend coordination with events.

5 Evaluation and Results

5.1 Evaluation Objectives

The evaluation examined whether Lana can effectively support hospital-related information tasks through a conversational embodied interface. The system was evaluated as a complete prototype, including speech recognition, backend retrieval, response generation, text-to-speech synthesis, avatar interaction, and the Unity frontend.

The evaluation focused on three main areas. First, it examined whether users could successfully complete common hospital information tasks such as finding departments, contact information, appointments, and hospital guidance information. Second, it evaluated whether the system could remain within its intended role by correctly handling unsupported or out-of-scope questions. Third, it assessed the overall user experience, including usability, conversational quality, responsiveness, speech interaction, and avatar presence. The evaluation was designed as a prototype feasibility study focused on usability, reliability, and usefulness in a controlled environment.

5.2 Evaluation Methodology

5.2.1 Participants

The evaluation involved 18 participants with different age groups and different familiarity with conversational AI systems. The participant group included 10 female and 8 male participants, with most of them belonging to the age range of 18-34. Most participants also reported previous experience using conversational assistants such as Siri, Alexa, or ChatGPT, although very few had previous experience with healthcare related virtual assistants.

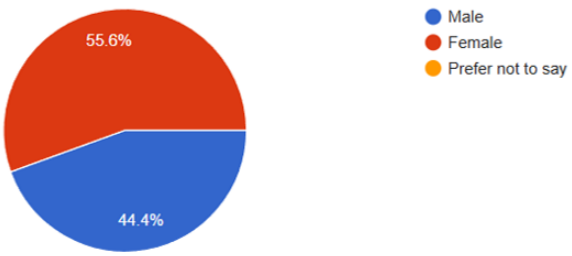


Figure 5.1: Gender distribution of participants

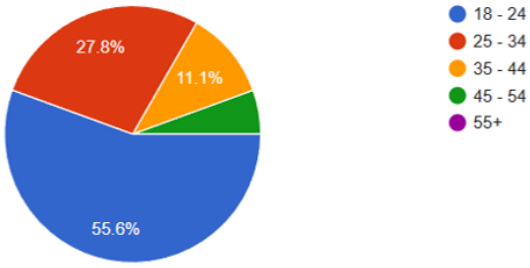


Figure 5.2: Age distribution of participants

Participants were also asked about their familiarity with conversational assistants. Most participants reported feeling comfortable using voice interaction systems and regularly using conversational assistants in daily activities. However, the majority had not previously interacted with hospital-related conversational systems, making the evaluation useful for testing first-time user experience in a healthcare setting.

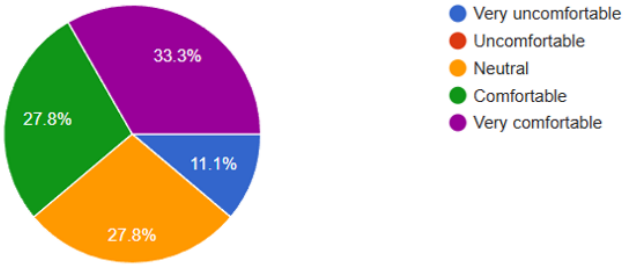


Figure 5.3: Participant comfort level with voice interaction systems

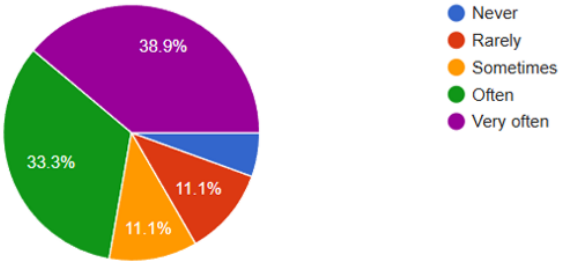


Figure 5.4: Frequency of conversational assistant usage

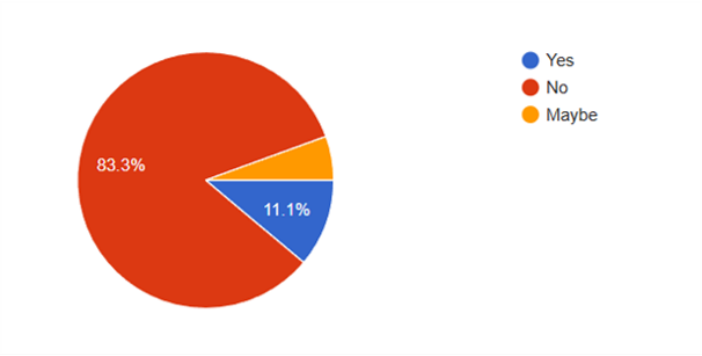


Figure 5.5: Previous experience with healthcare conversational assistants

5.2.2 Evaluation Environment

The evaluation was executed using the Unity-based prototype running on a laptop equipped with a microphone and speaker, as participants interacted with the system using both speech and text input. The evaluation sessions were performed in a controlled indoor environment to reduce background noise and retain stable speech recognition performance.

5.2.3 Evaluation Procedure

Participants first received a short explanation about the system and available interaction methods before completing a set of predefined hospital-related scenarios. These scenarios included department location, appointment questions, payment methods, ICU visiting hours, accessibility-related guidance, unsupported hospital questions, and out-of-scope interactions. After completing the tasks, participants answered a structured questionnaire using five-point Likert-scale ratings and provided optional open-ended feedback about strengths, weaknesses, and possible improvements.

5.2.4 Questionnaire Design

The questionnaire combined quantitative and qualitative feedback. The quantitative section used five-point Likert-scale responses that covers usability, interaction quality, speech understanding, conversational flow, avatar embodiment, usefulness, and reliability. Participants also provided open-ended comments discussing positive aspects of the system, difficulties, and suggestions for future improvements.

5.3 Task-Based Evaluation Results

The task-based evaluation examined whether participants could successfully complete realistic hospital-related information tasks using Lana. Each participant completed predefined interaction scenarios and recorded whether the task was completed successfully, partially completed, or failed.

Table 5.1: Task-based evaluation results

Scenario	Successful	Partial	Failed
Finding a department	16	2	0
Contacting a department	17	1	0
Booking an appointment	17	1	0
Follow-up question handling	13	4	1
ICU visiting hours	17	1	0
Payment methods	17	1	0
Hospital rules	18	0	0
Unsupported hospital question	18	0	0
Out-of-scope question	18	0	0
Accessibility-related location query	17	1	0

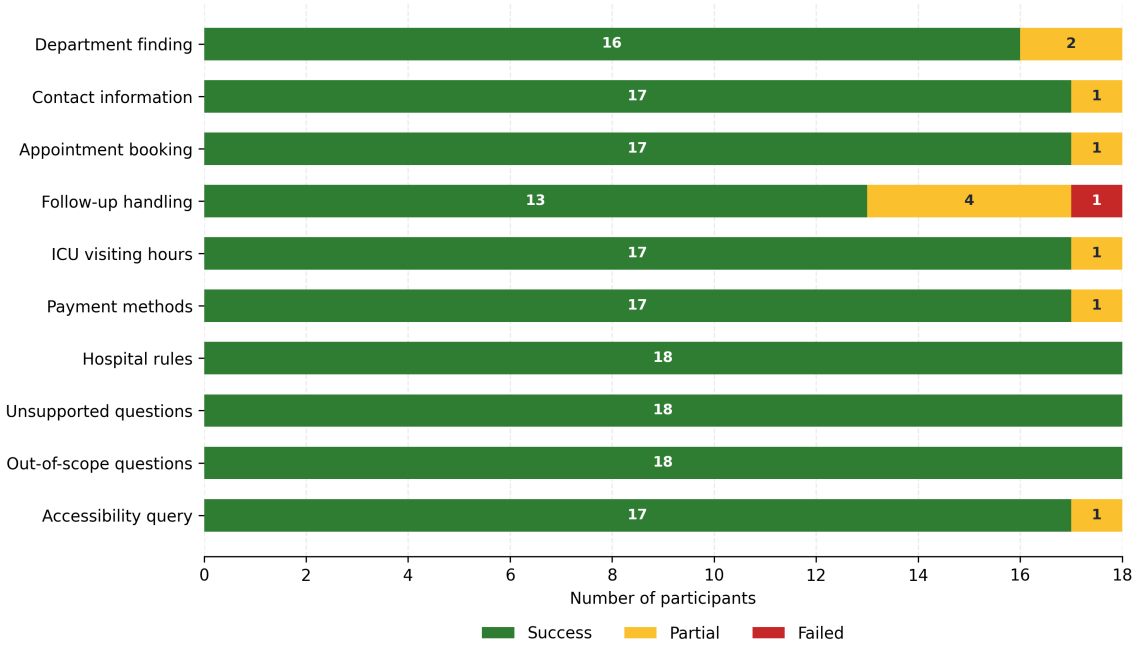


Figure 5.6: Task-based evaluation results across all hospital interaction scenarios

The results show strong performance across most hospital information tasks. Hospital rules, unsupported hospital questions, and out-of-scope requests achieved full success across all participants, showing that the system consistently remained within its intended role. High completion rates were also observed for department information, appointments, ICU visiting hours, payment methods, and accessibility-related queries, suggesting that the retrieval pipeline was effective for direct factual requests. The weakest area involved follow-up questions, where some participants needed to rephrase their questions or provide additional clarification before the correct information was retrieved. This indicates that the system is more reliable with explicit queries than with shorter conversational follow-ups that depend heavily on previous context.

5.4 User Evaluation Results

This section presents the questionnaire results collected after participants completed the interaction scenarios. The results are grouped into usability, conversational quality, embodiment, usefulness, and reliability categories in order to evaluate the overall user experience of the system.

5.4.1 Usability and Interaction Quality

Figure 5.7 presents participant responses related to ease of interaction, speech interaction preference, and interaction flow. Most participants rated the system positively, especially for easy use and overall the smoothness of the interaction. Responses regarding speaking instead of typing were slightly more mixed, although most participants still preferred speech interaction and considered it convenient.

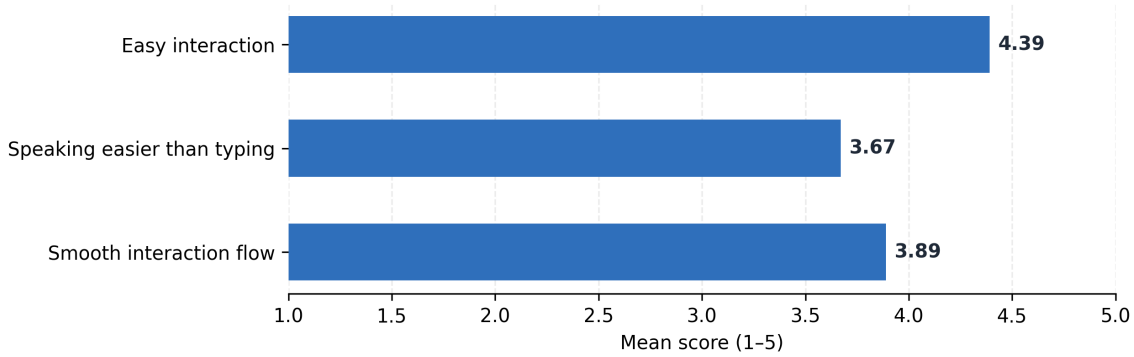


Figure 5.7: Usability and interaction quality results

The results suggest that participants were generally comfortable interacting with the system and could navigate the conversation without major difficulties, which is important since the system aims to support quick and accessible information retrieval in hospital environments.

5.4.2 Speech and Conversational Quality

Figure 5.8 summarizes responses related to speech understanding, response clarity, responsiveness, conversational naturalness, and wording flexibility. Answer clarity and relevance received particularly strong ratings, while speech understanding and conversational naturalness received slightly more varied responses.

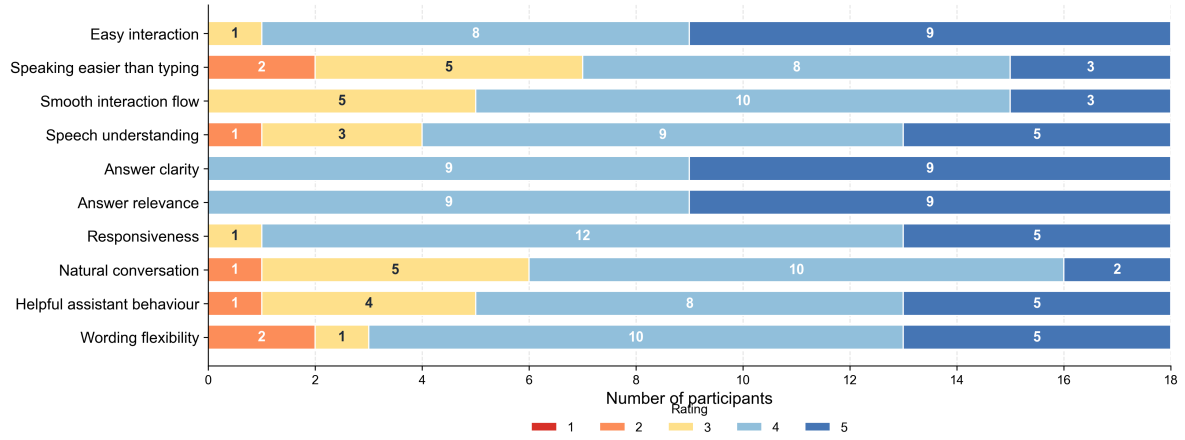


Figure 5.8: Speech and conversational quality results

The results indicate that participants generally considered the responses clear, useful, and relevant to their questions. However some participants experienced occasional wording or follow-up limitations, especially during shorter conversations. This is likely related to the retrieval-first design of the system, which prioritizes reliable responses from the knowledge base over highly flexible behaviour, as discussed in Chapter 4 [9],[29].

5.4.3 Avatar Embodiment and System Reliability

Figure 5.9 presents participant responses related to avatar engagement, lip-sync quality, attentiveness, trust, reliability, and role consistency. The system received particularly strong ratings for staying within its intended role and usefulness for spoken guidance.

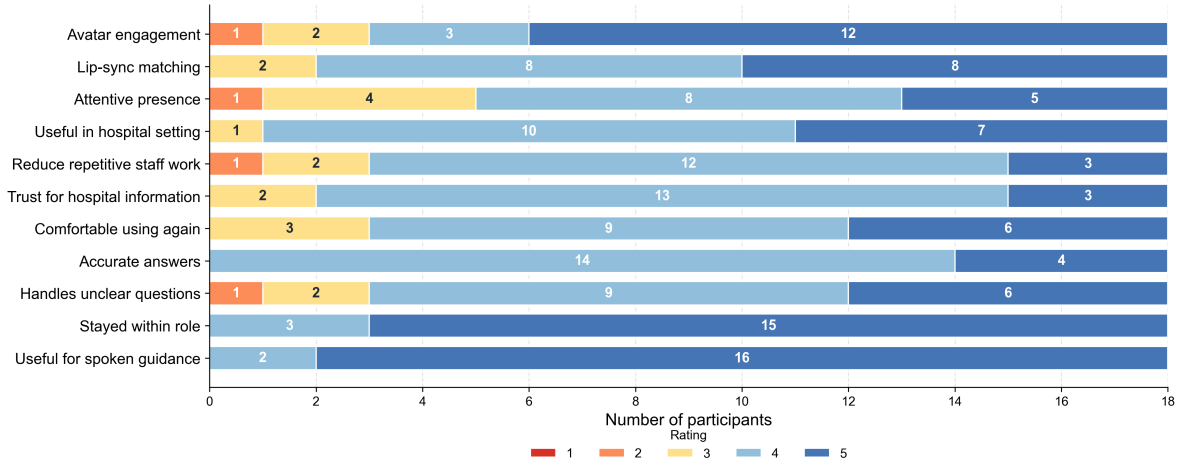


Figure 5.9: Avatar embodiment, usefulness, and reliability results

Participants also responded positively to the embodied avatar experience, particularly regarding engagement and lip synchronization. These results suggest that the embodied interface contributed positively to the interaction experience and helped the system feel more engaging than a traditional text-only chatbot [30],[31].

5.4.4 Comparative Evaluation Scores

Participants additionally rated the system in broader categories including overall experience, usefulness, responsiveness, confidence, and conversational naturalness. Figure 5.10 shows that most responses were concentrated in the high or highest rating categories, especially for usefulness and responsiveness.

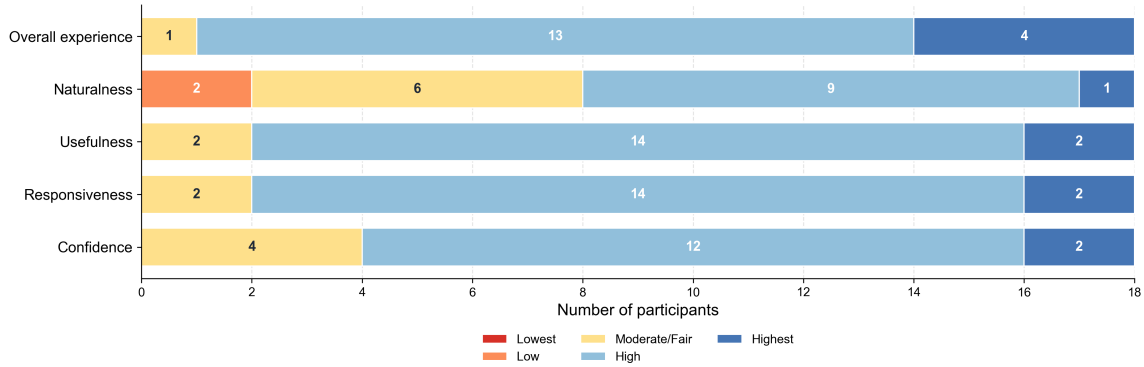


Figure 5.10: Comparative evaluation ratings

The lower ratings observed for the naturalness of the conversation are consistent with the follow-up limitations identified during the task-based evaluation. Despite this, the participants still rated the overall experience positively across most of the evaluation categories.

5.4.5 Qualitative Feedback

The participants also provided written feedback regarding aspects that they enjoyed, problems they experienced, and things that should be improved. Positive comments focused mainly on the avatar, usability, and the clarity of the answers. For instance, many participants mentioned that the avatar contributed to the realism, interaction, and engagement of the prototype. Other participants highlighted that the system responded quickly and gave answers that were concise, relevant, and did not contain redundant information. All this is consistent with the quantitative results that show that answer clarity, relevance, avatar interaction, and usefulness for spoken guidance received strong ratings.

The main issues reported were speech recognition, wording flexibility, and the size of the available knowledge base. For example, some users stated that they had to rephrase their queries due to difficulties with the system understanding their phrasing. In addition others mentioned that the microphone worked only with clear pronunciation and in an environment with low background noise [32],[12]. A few comments also suggested adding more detailed information, such as information about doctors, maps, and more Greek support. These comments are useful because they show that the current prototype works well for basic hospital information, but would need a larger knowledge base, stronger multilingual support, and more robust speech handling before being used in a real hospital setting.

5.5 Discussion

The evaluation results suggest that Lana performed well as a hospital information assistant within a controlled domain. Most participants completed the hospital-related tasks successfully, especially scenarios involving direct factual information such as appointments, department contacts, payment methods, hospital rules, and ICU visiting hours.

One of the strongest findings was that the system is able to stay within its intended role. Requests outside of the scope of the hospital or without supporting information were managed properly by all participants, showing that the combination of retrieval-based responses, validation steps, and constrained LLM usage helped reduce unrelated or unsupported answers [9],[29]. In addition the participants gave positive feedback regarding the avatar, stating that it made the system more engaging and realistic compared to a normal chatbot [30],[31].

The main limitation observed during the evaluation was mainly follow-up questions and wording flexibility. Some participants needed to rephrase questions or provide clearer wording before the correct answer was retrieved, especially during shorter conversations. A few participants also reported speech-recognition issues related to microphone quality or background noise [32],[12], while others suggested improvements such as better Greek support, doctor-related information,

and larger hospital data coverage.

The evaluation shows that the proposed system is effective for spoken hospital information assistance and provides a more interactive experience through speech and embodiment. At the same time the findings highlight areas that would require further improvement before deployment in a real hospital environment, such as conversational flexibility, multilingual support, improving speech recognition, and expanding the knowledge base.

5.6 Chapter Summary

This chapter presented the evaluation methodology and results for the Lana hospital information assistant. The results showed strong task completion performance, positive usability feedback, and the ability of the system to handle hospital information queries in a controlled domain. The participants also responded positively to the speech interaction and avatar embodiment features, while the evaluation additionally highlighted limitations related to follow-up flexibility, speech recognition, and knowledge base coverage. These findings are used in the next chapter to summarize the contributions of the thesis and discuss possible future improvements.

6 Conclusion and Future Work

6.1 Thesis Summary

This thesis presented the design, implementation, and evaluation of Lana, an embodied conversational assistant designed for hospital information support. The system combines speech interaction, retrieval-based question answering, constrained language model usage, and avatar embodiment within a Unity frontend connected to a FastAPI backend.

The implementation focused on maintaining reliable and grounded responses while still supporting a conversational interaction style. Instead of relying entirely on generative AI responses, the system was designed around a structured knowledge base and a hybrid retrieval pipeline, allowing answers to remain connected to predefined hospital information [9]. The evaluation results presented in Chapter 5 showed that participants were generally able to complete the predefined hospital-information tasks successfully, particularly for direct information requests such as appointments, department guidance, hospital rules, and payment questions. Users also provided positive feedback to the avatar and spoken interaction features, which implies that embodiment made the system feel more engaging than a traditional text-based chatbot [30],[31].

At the same time, the evaluation highlighted limitations related to follow-up, wording flexibility, and speech recognition. Participants occasionally needed to repeat or rephrase questions, especially during shorter conversational exchanges or when unclear wording was used. These findings reflect the design choices made during development, where the system focused more on reliable responses than highly flexible conversation.

6.2 Main Contributions

The main contribution of this thesis is the development of a complete end-to-end conversational assistant that combines speech interaction, retrieval-based information access, constrained LLM support, and embodied avatar interaction within a single system. The project demonstrates how these components can operate together as an interactive conversational pipeline.

Another contribution is the design of a modular architecture that separates the frontend interaction layer from the retrieval and knowledge components. Although the current implementation focuses on hospital information, the overall structure was intentionally designed so that the underlying domain can be changed without rebuilding the entire pipeline. Since the knowledge base, retrieval artifacts, and routing logic remain independent from the frontend interaction system, the same architecture could also be adapted for other controlled-domain environments such

as university support systems, museum guides, or public-service information assistants.

The evaluation itself also contributes useful observations regarding embodied conversational systems in controlled domains. The results showed that users valued fast and clear responses, but also responded positively to the avatar presence and speech interaction. At the same time, the weaker results in follow-up responses highlight some of the practical limitations involved in balancing natural conversation with reliable responses.

6.3 Limitations

Although the evaluation results were generally positive, the current prototype still has several limitations. The evaluation was conducted with a relatively small participant group and within a controlled environment, which means the results may not fully represent behaviour in real hospital settings with higher noise levels and more unpredictable interactions.

The knowledge base currently contains a limited set of hospital information collected from publicly available sources. As a result, some participants suggested adding more detailed information such as details about doctors, maps, multilingual support, and expanded hospital guidance information. Both English and Greek queries are partially supported, however the system was evaluated mainly in English, and further improvements would be required before deployment in multilingual real-world environments.

In addition, the system depends on a predefined knowledge base, meaning that unsupported or missing information cannot always be handled naturally. While this helps avoid hallucinated answers, it also limits flexibility in conversations, which is clearly visible as certain follow-up questions, ambiguous wording, or inputs with typos still caused difficulties for the system [9],[29]. Speech recognition performance was also affected in some cases by microphone quality and environmental noise [32],[12].

6.4 Future Work

There are several areas where the system could be improved further in future work. One important direction would be replacing the current static knowledge base with dynamically updated information sources. This could allow the assistant to provide real-time updates related to hospital schedules, available departments, or waiting times. The conversational side of the system could also be improved with better context handling and with more flexibility in follow-up interpretations, since some participants still needed to rephrase questions during the evaluation. In addition, future versions could improve multilingual speech processing, improvements in noisy environments, and the overall retrieval for more natural conversations. For a natural experi-

ence from the embodiment perspective, there could be more advanced avatar behaviours, facial expressions, and gesture systems, while mobile or kiosk-based deployment could allow the system to be used more practically inside hospital environments. Finally, larger scale evaluations with more diverse participant groups and real hospital settings would provide a more accurate understanding of usability, reliability, and user acceptance.

6.5 Final Remarks

All in all, this thesis showed that a conversational assistant can combine speech interaction, information retrieval, and avatar embodiment into a functional hospital support system. Although the current implementation remains a prototype, the evaluation results suggest that this type of approach has strong potential for future healthcare assistance applications, especially in controlled environments, while also remaining adaptable to other domains in the future.

BIBLIOGRAPHY

[1] L. Laranjo, A. G. Dunn, H. L. Tong, A. B. Kocaballi, J. Chen, R. Bashir, D. Surian, B. Gallego, F. Magrabi, A. Y. S. Lau, and E. Coiera, "Conversational agents in healthcare: a systematic review," *Journal of the American Medical Informatics Association*, vol. 25, no. 9, pp. 1248–1258, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:51626741

[2] J. Cassell, J. Sullivan, S. Prevost, and E. F. Churchill, Eds., *Embodied Conversational Agents*. MIT Press, 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:141363580

[3] S. C. Jolibois, A. Ito, and T. Nose, "The development of an emotional embodied conversational agent and the evaluation of the effect of response delay on user impression," *Applied Sciences*, vol. 15, no. 8, p. 4256, 2025.

[4] S. T. Völkel, D. Buschek, M. Eiband, B. R. Cowan, and H. Hussmann, "Eliciting and analysing users' envisioned dialogues with perfect voice assistants," in *CHI Conference on Human Factors in Computing Systems*. ACM, 2021. [Online]. Available: https://doi.org/10.1145/3411764.3445536

[5] F.-C. Yang, P. Acevedo, S. Guo, M. Choi, and C. Mousas, "Embodied conversational agents in extended reality: A systematic review," *IEEE Access*, vol. 13, pp. 79 805–79 824, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:278347837

[6] H. Chen, X. Liu, D. Yin, and J. Tang, "A survey on dialogue systems: Recent advances and new frontiers," *Acm Sigkdd Explorations Newsletter*, vol. 19, no. 2, pp. 25–35, 2017. [Online]. Available: http://dx.doi.org/10.1145/3166054.3166058

[7] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2020, pp. 6769–6781. [Online]. Available: https://aclanthology.org/2020.emnlp-main.550/

[8] Y. Zhang, Y. Li, L. Cui, D. Cai, L. Liu, T. Fu, X. Huang, E. Zhao, Y. Zhang, C. Xu, Y. Chen, L. Wang, A. T. Luu, W. Bi, F. Shi, and S. Shi, "Siren's song in the ai ocean: A survey on hallucination in large language models," 2023. [Online]. Available: https://arxiv.org/abs/2309.01219

[9] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. K"uttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation

for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems 33*, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30*, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/7181-attention-is-all-you-need

[11] S. Schulhoff, M. Ilie, N. Balepur *et al.*, "The prompt report: A systematic survey of prompt engineering techniques," 2024. [Online]. Available: https://arxiv.org/abs/2406.06608

[12] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 2023, pp. 28 492–28 518. [Online]. Available: https://proceedings.mlr.press/v202/radford23a.html

[13] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. J. Skerry-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4779–4783.

[14] Unity Technologies, "Unity engine: 2d & 3d development platform," 2026, official product page. [Online]. Available: https://unity.com/products/unity-engine

[15] FastAPI, "Fastapi," 2026, official documentation. [Online]. Available: https://fastapi.tiangolo.com/

[16] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009. [Online]. Available: https://doi.org/10.1561/1500000019

[17] J. Wiseman, "webrtcvad," 2017, pyPI project page. [Online]. Available: https://pypi.org/project/webrtcvad/

[18] Unity Technologies, "Audiosource," 2026, unity Scripting API. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/AudioSource.html

[19] Uvicorn, "Uvicorn," 2026, official documentation. [Online]. Available: https://uvicorn.dev/

[20] SYSTRAN, "faster-whisper," 2025, gitHub repository / README. [Online]. Available: https://github.com/SYSTRAN/faster-whisper

[21] Microsoft, “What is text to speech?” 2026, microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/speech-service/text-to-speech

[22] sentence-transformers, “all-mpnet-base-v2,” 2021, hugging Face model card. [Online]. Available: https://huggingface.co/sentence-transformers/all-mpnet-base-v2

[23] Ready Player Me, “How ready player me works,” 2026, official documentation. [Online]. Available: https://sketchfab.com/readyplayerme

[24] “Oculus ovr lipsync,” Ready Player Me, 2026, official documentation. [Online]. Available: https://developers.meta.com/horizon/documentation/unity/audio-ovrlipsync-unity/

[25] State Health Services Organisation, “Nicosia general hospital,” 2026, official website. [Online]. Available: https://www.shso.org.cy/en/hospital/geniko-nosokomeio-lefkosias/

[26] Unity Technologies, “Microphone.start,” 2026, unity Scripting API. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Microphone.Start.html

[27] “Skinnedmeshrenderer.setblendshapeweight,” Unity Technologies, 2026, unity Scripting API. [Online]. Available: https://docs.unity3d.com/6000.0/Documentation/ScriptReference/SkinnedMeshRenderer.SetBlendShapeWeight.html

[28] Fab, “Hospital waiting room,” 2026. [Online]. Available: https://www.fab.com/listings/8e3cae03-337e-41cb-9262-66495b5db5d9

[29] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, “A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions,” *arXiv preprint arXiv:2311.05232*, 2023, accepted by ACM Transactions on Information Systems. [Online]. Available: https://arxiv.org/abs/2311.05232

[30] M. Krajcovic, P. Demcak, and E. Kuric, “Talking surveys: How photorealistic embodied conversational agents shape response quality, engagement, and satisfaction,” *arXiv preprint arXiv:2508.02376*, 2025. [Online]. Available: https://arxiv.org/abs/2508.02376

[31] L. O. Kroczek, A. May, S. Hettenkofer, A. Ruider, B. Ludwig, and A. Mühlberger, “The influence of persona and conversational task on social interactions with a llm-controlled embodied conversational agent,” *Computers in Human Behavior*, vol. 172, p. 108759, Nov. 2025. [Online]. Available: https://doi.org/10.1016/j.chb.2025.108759

[32] G. Krishna, C. Tran, J. Yu, and A. H. Tewfik, “Speech recognition with no speech or with noisy speech,” *arXiv preprint arXiv:1903.00739*, 2019, accepted for ICASSP 2019. [Online]. Available: https://arxiv.org/abs/1903.00739

APPENDICES

APPENDIX I

User Manual

This appendix briefly demonstrates the basic operation and usage flow of the implemented hospital information assistant prototype.

I.1 Launching the Backend

The backend server is started locally using the FastAPI application.

```
uvicorn server:app --host 127.0.0.1 --port 8000
```

Figure I.1 shows the backend during active request handling.

Figure I.1: Backend runtime execution

I.2 Main Interface

After launching the Unity application, the user is presented with the main interaction interface shown in Figure I.2.

The interface contains:

- A 3D embodied avatar,
- A chat window displaying the conversation area,
- A text input field,

- A send button for submitting text queries
- A microphone button for voice interaction

Figure I.2: Main system interface

I.3 Basic Interaction Example

Users may ask hospital-related questions either through text input or voice input. Figure I.3 shows a department location query.

I.4 Follow-Up Question Example

The system supports lightweight contextual follow-up interaction within the same session. Figure I.4 demonstrates a follow-up question regarding another department.

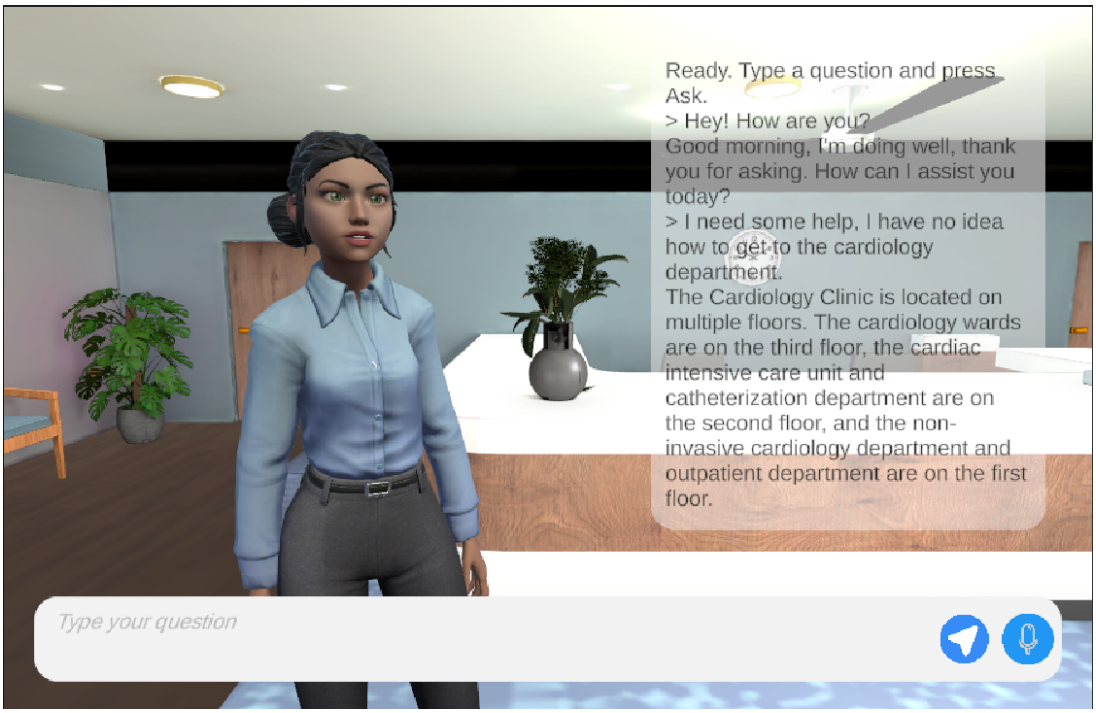


Figure I.3: Example hospital information query

Figure I.4: Follow-up interaction example

I.5 Voice Interaction

Voice interaction is performed through the microphone button located at the bottom-right corner of the interface.

The interaction process is:

1. Press the microphone button.
2. Speak clearly into the microphone.
3. Wait until recording automatically stops after silence detection.
4. The recorded audio is transmitted to the backend for transcription.
5. The recognized text is processed and becomes speech.
6. Receive the final response played through the avatar.

During recording, the interface provides visual feedback indicating that microphone capture is active.

I.6 Out-of-Scope Requests

The assistant is intentionally restricted to hospital-related information. Figure I.5 shows examples of unsupported or unrelated requests.

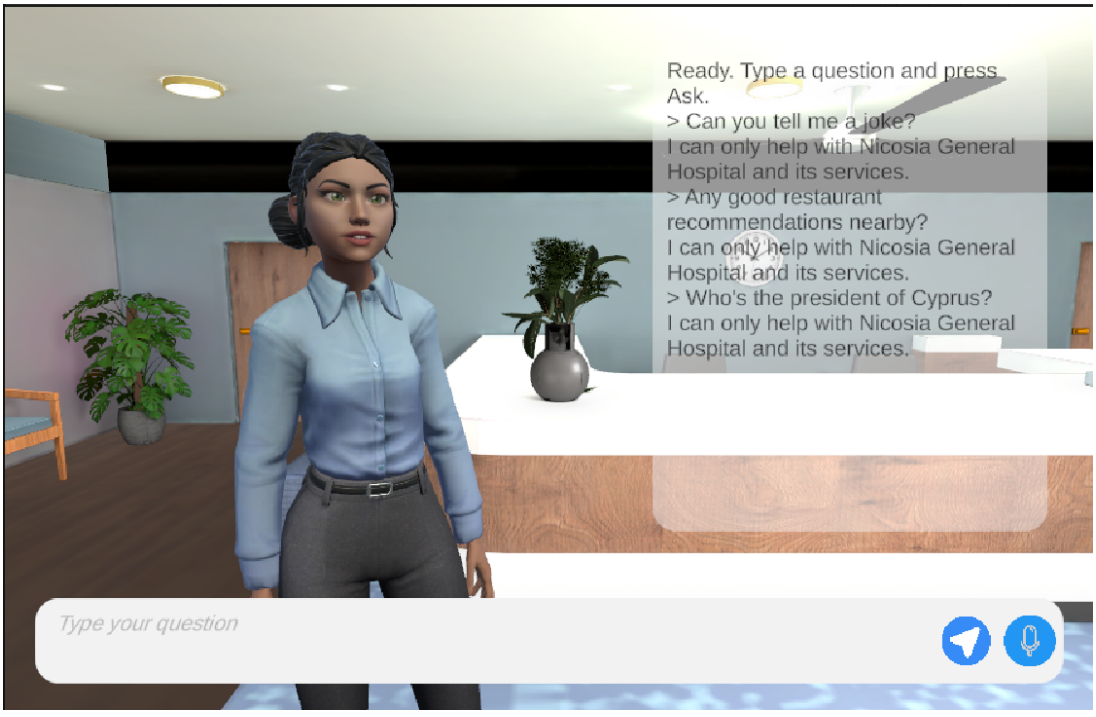


Figure I.5: Out-of-scope request handling

I.7 Avatar Speech Animation

During speech playback, the avatar performs automatic lip synchronization driven by the generated audio stream.

Figure I.6: Example avatar lip synchronization states

APPENDIX II

Technical Setup and Execution Manual

This appendix provides the technical steps required to run the implemented prototype locally. The setup reflects the final implementation used during development and evaluation.

II.1 Development Environment

Table II.1: Main development and execution tools

Component	**Tool / Version**
Unity Editor	Unity 2022.3.62f2
Backend Language	Python 3.11
Backend Framework	FastAPI
Backend Server	Uvicorn
Frontend Logic	C# scripts in Unity
Speech-to-Text	Faster-Whisper
Text-to-Speech	Edge Neural TTS
Retrieval Model	sentence-transformers/all-mpnet-base-v2
Development Tools	PyCharm, Unity Editor, terminal

II.2 Backend Setup

The backend is located inside the `ml` directory of the project. The following commands are used to create the Python environment, install dependencies, and start the FastAPI server.

```
cd AvatarChatbotDemo/ml

python -m venv .venv

.venv\Scripts\activate

pip install -r requirements.txt

uvicorn server:app --host 127.0.0.1 --port 8000 --reload
```

After startup, the backend listens locally at:

`http://127.0.0.1:8000`

The backend must remain running while the Unity application is being tested.

II.3 Required Runtime Files

The backend depends on precomputed runtime files generated from the canonical hospital knowledge base. These files are loaded when the backend starts.

Table II.2: Required backend runtime files

File	Purpose
`kb_canonical.json`	Canonical hospital knowledge base constructed from official hospital information.
`answer_bank.json`	Stores canonical answers used by the retrieval system.
`faq_items.json`	Stores generated query variations linked to answer identifiers.
`bm25_stats.json`	Stores BM25 lexical retrieval statistics.
`q_emb.npy`	Stores precomputed dense query embeddings.
`department_aliases.json`	Maps department aliases and multilingual names to internal department identifiers.
`model_name.txt`	Stores the sentence-transformer model used for embedding generation.

If the canonical knowledge base is modified, the index build script must be executed again before running the backend.

II.4 Index Build Command

The retrieval artifacts are generated offline using the index build script.

```
cd AvatarChatbotDemo/ml

.venv\Scripts\activate

python build_index.py
```

This process rebuilds the answer bank, FAQ items, BM25 statistics, department aliases, and embedding matrix.

II.5 Unity Frontend Setup

The Unity frontend is opened from the main project folder using Unity Hub.

1. Open `AvatarChatbotDemo` in Unity Hub.
2. Use Unity Editor version `2022.3.62f2`.
3. Open the main interaction scene.
4. Confirm that the backend URL points to:

`http://127.0.0.1:8000`

5. Start the backend server.
6. Press Play in the Unity Editor.
7. Submit a text question or use the microphone button.

II.6 Backend Endpoints

Table II.3: Backend API endpoints

Endpoint	**Purpose**
`/health`	Checks whether the backend server is running.
`/ask`	Receives a text query, performs routing and retrieval, and returns an answer.
`/stt`	Receives audio input and returns the transcribed text.
`/tts`	Receives answer text and returns synthesized audio for Unity playback.

The health endpoint can be tested in a browser:

`http://127.0.0.1:8000/health`

II.7 Example Text Query Test

The `/ask` endpoint can be tested independently before using Unity.

```
POST http://127.0.0.1:8000/ask
Content-Type: application/json

{
  "question": "Where is Cardiology located?",
  "session_id": "default"
}
```

A successful response confirms that the backend, routing module, retrieval engine, and answer selection pipeline are operating correctly.

II.8 External Provider Configuration

External provider credentials are stored locally through environment variables or a local `.env` file. These keys are required only for provider-based LLM routing, rephrasing, or social response handling.

```
GROQ_API_KEY=...
OPENROUTER_API_KEY=...
```

API keys are excluded from the submitted thesis and source code for security reasons.

II.9 Execution Workflow

The complete local execution workflow is:

1. Start the backend using Uvicorn.
2. Confirm that the backend is reachable through `/health`.
3. Open the Unity project.
4. Run the main scene.
5. Ask a question using text input or microphone input.
6. Unity sends the request to the backend.
7. The backend performs transcription, routing, retrieval, and response generation.
8. Unity receives the response and plays the speech through the avatar.

APPENDIX III

Evaluation Questionnaire

This appendix presents the questionnaire that was used during the evaluation of the Lana hospital assistant prototype. The questionnaire was distributed to participants after completing the predefined interaction scenarios described in Chapter 5. The purpose of the questionnaire was to collect both quantitative and qualitative feedback regarding usability, conversational quality, speech interaction, avatar embodiment, usefulness, and overall system reliability. A total of 18 participants completed the evaluation.

III.1 Questionnaire Structure

The questionnaire consisted of three main sections:

1. Participant background information
2. Likert-scale evaluation questions
3. Open-ended feedback questions

The quantitative evaluation used a five-point Likert scale:

- Strongly Disagree
- Disagree
- Neutral
- Agree
- Strongly Agree

III.2 Participant Background Questions

Participants were initially asked several background questions related to demographics and familiarity with conversational systems.

- Gender
- Age group
- Previous use of conversational assistants
- Comfort level with voice interaction systems

- Previous experience with healthcare-related assistants

III.3 System Evaluation Questions

The main evaluation section included questions covering usability, conversational quality, avatar interaction, usefulness, and reliability.

Table III.1: Main questionnaire evaluation categories

Category	**Example Evaluation Focus**
Usability	Ease of use and interaction flow
Speech Interaction	Speech recognition and speaking preference
Conversational Quality	Response clarity and naturalness
Avatar Embodiment	Engagement and lip synchronization
Usefulness	Helpfulness for hospital guidance
Reliability	Trustworthiness and role consistency

III.4 Open-Ended Feedback

Participants were also invited to provide optional written feedback regarding:

- Positive aspects of the system
- Difficulties experienced during interaction
- Suggested improvements

The collected responses were analyzed and summarized in Chapter 5.

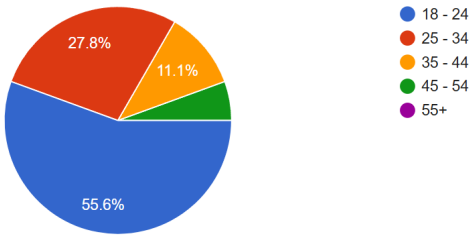
III.5 Questionnaire Screenshots

Figure III.1 presents example sections from the evaluation questionnaire used during the study.

A1. Age group

Copy chart

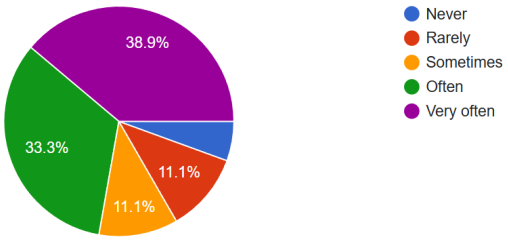
18 responses



A3. How often do you use voice assistants or chatbots?

Copy chart

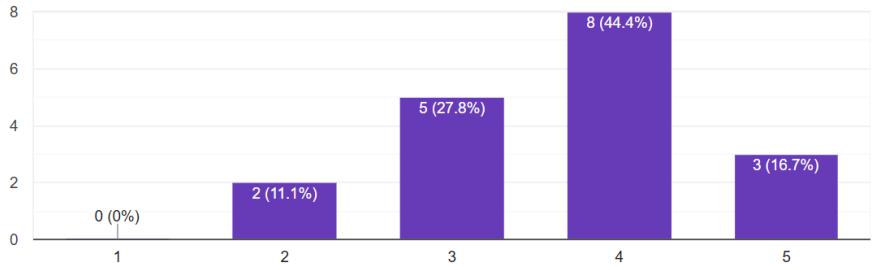
18 responses



2. Speaking to the system felt easier than typing would have been.

Copy chart

18 responses



20. I felt the system remained within its role as a hospital information assistant.

Copy chart

18 responses

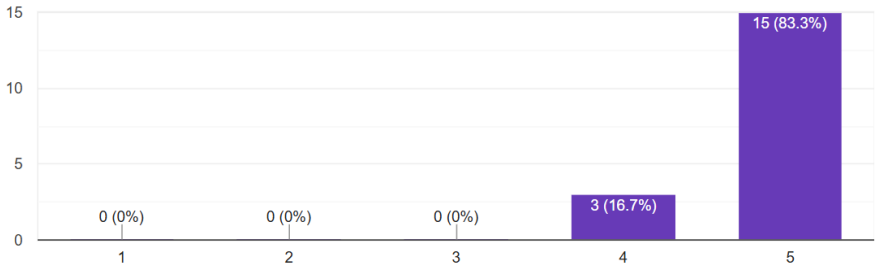


Figure III.1: Example sections from the evaluation questionnaire

APPENDIX IV

Supported Knowledge Scope and Coverage

This appendix summarizes the operational scope of the implemented hospital information assistant prototype, including supported hospital departments, supported query categories, multilingual handling, and retrieval behaviour.

IV.1 Supported Hospital Departments

The current prototype supports information retrieval for multiple departments and clinical units derived from the official Nicosia General Hospital website.

Clinical	**Surgical**	**Diagnostic**	**Specialized Units**
Cardiology	General Surgery	Radiology	ICU
Neurology	Neurosurgery	Microbiology	Emergency Department
Pulmonology	Orthopedic Clinic	Pathology	Oncology – Radiology Therapy
Dermatology	Plastic Surgery	Clinical Biochemistry	Breast Centre
Gastroenterology	Urology	Medical Physics	Trauma Centre
Hematology	Vascular Surgery	Cytopathology	Transplant Centre
Rheumatology	Cardiothoracic Surgery	Nuclear Medicine	Diabetes Centre
Nephrology	ENT	Hematology Laboratory	Pain Clinic
Internal Medicine	Maxillofacial Surgery	Immunology	Physiotherapy Department
Cardiometabolic Clinic	Endocrine Surgery		Smoking Cessation Clinic

Table IV.1: Examples of supported hospital departments and units

IV.2 Supported Query Categories

The system supports multiple categories of hospital-related informational requests.

Query Type	Example
Department overview	“What is Cardiology?”
Location queries	“Where is Neurology located?”
Contact information	“What is the Oncology phone number?”
Appointment information	“How do I book an appointment?”
Department services	“What services does ICU provide?”
Follow-up interaction	“What about Neurology?”
Multilingual queries	“Πού βρίσκεται η Καρδιολογική;”
Out-of-scope handling	“Who won the World Cup?”

Table IV.2: Supported query categories

IV.3 Knowledge Base Characteristics

The retrieval dataset was generated from structured hospital information extracted from the official hospital website.

Characteristic	Value
Indexed QA pairs	2491
Canonical answer entries	765
Supported departments/entities	43
Supported languages	English and Greek
Retrieval method	Hybrid dense + BM25 retrieval
Embedding model	all-mpnet-base-v2

Table IV.3: Knowledge base statistics

IV.4 Multilingual Alias Handling

The system supports multilingual aliases and alternative department references during retrieval.

Canonical Department	Supported Aliases
Emergency Department	A&E, Emergency, First Aid, ΤΑΕΠ
Cardiology	Cardiology Clinic, Καρδιολογική
ICU	Intensive Care Unit, ΜΕΘ
Oncology	Oncology - Radiology Therapy, Ογκολογία
ENT	Otolaryngology, ENT, ΩΡΛ

Table IV.4: Examples of multilingual alias normalization

IV.5 Representative System Behaviour

The following examples summarize representative behaviour observed during testing.

IV.5.1 Follow-Up Query Resolution

User: “Can you give me the Oncology contact information?”

User: “And the same for Neurology?”

The routing pipeline preserves conversational context and resolves the second query as a follow-up request for Neurology contact information.

IV.5.2 Out-of-Scope Restriction

User: “Who is the president of Cyprus?”

The assistant rejects unrelated requests and remains restricted to hospital-related information only.

IV.5.3 Typo Handling

User: “Where is cardiolgy?”

The retrieval pipeline successfully resolves the intended department despite the spelling error.