BIRMINGHAM CITY
University

CMP6200
Individual Undergraduate Project
2024 – 2025

University Artificially Intelligent Assistant

Course: Computer & Data Science
Student Name: Lewis Higgins
Student Number: 22133848
Supervisor Name: Dr. Atif Azad

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et erat consectetur, scelerisque eros nec, ultricies est. Donec mi ipsum, imperdiet vitae arcu quis, luctus venenatis quam. Integer ac massa a augue venenatis fringilla. Etiam posuere libero sed nulla tristique volutpat.

Acknowledgements

I would like to acknowledge. . .

Contents

Glossary

Term	Definition
RAG	Retrieval-Augmented Generation is...

List of Figures

List of Tables

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et erat consectetur, scelerisque eros nec, ultricies est. Donec mi ipsum, imperdiet vitae arcu quis, luctus venenatis quam. Integer ac massa a augue venenatis fringilla. Etiam posuere libero sed nulla tristique volutpat.

1.1 Problem definition

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et erat consectetur, scelerisque eros nec, ultricies est. Donec mi ipsum, imperdiet vitae arcu quis, luctus venenatis quam. Integer ac massa a augue venenatis fringilla. Etiam posuere libero sed nulla tristique volutpat.

1.2 Scope

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et erat consectetur, scelerisque eros nec, ultricies est. Donec mi ipsum, imperdiet vitae arcu quis, luctus venenatis quam. Integer ac massa a augue venenatis fringilla. Etiam posuere libero sed nulla tristique volutpat.

1.3 Rationale

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et erat consectetur, scelerisque eros nec, ultricies est. Donec mi ipsum, imperdiet vitae arcu quis, luctus venenatis quam. Integer ac massa a augue venenatis fringilla. Etiam posuere libero sed nulla tristique volutpat.

1.4 Aims and Objectives

This project aims to aid new and existing students alike while they are attending university with helpful information about university itself, such as university societies, locations/campuses, and policies through the medium of a digital chatbot companion to converse with. The project's objectives are:

- Conduct a thorough literature review on the surrounding topics, namely AI, LLMs and NLP.
- Create effective documentation for all stages of development, highlighting challenges faced during the process.
- Leverage Retrieval-Augmented Generation alongside a cloud-based LLM to query a vector database of university-related data.
- Develop a chatbot capable of accurately answering user queries related to university buildings, policies, and societies with a minimum 80% accuracy rate.
- Evaluate the effectiveness of an AI assistant on university student acclimatization.

1.5 Background information

Possibly unnecessary.

Literature Review

2.1 Review of Literature

2.1.1 Artificial Intelligence (AI)

Researchers have always wanted to harness the processing power of computers to act in a manner indistinguishable from that of humans from as long ago as 1950, where the question was posed 'Can machines think?' (Turing, 1950). Ever since, constant innovations were made in computer intelligence and machine learning, from playing games of checkers at a better level than human players (Samuel, 1959) to classifying the contents of millions of images using convolutional neural networks (Krizhevsky, Sutskever and Hinton, 2012).

Recently, AI is used across many disciplines for different purposes to complete tasks faster than, and in some cases better than, human workers, especially with the introduction of large language models (LLMs) (Maedche et al., 2019). Wirtz et al. (2018) write that 'service robots' [1] can complete a variety of tangible or intangible actions, such as two-way conversation with chatbots.

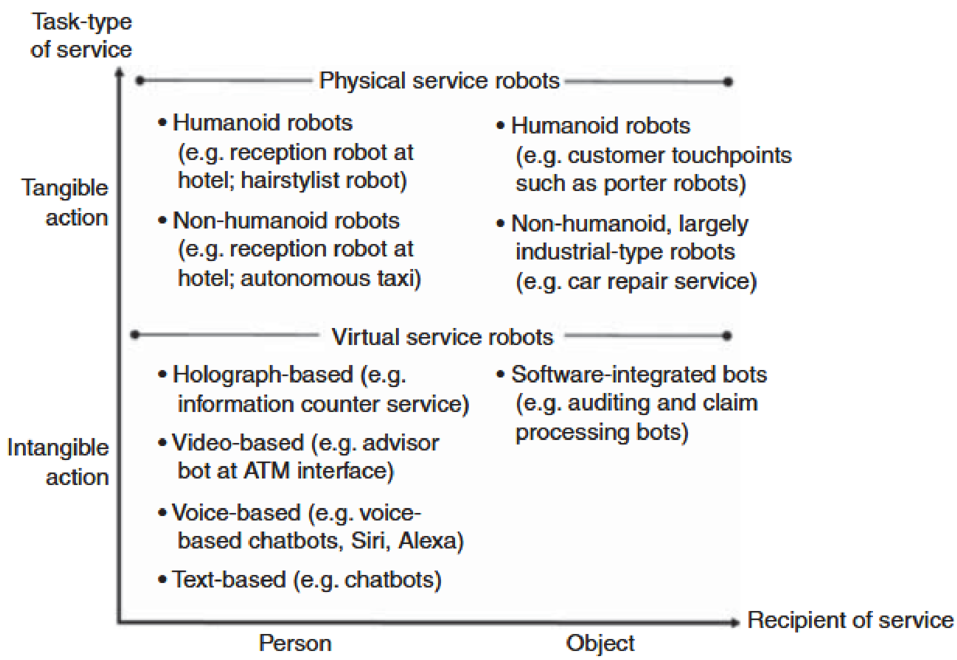

Figure 2.1: Service robots categorization by task-type and recipient of service (Wirtz et al., 2018).

When developing an AI project, it is important that the development process is ethical and human-centred, which is known as Human-Centred AI (HCAI). Another issue is the "black-box problem" - the inability to know an AI's reasoning, meaning that eXplainable AI (XAI) is a growing necessity (Miró-Nicolau, Jaume-i-Capó and Moyà-Alcover, 2025).

[1] Defined as "system-based autonomous and adaptable interfaces that interact, communicate and deliver service to an organization's customers" (Wirtz et al., 2018, p.909)

Focusing on HCAI and XAI means the focus shifts from the machine to the user and their experience using the AI. Shneiderman (2020) strongly advocates for the promotion of HCAI for the benefit of both companies and their users, which is a commonly accepted idea due to the ethical risks of using AI.

Because AI calculates outcomes from its training data rather than understanding social norms and perspectives, using it in sociotechnical systems poses serious risks due to the 'traps' it can fall into, because it cannot account for every possibility such as the personal tendencies and biases of its users (Selbst et al., 2019), and therefore developers require a shift in focus - from the final product to the development process itself and end users, which also echoes Shneiderman's views.

2.1.2 Natural language processing (NLP)

The ability for a computer to interpret and understand human language greatly enhances the scale of their capabilities. This was recognised during the 1950s, where machine translation from Russian to English was demonstrated for the first time, albeit in a basic form (Jones, 1994). Ever since, NLP has been a key topic in computing, especially in recent years, with its applications widening in scope with modern processing power.

One of the key advancements in NLP is vectorisation, a process where data is embedded into a numerical equivalent that a computer can interpret, enabling Natural Language Understanding (NLU) and the identification of semantic similarities between words through the use of an embedding model like Word2Vec (Mikolov et al., 2013) without the need to manually label data. Word2Vec was a key innovation in NLP, and Mikolov and Le went on to improve it further with Doc2Vec (Le and Mikolov, 2014), which could embed entire documents into semantically searchable vectorised forms.

Embedding models have further improved since, most notably with Vaswani et al. (2017)'s Transformer architecture enhancing models such as BERT (Devlin et al., 2019), which establishes context through analysing multiple neighbours of a word rather than reading from left to right, gaining a higher understanding of the text it processes. Many embedding models have since been developed, though one of the most reputable is OpenAI's recent text-embeddings-3 model (OpenAI, 2024c), which can be used in the development of the chatbot at a low cost.

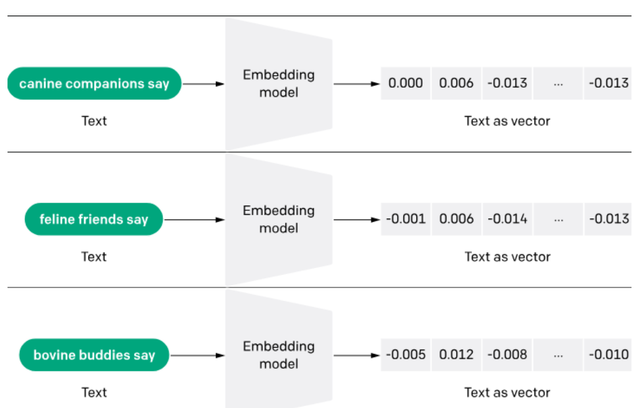

Figure 2.2: A basic overview of vectorisation (OpenAI, 2024a).

2.1.3 Large language models

LLMs are colossal machine learning models that leverage NLP to generate text, and have become widely used across industries in place of technical support and human resources (Vrontis et al., 2022). The training data required for an LLM is immense, reaching 45 terabytes of text data for ChatGPT in 2023 (Dwivedi et al., 2023).

This data is harvested from websites and social media due to them being the largest repositories of opinionated text data (Dubey et al. (2024), Z. Wang et al. (2016)). However, meticulous care is taken into the specific sources used to remove Personally Identifiable Information (PII) to minimise privacy and ethical concerns (Dubey et al., 2024).

The previously mentioned Transformer by Vaswani et al. (2017) became a staple in LLMs due to the major reduction in necessary processing power to produce higher-quality results, and it continues to underpin many LLMs today, including ChatGPT (Brown et al., 2020). Even with these enhancements, LLMs are still extremely performance intensive, requiring more than 8 top-range server-grade GPUs to run some of the most powerful high-parameter models like LLaMA 3.1's 405 billion parameter model (Dubey et al., 2024), and many therefore use cloud API solutions to access LLMs.

The amount of parameters in a model does not entirely account for the quality of its responses, as studied by Ouyang et al. (2022) in Figure 2.3 wherein their surveys revealed their fine-tuned LLM "InstructGPT" with over 100x less parameters than a 175 billion parameter GPT3 model would often give answers preferred by its human assessors, which reveals that the fine-tuning and prompt engineering of an LLM is as vitally important to the quality of its responses as the amount of parameters.

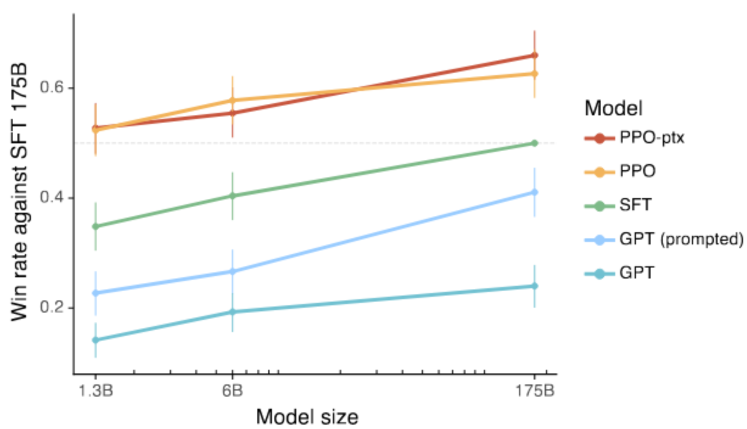

Figure 2.3: Human evaluations of the GPT models produced by Ouyang et al. (2022). PPO and PPO-ptx are their models.

The simplest way to measure the accuracy and quality of an LLM's responses is through human evaluation surveys such as that conducted by Ouyang et al. (2022), though software approaches such as DeepEval can be used. DeepEval offers 14 metrics to test LLM outputs with (DeepEval, 2024), with a notable metric being "G-Eval", originally introduced by Liu et al. (2023), which uses an "LLM-as-a-judge" approach where an LLM will evaluate and grade the quality of the output.

2.1.4 Retrieval-Augmented Generation

While LLMs are highly useful tools across many industries, they are not without limitations. The most notable of these limitations are hallucinations (P. Lewis et al., 2021), where the LLM will fabricate information that conflicts with user input, earlier conversation context or true facts (Zhang et al., 2023). This occurs as a direct result of the LLM's parametric memory[2] being overfitted or biased, which can be counteracted through introducing an external knowledge source, known as non-parametric memory (Komeili, Shuster and Weston (2022), Siriwardhana et al. (2023)).

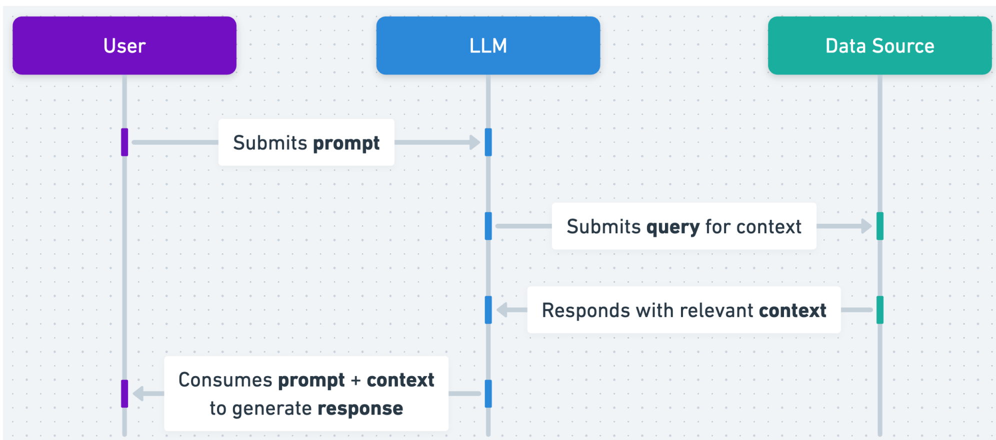

Figure 2.4: A basic overview of a RAG workflow (OpenAI, 2024b).

Siriwardhana et al. (2023) expanded upon the earlier works of Karpukhin et al. (2020) and M. Lewis et al. (2020) by creating "RAG-end2end", which explored the capabilities of RAG on a dynamically updating knowledge store, meaning the LLM itself would not have to be retrained every time the data updates, saving enormous amounts of processing power.

RAG is dependent upon external knowledge stores such as vector databases, which store and process vectorised data for non-parametric memory (Li, 2023), which makes them an essential part of the backend of a RAG-enabled chatbot as studied by Odede and Frommholz (2024).

Many software options exist for vector databases, such as Milvus (J. Wang et al., 2021), Pinecone (Pinecone, 2024), Chroma (Chroma, 2024). Xie et al. (2023) compared these three, citing Pinecone's 'robust distributed computing capabilities and scalability', and its common usage in real-time searching scenarios. Pinecone was also used in chatbots by Odede and Frommholz (2024) and Singer et al. (2024), showcasing its potential as a vector database solution for chatbots.

However, another open-source option with proven capabilities is FAISS, which was designed by engineers at Facebook (now Meta) which can be up to 8.5x faster than alternative options as written by Johnson, Douze and Jégou (2017). The speed and open-source nature of FAISS are very desirable in real-time applications such as chatbots, with FAISS also supporting direct integration with LLM development frameworks such as LangChain.

[2]Knowledge that the LLM has from its training data (Siriwardhana et al., 2023).

LangChain (LangChain, 2024) is a popular open-source framework for LLM development, and RAG pipelines by extension. that can be used to connect backend elements together, as described by Singer et al. (2024) when they used it to chunk their text data and connect to their vector database to store their embedded data.

2.1.5 Agentic RAG

A very recent development in the LLM space is the use of "agents". Agents increase the capabilities of LLMs by giving them access to tools created by developers, effectively allowing the LLM to execute its own code to perform tasks such as web searching and data retrieval. Agents can also evaluate themselves, as demonstrated in Figures 2.5 and 2.6, wherein the LLM will execute an action based on the query and evaluate the results. If the results are unsatisfactory, it can perform a slightly different action until a suitable answer is found. In a RAG context, this would often refer to continuous optimisation of the semantic search query used on the vector database.

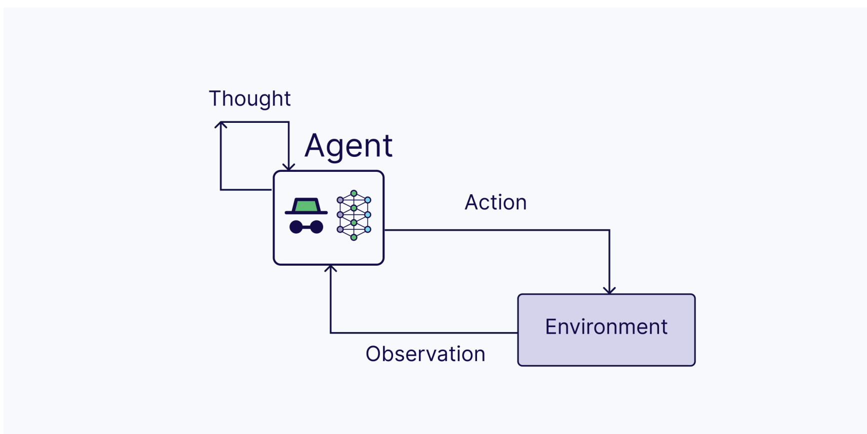

Figure 2.5: A basic ReAct (Reason + Act) agent workflow (Weaviate, 2024).

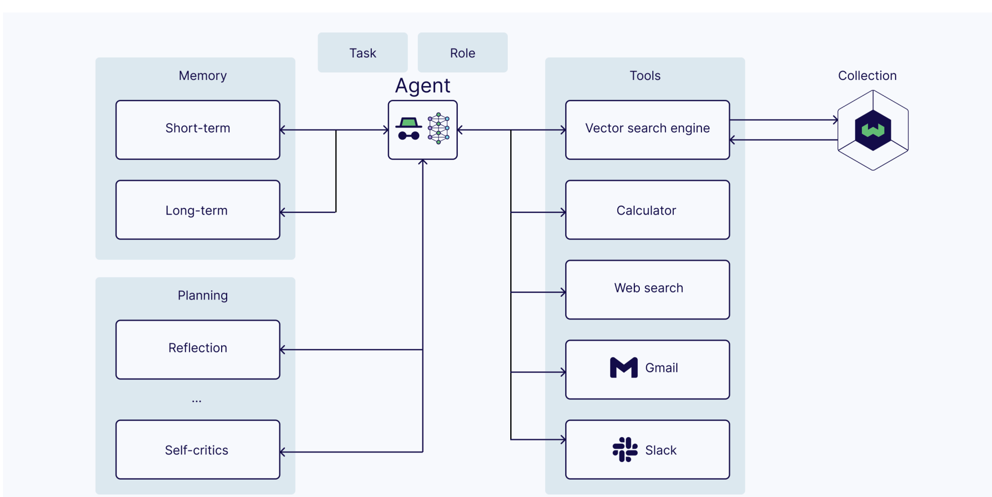

Figure 2.6: An advanced example agent workflow (Weaviate, 2024).

Figure 2.6 demonstrates the ability for agents to leverage multiple tools not only limited to searching a vector store, and also showcases their reflective and self-evaluative capabilities. With an agent that uses an architecture like this (known as Corrective RAG/ CRAG), answers would be extensively evaluated and regenerated until the agent deems them a suitable answer to the user's query. While this largely increases the time taken to generate results, it ensures those results will be accurate and useful to the end user.

In academic works, Woo et al. (2025) explored the implementation of augmenting base LLMs with agentic retrieval capabilities in a RAG workflow, which enhanced the accuracy of a GPT4 LLM by 95% on their medical Q&A dataset.

M. Bran et al. (2024)'s works were among the best reviewed in demonstrating the capabilities of Agentic AI, with their model they named ChemCrow having the ability to call a massive variety of tools including web search and even accessing advanced chemistry equipment to formulate chemical catalysts from a singular natural language prompt.

2.1.6 Chatbots / Conversational Agents

Conversational agents, better known as chatbots, leverage NLP in order to simulate a conversational flow between a user and machine, and have become mainstream products in recent years (Liao et al., 2018), though have existed as far back as 1966 with the creation of "ELIZA" for the IBM-7094 (Weizenbaum, 1966). As time has passed, advancements in chatbots have occurred in "waves", where each new wave has brought a major innovation (Schöbel et al., 2024).

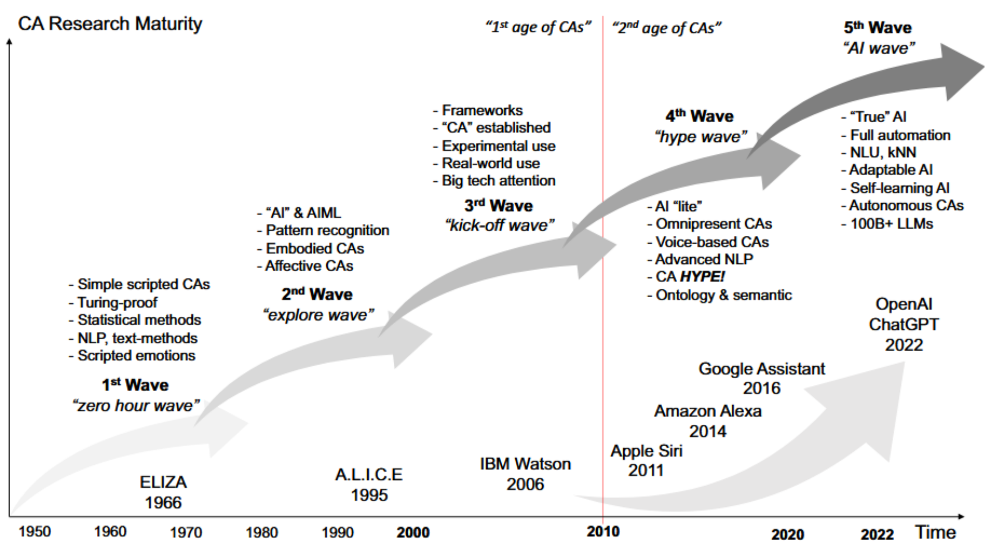

Figure 2.7: The five waves of conversational agent research (Schöbel et al., 2024).

Due to these considerable developments in the field, chatbots are now widely used across industries such as education (Kuhail et al., 2023). However, the use of the latest wave of chatbots based on LLMs poses significant risks, especially in educational settings as studied by Neumann et al. (2024), due to the risk of hallucinations being interpreted as absolute fact, although Shuster et al. (2021) argued that this risk can be greatly reduced through introducing RAG to the backend LLM, which is further backed by the RAG-based chatbot created by Ge et al. (2023), which they found to also give superior answers to those of a general-purpose chatbot without RAG.

Many platforms exist to aid chatbot development, though they are typically aimed at users from non-IT backgrounds (Srivastava and Prabhakar, 2020). Popular platforms include IBM's watsonx Assistant (IBM, 2024a), Google's Dialogflow (Google, 2024) and Microsoft's Bot Framework (Microsoft, 2024). However, these are primarily targeted at enterprise clients which is reflected in their pricing. Instead of using these, the chatbot can be manually developed using LangChain as its framework.

2.1.7 User experience and Human-Computer Interaction

The way people interact with their devices has drastically evolved over time, from early MS-DOS command-line interfaces (CLIs) to mouse-based graphical user interfaces (GUIs), to touch screens (Kotian et al., 2024), greatly broadening the userbase of computers worldwide. Therefore, inclusive and accessible design is increasingly important to maximise the audience of any software, especially considering the growing disabled population (Putnam et al., 2012).

As well as being inclusive, the design should also be user-centred, meaning it should be an iterative process that is constantly taking user feedback into account (Chammas, Quaresma and Mont'Alvão, 2015). However, there are some barriers in this process when developing chatbots, as studied by Clark et al. (2019) in their survey of university students who stated that they view chatbots as tools, and would not converse with them in the same way as they would a person, which would limit their potential use and hinder the overall design process.

Users also often struggle to get chatbots to respond how they want, as their prompts may be poorly understood due to issues like overgeneralisation (Zamfirescu-Pereira et al., 2023), and studies show that they grow impatient after around 2 to 6 failed attempts, often branding the product as poor if this occurs (Luger and Sellen, 2016).

2.2 Summary

In conclusion, this literature review has revealed multiple key focus areas for the chatbot's development. The overall design of the chatbot must be iterative and human-centred, and user feedback should be obtained at every possible opportunity to ensure the resultant product is high quality.

A deep exploration into AI, specifically in its applications in NLP, LLMs and RAG, has revealed that the best approach will be to leverage a pre-existing cloud-based LLM, such as GPT-4o-mini, via an API, as running an LLM on a local machine would require an infeasible amount of processing power.

The non-parametric memory accessed through RAG would be a vector database created with Pinecone storing embeddings generated by OpenAI's text-embeddings-3-small model, and the overall framework will be LangChain. This will keep the cost of the project low while maintaining a tolerable level of quality in the bot's responses.

Methods and Implementation

This chapter focuses on the experimental design and implementation of the artefact, covering the self-imposed project management methodology, original concept design and the overall development process.

3.1 Methodology

When developing software, there are a wide variety of available options to manage the development process, which help to structure how time should be allocated as development progresses.

3.1.1 Waterfall

The first methodology considered was the Waterfall methodology, which is a very common approach to software development being sometimes referred to as the Software Development Life Cycle, or SDLC (Adobe, 2023). Waterfall is a highly structured and strict methodology which enforces that one stage of development must be completed before the next can begin, which creates a cascading set of steps, hence its namesake.

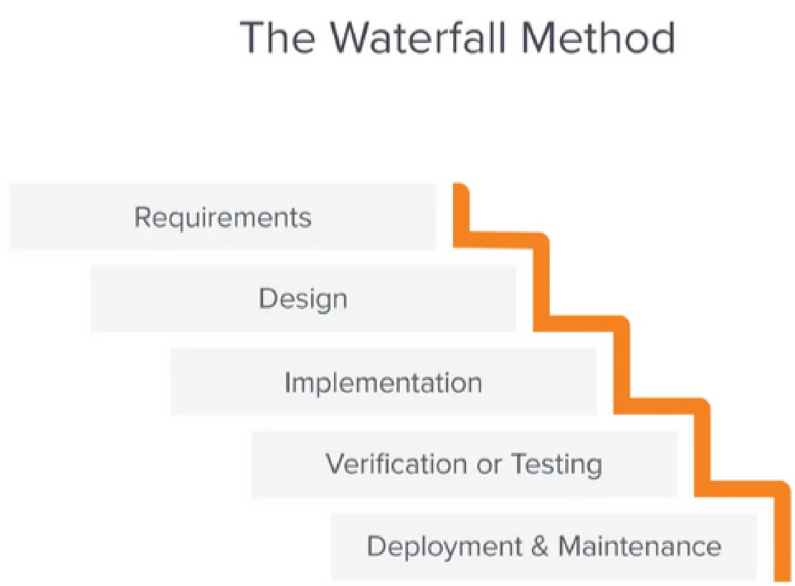

Figure 3.1: An overview of a Waterfall workflow (Adobe, 2025).

Waterfall begins by ascertaining all project requirements for all stages of the project, which would include costs, risks, associated dependencies and overall timelines for completions of each stage. Following this is the design stage, where a general high-level design is created to demonstrate the project, and this design is then acted upon and implemented in the implementation stage. Then, the implementation is rigorously tested before its eventual deployment.

It is a methodology with a strong reputation due to its clear structure, with all necessary facts and figures being calculated in the requirements stage before any designs or development occur. The clear structure allows progress to be easily measured against each predefined milestone.

Though, despite these advantages, Waterfall brings with it some clear disadvantages - the first of which being that with all requirements being defined at the very beginning of the project's development, it introduces significant difficulty should there be any further requirements specified during development. This would also bring in the second disadvantage known as 'deadline creep' (Adobe, 2025); if one stage is delayed, such as by request for additional features, this would then impact all subsequent stages.

3.1.2 Agile

The second methodology considered was another highly reputed software development methodology known as Agile. Unlike Waterfall which defines all stages and requirements at the beginning, Agile is a highly iterative methodology with steps known as 'sprints' which are frequently repeated, providing a more incremental approach to development. Each of these sprints would represent a small part of the program, eventually building up to the full version.

As depicted in Figure 3.2, Agile sprints begin by planning the overall aims of that particular sprint. Similarly to Waterfall, a high-level design is then created and developed, before being rigorously tested. This is also one of Agile's key benefits; the constant testing of the small parts developed in each sprint helps ensure that all bugs can be rectified, unlike Waterfall where the whole product is tested and some smaller elements with bugs could potentially be overlooked. After testing, the product of that sprint is deployed and reviewed. Then, the cycle begins anew with another sprint.

Figure 3.2: An overview of an Agile sprint (Asana, 2025)

The most prominent key benefit of Agile is its sprint-based iterative nature that allows for requirements to shift throughout development without major disruption. Furthermore, this incremental process minimises the risk of total project failure as usable components are constantly produced. In business environments, Agile also allows for enhanced teamwork, though this will not be present in this particular project.

As with Waterfall, Agile is not without drawbacks. Agile's most notable drawback is known as 'scope creep' (Malsam, 2024), which occurs when requirements are continually added to a point where development can never truly end; the product continues to expand far beyond its original intentions to the point where maintenance becomes extremely difficult or outright impossible with an ever-expanding codebase. Furthermore, it is possible that because of this, the end product can be almost entirely different to its original concept.

3.1.3 Comparison and decision

Both methodologies bear strong benefits and drawbacks. The particular choice for this project is Agile, primarily because of the reduced risk through constant testing and also for its deeply flexible nature allowing the requirements of the project to potentially shift over time as needed, unlike Waterfall where this could cause major deadline creep. Additionally, the time-sensitive nature of this project best suits Agile's fast incremental sprints rather than the slower, more methodical Waterfall.

3.2 Potential limitations

The project as a whole bears some limitations of its own that may hinder the development process or the final product.

3.2.1 Time

The project is likely to take a considerable amount of time to develop to the excellent standard desired. This poses an issue in balancing time throughout the academic year alongside four other modules each with their own independent deadlines and workloads of similar scale. As such, it is possible that if the product has issues, they could have been remedied with additional development time.

3.2.2 Cost

Due to the inability to use OpenAI's LLMs on a local device because of both their proprietary nature and extreme hardware requirements, their public API will need to be used instead. This incurs a financial cost for every query sent and response received from the LLM, dependent on the model chosen. For example, GPT-4o has a cost of $2.50 per 1,000,000 tokens. Throughout the development and testing processes in each sprint, a cost will slowly begin to accrue.

3.2.3 Experience

Personally, I have never worked with LLM APIs before, nor the frameworks used to create apps with them such as LangChain. As such, it is highly likely that many issues will be faced during the development process as I am forced to learn a tech stack that is completely new to me. This also links back to the previously mentioned time constraint, with the time taken to learn the modules used being time that could have been spent on development had I known them ahead of time.

3.2.4 Independence

This project is a solo venture with no support from others. As such, the previously mentioned issues of time and cost are entirely my own burden and responsibility.

3.2.5 LLM Unpredictability

LLMs are an extremely useful tool, being able to execute instructions given to them in natural language. However, without specific tuning, an LLM will not give the same response to the same prompt every time it is given. While this does add a sense of personality which could aid with a chatbot, it may risk answering questions incorrectly. This also can make LLM-based programs extremely challenging to debug due to this lack of reproducibility.

3.2.6 LangChain Documentation

LangChain will be a critical element in this project's development, serving as the backend framework that the chatbot will run on. Therefore, it is mandatory that I learn about it in order to produce a functional product, which would typically involve reading the documentation as is common when learning new modules. However, LangChain's documentation is frequently outdated and/or references functions or classes that have since been deprecated, without the documentation being updated. LangChain also frequently deprecates classes and functions with each new update, meaning that finding what currently works is a challenge in itself.

3.3 Design

Before any design concepts can be created, it is first necessary to establish what is being designed. Therefore, the functional and non-functional requirements for the chatbot were considered.

3.3.1 Requirements

Functional Requirements

The following requirements are deemed essential to the chatbot's function, and the project cannot be considered complete unless they are fulfilled:

- The chatbot must interpret and respond to answers in English.
- The chatbot must accept text queries.
- The chatbot must respond using text.
- The chatbot must be accessible at all times.
- The chatbot must supply BCU-related information.
- The chatbot must answer at least 75% of BCU-related queries correctly.

- The chatbot must have a GUI for ease of use and accessibility.
- Multiple users must be able to use the chatbot at the same time.

Non-functional Requirements

The following requirements, while not essential, would be beneficial if fulfilled:

- The chatbot should respond to queries within 10 seconds.
- The chatbot could allow for voice input and output.
- The chatbot could be deployed on an existing messaging service such as Teams.

3.3.2 Conceptual flowchart

The flowchart presented in Figure XXX depicts a model interaction from the user's perspective.

> **Not present in draft**
>
> Time constraints on this draft meant I simply don't have time to make this diagram.

In the interest of saving costs and reducing response times, the chatbot will ideally not query its university information vector store unless it cannot answer a question without it. This is because appending the university information, even in small amounts, would greatly increase the token usage of each individual prompt. This decision functionality would be provided by LangGraph, and is detailed further in Section 3.4.3.

3.3.3 Wireframes

The wireframe presented in Figure XXX depicts an early concept of how the chatbot's GUI could look like.

> **Not present in draft**
>
> Time constraints on this draft meant I simply don't have time to make this diagram.

Users will have a clearly labelled text input box, and a messaging interface similar to other text messaging apps which they would hopefully be familiar with allowing for them to quickly understand how to interact with the chatbot.

3.4 Implementation

3.4.1 Software requirements

The project was developed using Visual Studio Code as a development environment and Python as the programming language. Both of these are available freely with no limitation for academic use.

Many Python modules were used, with the UV package manager (Astral, 2025) being used for their installation due to its own high speed and ease of use. The version of Python used was 3.10.16 to ensure compatibility with the wide variety of modules used, which are detailed in Table 3.1.

Module(s)	Purpose
langchain	The framework used to handle LLM interactions, as well as embedding documents and user queries.
langchain-community	Provides additional helper classes and functions to assist development.
langchain-openai openai	Provides the functions used to interact with OpenAI models such as gpt-4o-mini and text-embedding-3-small in LangChain.
langgraph	Used to create a directed sequence of events for the chatbot to execute. A major part of the backend, further described in Section 3.4.3.
pdfminer-six pypdf	Dependencies of LangChain for PDF reading.
Streamlit	Used as the frontend of the chatbot and also stores the conversation in memory. Described further in Section 3.4.4.

Table 3.1: The Python modules used in the project's development.

3.4.2 Data storage

The backbone of this project is the BCU-related data that the chatbot will pull from when queried. The vast majority of this data was sourced from the official Birmingham City University website (BCU, 2025), where individual policies are stored as PDF files for public download without any access limitations or restrictions. An observation made through an analysis of many of the policies was that none of them explicitly state key information about the university, such as campus building locations or information about its student union. Therefore, an additional document of my own creation with LaTeXwas included amongst the downloaded data. This document contained key information about BCU itself, with information on campus addresses and miscellaneous helpful information for students.

With all documents downloaded or created, the next stage would be to incorporate them in a format an LLM can interpret. This introduces LangChain, a popular framework for LLM app development (LangChain, 2024), which provides helper classes to directly read PDF files from a directory and split the text data within into smaller chunks, as seen in Figure 3.3.

```python
# Loads every PDF from the data path.
def load_documents():
    # In the data path, load every (signified by asterisk) PDF file.
    # Because they're PDF files, the PyPDFLoader can be used to load each.
    loader = DirectoryLoader(DATA_PATH,  glob = "*.pdf",
                             loader_cls = PyPDFLoader)
    documents = loader.load()
    return documents

# Uses LangChain's RecursiveCharacterTextSplitter to split the documents into chunks for embedding.
def split_text(documents):
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=2000,
        chunk_overlap=500,
        length_function=len,
        add_start_index=True,
    )

    # Save the split chunks.
    chunks = text_splitter.split_documents(documents)

    # Example: "Split 100 documents into 700 chunks"
    # Just for verification that the script ran.
    print(f"Split {len(documents)} documents into {len(chunks)} chunks.")

    # Return the chunks so that they can be embedded and saved.
    return chunks
```

Figure 3.3: Code used to load all PDFs from the Policies directory and split them into chunks.

Multiple chunk sizes and overlaps were tested during the artefact's development, with an eventual settlement on 2000 character chunks with 500 character overlaps being used. Maximising the size of chunks is a key part in assisting the chatbot's retrieval process, as it will be able to fetch more data with a single query which allows it to answer questions with greater detail and factual accuracy. The chunk overlap defines how many characters appear across multiple sequential chunks, ensuring that key information is unlikely to be split over multiple chunks where the chatbot then may be unable to cite it. LangChain's RecursiveCharacterTextSplitter also provides additional arguments for a custom length function if desired, though there was no need in this project, as well as adding start indexes to each vector, which adds metadata stating the numerical ID of each chunk as determined by the sequential order they are split in.

Once these chunks have been created, they must then be embedded as vectors, which will allow an LLM to interpret them. These vectors were then stored in a Facebook AI Similarity Search (FAISS) database as researched in Section 2.1.4, which ensured that the policies only needed to be embedded once rather than every time the chatbot was run, and would be retrieved at high speeds thanks to FAISS' efficiency.

```python
def save_to_faiss(chunks):
    # Clear out the database first if it exists.
    if os.path.exists(FAISS_PATH):
        shutil.rmtree(FAISS_PATH)

    # Create a new DB from the documents.
    # Takes the chunks and uses OpenAI text-embedding-3-small to embed them as vectors.
    faiss = FAISS.from_documents(
        chunks,
        OpenAIEmbeddings(
            model = "text-embedding-3-small", # Cost-efficient
            openai_api_key = os.environ["OPENAI_API_KEY"]
            )
    )

    # Save the generated DB to the given path.
    faiss.save_local(folder_path = FAISS_PATH)

    print(f"Saved {len(chunks)} chunks to {FAISS_PATH}.")
```

Figure 3.4: Code used to embed and store the chunks into a FAISS DB.

Firstly, any existing database in the specified directory is cleared to ensure that there are no I/O errors when attempting to save to the directory. LangChain provides wrapper functions for both the embedding and storage of this data, making it a smooth and simple process in very few lines of code.

The embedding model used is OpenAI's text-embedding-3-small model (OpenAI, 2024c). The motivation behind the use of this model was primarily due to its cost efficiency, with OpenAI approximating 62,500 pages can be embedded for each dollar spent. For each 2000-character chunk, the embedding model translates it into vector space for the LLM's interpretation. The vectors are produced based on the semantic similarities of each word as previously discussed and visualised in Section 2.1.2.

3.4.3 Chatbot backend

As previously mentioned, the core functionality of the chatbot was developed using the LangChain and LangGraph frameworks. LangChain in particular simplifies the development process by providing various functions and classes for quick and easy integration with necessary services such as FAISS and the OpenAI API, with LangGraph defining the chatbot's structure as depicted in Figure 3.5.

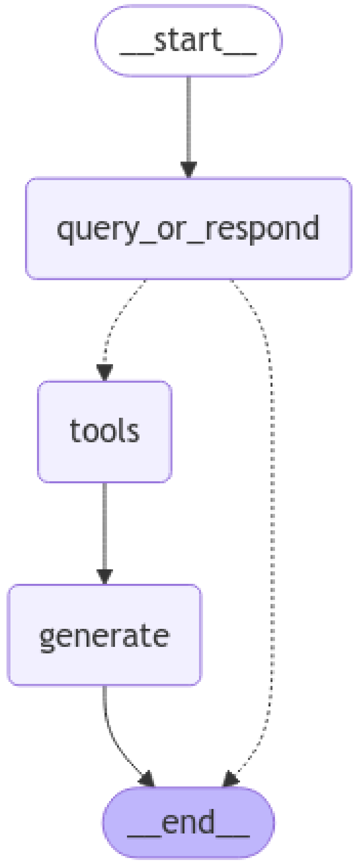

Figure 3.5: The graph for the chatbot.

Before detailing each node of the graph, it is first necessary to establish some prerequisite variables such as the LLM itself. The LLM used is OpenAI's gpt-4o-mini due to cost-efficiency. The difference in performance between 4o-mini and 4o was deemed not significant enough for the price increase of 1500% per 1 million tokens in relation to 4o-mini (OpenAI, 2025), especially considering that 4o-mini performs suitably for the task at hand.

Figure 3.6 shows each of the variables being established, including the LLM, via LangChain's 'init_chat_model' function. The LLM is initialised with a temperature of 0, which means that it should give the same answer to the same prompt whenever it is given. As mentioned, this does reduce the 'personality' of the chatbot, though it greatly helps to reduce the potential for hallucinations in a Q&A RAG scenario such as this one.

```python
# Sets the directory of the FAISS DB that's being loaded from.
    # Options (all begin with "VectorStores/"):
        #   FAISS: Chunk size 1000, Overlap 200, PyPDFLoader with default args.
        #   FAISS-Unstructured: Chunk size 1000, Overlap 200, UnstructuredPDFLoader with default args.
        #   FAISS-SmallChunks: Chunk size 500, Overlap 100, PyPDFLoader with default args.
        #   FAISS-BigChunks: Chunk size 1500, Overlap 300, PyPDFLoader with default args.
        #   FAISS-HugeChunks: Chunk size 2000, Overlap 500, PyPDFLoader with default args.
FAISS_PATH = "VectorStores/FAISS-HugeChunks"


# Sets up the embedding model with the API key.
embedder = OpenAIEmbeddings(
    model = "text-embedding-3-small",
    openai_api_key = os.environ["OPENAI_API_KEY"]
)

# Load the vector database.
db = FAISS.load_local(folder_path = FAISS_PATH,
                      embeddings = embedder,
                      allow_dangerous_deserialization=True)

# Initialise the LLM.
# LangChain automatically interprets the LLM in question to be OpenAI's gpt-4o-mini simply by
# specifying its name as a string argument.
llm = init_chat_model("gpt-4o-mini", temperature = 0,
                      openai_api_key = os.environ["OPENAI_API_KEY"])
```

Figure 3.6: Establishing prerequisite variables for the chatbot.

Thorough experimentation with the chunk sizes previously discussed in Figure 3.3 occurred during development, with the eventual decision of settling on the 2000-character chunks being made due to its more reliable performance on a set of sample questions discussed in Chapter 4. Following the relative directory of the FAISS database being set, the OpenAI embedding model is once again used so that similarity searches may be performed on the database. An additional argument, 'allow_dangerous_deserialization', is given when loading the database. When a FAISS database is saved using LangChain, it is saved as a serialized file known as a Pickle file, using a .pkl file extension. It is possible for malicious code to be embedded inside these files which could be executed when they are deserialized. However, as the files were generated specifically for this project and their contents are already known, it is safe to deserialize them.

With the prerequisite variables established, the first node of the graph was created. This is the core functionality of the LangGraph module, which builds on LangChain by defining an app's workflow as nodes and edges on a graph (LangGraph, 2025). In development, these nodes and edges can be created, with support for conditional edges that ensure certain nodes such as tool calls only activate when necessary. This allows for the creation of self-directed agents which make decisions independently, with this functionality being used in the chatbot to decide whether a response needs BCU-related context or not. This occurs in the first node and entry point of the graph: the 'query_or_respond' node.

```python
def query_or_respond(state: MessagesState):
    # Creates the retrieval agent by giving the LLM access to the retrieval tool.
    retrievalAgent = llm.bind_tools([retrieve])

    # The LLM decides on its own if it needs retrieval based on the existing conversation.
    response = retrievalAgent.invoke(state["messages"])

    # If it wants to use a tool, it will return a blank message with the metadata requesting a tool call.
    # Otherwise, it will return a generic message without any context, which occurs if the information is
    # already known or the query is simply too general ("Hello", for example).
    return {"messages": [response]}
```

Figure 3.7: Code used for the 'query_or_respond' graph node.

This function clearly demonstrates LangChain's abstractions of the backend functionality; these three lines of code serve as the entire decision-making logic for this node, as the LLM itself will decide whether it can immediately answer the user's query, such as in a scenario where the information they are requesting is already known from earlier in the conversation, or if their query is too general such as stating their name. If the LLM decides it cannot answer the query with the information it currently has available within the conversation, it will instead invoke the left conditional branch of the graph in Figure 3.5 by calling on the retriever tool denoted in Figure 3.8.

```python
@tool(response_format = "content")
def retrieve(query):
    # This docstring is used as the context for the LLM, letting it know what the tool does.
    """Retrieves the 3 most relevant context chunks for a given query.

    Args:
        query: The user's question, optimized for a semantic search."""

    retrievedChunks = db.similarity_search(query, k = 3)

    # Each retrieved chunk is seperated by two newlines.
    content = "\n\n".join(
        (f"{chunk.page_content}")
        for chunk in retrievedChunks
    )

    return content
```

Figure 3.8: Code used for the retrieval tool.

Using the '@tool' decorator informs LangChain that the following function is a tool to be used by the LLM. The retriever tool itself is simplistic in function: it will perform a semantic search on the FAISS database based on the user's query. LangChain enforces that all tools require a Python docstring explaining their function, as the LLM will read this docstring to understand what the tool is and how to use it. By specifying that the 'query' argument should be optimised for a semantic search, the 'query_or_respond' node will output a modified version of the user's query as the input to the tool, which is further discussed in Section 3.4.4.

When the 3 most similar chunks have been retrieved as defined by 'k' in the similarity search method, the text of each chunk is saved and separated by two newline characters to assist the LLM in understanding they are not part of the same text. The large text block is then returned as this tool's output, ready to then be passed into the 'generate' node.

```python
# The final step of the process, generating a message based on the info gathered
# from the retrieval tool.
def generate(state: MessagesState):
    # Retrieves the most recent tool call from the tools node.
    recentToolMsgs = []

    # The MessagesState stores the most recent messages at the bottom, as it's an append-only list.
    # This means that it'll need to be reversed for the most recent messages.
    for message in reversed(state["messages"]):
        if message.type == "tool":
            recentToolMsgs.append(message)
        else:
            # If it's a normal message, stop.
            # This is because the context from earlier messages doesn't need
            # repeating again, as it would enormously increase token usage and therefore cost.
            break

    # Saves the context from the retriever tool.
    docsContent = "\n\n".join(doc.content for doc in recentToolMsgs)

    # This is the LLM's system prompt, which decides how the LLM behaves.
    systemPrompt = f"""
        You are a friendly assistant to help new students get acclimated to Birmingham City University.
        If you don't know the answer, say that you don't know.
        When referring to context, be specific and quote the context.
        Use five sentences maximum and keep the answer concise.
        Use the following pieces of retrieved context to answer the question.
        \n\n
        Context: {docsContent}
    """
```

Figure 3.9: Code used for the 'generate' node (1/2).

The 'generate' function is the largest function of the chatbot, and is split across Figures 3.9 and 3.10. In the interests of saving cost, time and the potential risk of maximising the LLM's context window, the most recent retrieval tool call is saved. This tool call contains the RAG context from the retrieval tool, and will be at least 6,000 characters in length due to the previously mentioned chunk sizes and amount of chunks retrieved. This most recent tool call is the only context given to the LLM to reduce the token cost of each prompt and to mitigate any potential confusion if the LLM is given thousands of words of input context.

The retrieval tool returns three LangChain Document objects, each containing one chunk. Therefore, the content of each of these Documents is extracted and saved before being appended to the LLM's system prompt.

The system prompt is a massive part of LLM usage, and almost entirely dictates what the LLM will do based on any given input. The prompt is written in natural language which is interpreted by the LLM as a set of instructions to follow at all times. As such, the prompt given for the chatbot defines that it is a BCU assistant which should specifically quote context and keep all answers brief (for cost efficiency). This prompt was found to be highly effective, with the LLM responding with mostly satisfactory results as detailed in Chapter

4.

```
# The list of messages in the conversation.
# Only adds messages that AREN'T tool calls, as tool calls are blank messages.
conversation = [
    message for message in state["messages"] # Every message in the conversation
    if message.type in ("human", "system") # If it's human input or the system prompt
    or (message.type == "ai" and not message.tool_calls) # Or from the LLM and isn't a tool call.
]


# The LLM is given its system prompt (containing current retrieved context if there is any)
# alongside all other messages in the conversation.
history = [SystemMessage(systemPrompt)] + conversation


# Get the LLM's response to the prompt and return the response.
response = llm.invoke(history)
return {"messages": [response]}
```

Figure 3.10: Code used for the 'generate' node (2/2).

With the system prompt prepared, the LLM should also account for the current conversational history, which is retrieved from the LangGraph MessagesState. The MessagesState is an append-only list containing all messages in the current conversation, stored as a HumanMessage, AIMessage, or SystemMessage. These are three LangChain objects used to represent the various actors in a conversation: the human user, the LLM and the system prompt.

When retrieving the current conversation, tool calls are excluded. This is because through experimentation, it was discovered that when the chatbot calls on a tool, it generates a blank message with metadata indicating a tool call. This blank message is not relevant, and therefore does not need to be included in the conversational history.

Finally, the new system prompt is inserted as the most recent message in the conversation, and the LLM is invoked with the filtered conversation history, with the generated response being returned.

```
# Initialise the graph.
graph = StateGraph(MessagesState)

# Add all the nodes.
graph.add_node(query_or_respond)
graph.add_node(tools)
graph.add_node(generate)

# The graph starts with choosing whether to query the DB or directly respond.
graph.set_entry_point("query_or_respond")

# query_or_respond has conditions:
# If the user's query needs RAG context, the retrieve tool will be called.
# If it does not, the LLM will generate a response by itself.

# "What will my grade be reduced by if I submit 3 days late?" invokes the retrieval tool.
# "Hello!" should not.
graph.add_conditional_edges(
    "query_or_respond", tools_condition,
    # tools_condition is True if the Agent wants to use a tool.
    # If retrieval is not needed, skip to the end, which generates a general non-BCU related answer.
    # If retrieval is needed, call the retrieval tool.
    {END: END,
     "tools": "tools"},
)

# An edge is also needed between the tool call and response generation
# to ensure the response has the RAG context.
graph.add_edge("tools", "generate")

# After the response is generated, the graph is done.
graph.add_edge("generate", END)

# Compile the graph so Streamlit can use it.
graph = graph.compile()
```

Figure 3.11: Code used to form the graph.

To conclude the chatbot's backend Python script, the LangGraph is created using the conditions mentioned previously. A particularly helpful feature of LangGraph for this scenario was 'tools_condition', which is set to True if the LLM calls on a tool, and False if it does not. Based on this condition, the graph will either go down the left branch, invoking the retrieval tool and generating a contextualised response, or it will skip directly to the end, where the LLM will generate a generic answer without any RAG.

3.4.4 Accessing the chatbot

Evaluation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et erat consectetur, scelerisque eros nec, ultricies est. Donec mi ipsum, imperdiet vitae arcu quis, luctus venenatis quam. Integer ac massa a augue venenatis fringilla. Etiam posuere libero sed nulla tristique volutpat.

4.1 Methodology

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et erat consectetur, scelerisque eros nec, ultricies est. Donec mi ipsum, imperdiet vitae arcu quis, luctus venenatis quam. Integer ac massa a augue venenatis fringilla. Etiam posuere libero sed nulla tristique volutpat.

4.1.1 Metrics

!!!

4.1.2 Baseline systems

4.1.3 Dataset

Likely not applicable. OpenAI's models are all closed-source.

4.2 Results

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et erat consectetur, scelerisque eros nec, ultricies est. Donec mi ipsum, imperdiet vitae arcu quis, luctus venenatis quam. Integer ac massa a augue venenatis fringilla. Etiam posuere libero sed nulla tristique volutpat.

4.3 Discussion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et erat consectetur, scelerisque eros nec, ultricies est. Donec mi ipsum, imperdiet vitae arcu quis, luctus venenatis quam. Integer ac massa a augue venenatis fringilla. Etiam posuere libero sed nulla tristique volutpat.

Conclusions

"You should not include any new information or discussion in this section." This section must also link to the project's objectives.

Recommendations for future work

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et erat consectetur, scelerisque eros nec, ultricies est. Donec mi ipsum, imperdiet vitae arcu quis, luctus venenatis quam. Integer ac massa a augue venenatis fringilla. Etiam posuere libero sed nulla tristique volutpat.

References

Adobe (8th Aug. 2023). *Popular Project Management Methodologies*. Popular project management methodologies. URL: https://business.adobe.com/blog/basics/methodologies (visited on 22/03/2025).

Adobe (2025). *Waterfall Methodology: Project Management | Adobe Workfront | Adobe UK*. URL: https://business.adobe.com/uk/blog/basics/waterfall (visited on 22/03/2025).

Asana (2025). *What Is Agile Methodology?* Asana. URL: https://asana.com/resources/agile-methodology (visited on 22/03/2025).

Astral (2025). *UV*. URL: https://docs.astral.sh/uv/ (visited on 22/03/2025).

BCU (2025). *Policies and Procedures*. Birmingham City University. URL: https://www.bcu.ac.uk/about-us/corporate-information/policies-and-procedures (visited on 22/03/2025).

Chammas, Adriana, Manuela Quaresma and Cláudia Mont'Alvão (1st Jan. 2015). 'A Closer Look on the User Centred Design'. In: *Procedia Manufacturing*. 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015 3, pp. 5397–5404. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2015.07.656.

Chroma (2024). *Chroma*. URL: https://www.trychroma.com/ (visited on 24/11/2024).

Clark, Leigh, Nadia Pantidi, Orla Cooney, Philip Doyle, Diego Garaialde, Justin Edwards, Brendan Spillane, Emer Gilmartin, Christine Murad, Cosmin Munteanu, Vincent Wade and Benjamin R. Cowan (2nd May 2019). 'What Makes a Good Conversation? Challenges in Designing Truly Conversational Agents'. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. New York, NY, USA: Association for Computing Machinery, pp. 1–12. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300705.

DeepEval (22nd Nov. 2024). *Introduction | DeepEval - The Open-Source LLM Evaluation Framework*. URL: https://docs.confident-ai.com/docs/metrics-introduction (visited on 24/11/2024).

Devlin, Jacob, Ming-Wei Chang, Kenton Lee and Kristina Toutanova (June 2019). 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding'. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. NAACL-HLT 2019. Ed. by Jill Burstein, Christy Doran and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

Dubey, Abhimanyu et al. (15th Aug. 2024). *The Llama 3 Herd of Models*. DOI: 10.48550/arXiv.2407.21783. arXiv: 2407.21783.

Dwivedi, Y.K., N. Kshetri, L. Hughes, E.L. Slade, A. Jeyaraj, A.K. Kar, A.M. Baabdullah, A. Koohang, V. Raghavan, M. Ahuja, H. Albanna, M.A. Albashrawi, A.S. Al-Busaidi, J. Balakrishnan, Y. Barlette, S. Basu, I. Bose, L. Brooks, D. Buhalis, L. Carter, S. Chowdhury, T. Crick, S.W. Cunningham, G.H. Davies, R.M. Davison, R. Dé, D. Dennehy, Y. Duan, R. Dubey, R. Dwivedi, J.S. Edwards, C. Flavián, R. Gauld, V. Grover, M.-C. Hu, M. Janssen, P. Jones, I. Junglas, S. Khorana, S. Kraus, K.R. Larsen, P. Latreille, S. Laumer, F.T. Malik, A. Mardani, M. Mariani, S. Mithas, E. Mogaji, J.H. Nord, S. O'Connor, F. Okumus, M. Pagani, N. Pandey, S. Papagiannidis, I.O. Pappas, N. Pathak,

J. Pries-Heje, R. Raman, N.P. Rana, S.-V. Rehm, S. Ribeiro-Navarrete, A. Richter, F. Rowe, S. Sarker, B.C. Stahl, M.K. Tiwari, W. van der Aalst, V. Venkatesh, G. Viglia, M. Wade, P. Walton, J. Wirtz and R. Wright (2023). '"So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy'. In: *International Journal of Information Management* 71. DOI: 10.1016/j.ijinfomgt.2023.102642.

Ge, Jin, Steve Sun, Joseph Owens, Victor Galvez, Oksana Gologorskaya, Jennifer C Lai, Mark J Pletcher and Ki Lai (1st Nov. 2023). 'Development of a Liver Disease-Specific Large Language Model Chat Interface using Retrieval Augmented Generation'. In: *medRxiv*, p. 2023.11.10.23298364. DOI: 10.1101/2023.11.10.23298364.

Google (2024). *Conversational Agents and Dialogflow*. Google Cloud. URL: https://cloud.google.com/products/conversational-agents (visited on 24/11/2024).

IBM (3rd Apr. 2024a). *IBM watsonx Assistant Virtual Agent*. URL: https://www.ibm.com/products/watsonx-assistant (visited on 24/11/2024).

Karpukhin, Vladimir, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen and Wen-tau Yih (Nov. 2020). 'Dense Passage Retrieval for Open-Domain Question Answering'. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. EMNLP 2020. Ed. by Bonnie Webber, Trevor Cohn, Yulan He and Yang Liu. Online: Association for Computational Linguistics, pp. 6769–6781. DOI: 10.18653/v1/2020.emnlp-main.550.

Komeili, Mojtaba, Kurt Shuster and Jason Weston (May 2022). 'Internet-Augmented Dialogue Generation'. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2022. Ed. by Smaranda Muresan, Preslav Nakov and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, pp. 8460–8478. DOI: 10.18653/v1/2022.acl-long.579.

Kotian, Abhijith L, Reshna Nandipi, Ushag M, Usha Rani S, VARSHAUK and Veena G T (Jan. 2024). 'A Systematic Review on Human and Computer Interaction'. In: *2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*. 2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT), pp. 1214–1218. DOI: 10.1109/IDCIoT59759.2024.10467622.

Krizhevsky, Alex, Ilya Sutskever and Geoffrey E Hinton (2012). 'ImageNet Classification with Deep Convolutional Neural Networks'. In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc. DOI: 10.1145/3065386.

Kuhail, M.A., N. Alturki, S. Alramlawi and K. Alhejori (2023). 'Interacting with educational chatbots: A systematic review'. In: *Education and Information Technologies* 28 (1), pp. 973–1018. DOI: 10.1007/s10639-022-11177-3.

LangChain (2024). *Introduction | LangChain*. URL: https://python.langchain.com/docs/introduction/ (visited on 24/11/2024).

LangGraph (2025). *LangGraph*. URL: https://langchain-ai.github.io/langgraph/ (visited on 23/03/2025).

Le, Quoc V. and Tomas Mikolov (22nd May 2014). *Distributed Representations of Sentences and Documents*. DOI: 10.48550/arXiv.1405.4053. arXiv: 1405.4053.

Lewis, Mike, Marjan Ghazvininejad, Gargi Ghosh, Armen Aghajanyan, Sida Wang and Luke Zettlemoyer (26th June 2020). *Pre-training via Paraphrasing*. DOI: 10.48550/arXiv.2006.15020. arXiv: 2006.15020.

Lewis, Patrick, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel and Douwe Kiela (12th Apr. 2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. DOI: 10.48550/arXiv.2005.11401. arXiv: 2005.11401.

Li, Feifei (1st Aug. 2023). 'Modernization of Databases in the Cloud Era: Building Databases that Run Like Legos'. In: *Proc. VLDB Endow.* 16 (12), pp. 4140–4151. ISSN: 2150-8097. DOI: 10.14778/3611540.3611639.

Liao, Q. Vera, Muhammed Mas-ud Hussain, Praveen Chandar, Matthew Davis, Yasaman Khazaeni, Marco Patricio Crasso, Dakuo Wang, Michael Muller, N. Sadat Shami and Werner Geyer (19th Apr. 2018). 'All Work and No Play?' In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI '18. New York, NY, USA: Association for Computing Machinery, pp. 1–13. ISBN: 978-1-4503-5620-6. DOI: 10.1145/3173574.3173577.

Liu, Yang, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu and Chenguang Zhu (23rd May 2023). *G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment*. DOI: 10.48550/arXiv.2303.16634. arXiv: 2303.16634.

Luger, Ewa and Abigail Sellen (7th May 2016). '"Like Having a Really Bad PA": The Gulf between User Expectation and Experience of Conversational Agents'. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI '16. New York, NY, USA: Association for Computing Machinery, pp. 5286–5297. ISBN: 978-1-4503-3362-7. DOI: 10.1145/2858036.2858288.

Maedche, Alexander, Christine Legner, Alexander Benlian, Benedikt Berger, Henner Gimpel, Thomas Hess, Oliver Hinz, Stefan Morana and Matthias Söllner (1st Aug. 2019). 'AI-Based Digital Assistants'. In: *Business & Information Systems Engineering* 61 (4), pp. 535–544. ISSN: 1867-0202. DOI: 10.1007/s12599-019-00600-8.

Malsam, William (26th Aug. 2024). *What Is Scope Creep and How Can I Avoid It?* ProjectManager. URL: https://www.projectmanager.com/blog/5-ways-to-avoid-scope-creep (visited on 22/03/2025).

Microsoft (2024). *Microsoft Bot Framework*. URL: https://dev.botframework.com/ (visited on 24/11/2024).

Mikolov, Tomas, Kai Chen, Greg Corrado and Jeffrey Dean (7th Sept. 2013). *Efficient Estimation of Word Representations in Vector Space*. DOI: 10.48550/arXiv.1301.3781. arXiv: 1301.3781.

Miró-Nicolau, M., A. Jaume-i-Capó and G. Moyà-Alcover (2025). 'A comprehensive study on fidelity metrics for XAI'. In: *Information Processing and Management* 62 (1). DOI: 10.1016/j.ipm.2024.103900.

Neumann, Alexander Tobias, Yue Yin, Sulayman Sowe, Stefan Decker and Matthias Jarke (2024). 'An LLM-Driven Chatbot in Higher Education for Databases and Information Systems'. In: *IEEE Transactions on Education*. Conference Name: IEEE Transactions on Education, pp. 1–14. ISSN: 1557-9638. DOI: 10.1109/TE.2024.3467912.

Odede, Julius and Ingo Frommholz (10th Mar. 2024). 'JayBot – Aiding University Students and Admission with an LLM-based Chatbot'. In: *Proceedings of the 2024 Con-*

ference on Human Information Interaction and Retrieval. CHIIR '24. New York, NY, USA: Association for Computing Machinery, pp. 391–395. ISBN: 9798400704345. DOI: 10.1145/3627508.3638293.

OpenAI (2024a). *Introducing text and code embeddings*. URL: https://openai.com/index/introducing-text-and-code-embeddings/ (visited on 25/11/2024).

OpenAI (2025). *Pricing*. URL: https://openai.com/api/pricing/ (visited on 23/03/2025).

OpenAI (2024b). *Retrieval Augmented Generation (RAG) and Semantic Search for GPTs | OpenAI Help Center*. URL: https://help.openai.com/en/articles/8868588-retrieval-augmented-generation-rag-and-semantic-search-for-gpts (visited on 23/11/2024).

OpenAI (2024c). *Vector embeddings*. Vector embeddings. URL: https://platform.openai.com/docs/guides/embeddings (visited on 24/11/2024).

Ouyang, Long, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike and Ryan Lowe (4th Mar. 2022). *Training language models to follow instructions with human feedback*. DOI: 10.48550/arXiv.2203.02155. arXiv: 2203.02155.

Pinecone (2024). *Pinecone Documentation*. Pinecone Docs. URL: https://docs.pinecone.io/guides/get-started/overview (visited on 24/11/2024).

Putnam, Cynthia, Kathryn Wozniak, Mary Jo Zefeldt, Jinghui Cheng, Morgan Caputo and Carl Duffield (22nd Oct. 2012). 'How do professionals who create computing technologies consider accessibility?' In: *Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility*. ASSETS '12. New York, NY, USA: Association for Computing Machinery, pp. 87–94. ISBN: 978-1-4503-1321-6. DOI: 10.1145/2384916.2384932.

Samuel, A. L. (July 1959). 'Some Studies in Machine Learning Using the Game of Checkers'. In: *IBM Journal of Research and Development* 3 (3). Conference Name: IBM Journal of Research and Development, pp. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210.

Schöbel, Sofia, Anuschka Schmitt, Dennis Benner, Mohammed Saqr, Andreas Janson and Jan Marco Leimeister (1st Apr. 2024). 'Charting the Evolution and Future of Conversational Agents: A Research Agenda Along Five Waves and New Frontiers'. In: *Information Systems Frontiers* 26 (2), pp. 729–754. ISSN: 1572-9419. DOI: 10.1007/s10796-023-10375-9.

Selbst, Andrew D., Danah Boyd, Sorelle A. Friedler, Suresh Venkatasubramanian and Janet Vertesi (29th Jan. 2019). 'Fairness and Abstraction in Sociotechnical Systems'. In: *Proceedings of the Conference on Fairness, Accountability, and Transparency*. FAT* '19. New York, NY, USA: Association for Computing Machinery, pp. 59–68. ISBN: 978-1-4503-6125-5. DOI: 10.1145/3287560.3287598.

Shneiderman, Ben (16th Oct. 2020). 'Bridging the Gap Between Ethics and Practice: Guidelines for Reliable, Safe, and Trustworthy Human-centered AI Systems'. In: *ACM Trans. Interact. Intell. Syst.* 10 (4), 26:1–26:31. ISSN: 2160-6455. DOI: 10.1145/3419764.

Shuster, Kurt, Spencer Poff, Moya Chen, Douwe Kiela and Jason Weston (Nov. 2021). 'Retrieval Augmentation Reduces Hallucination in Conversation'. In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Findings 2021. Ed. by Marie-

Francine Moens, Xuanjing Huang, Lucia Specia and Scott Wen-tau Yih. Punta Cana, Dominican Republic: Association for Computational Linguistics, pp. 3784–3803. DOI: 10.18653/v1/2021.findings-emnlp.320.

Singer, Maxwell B., Julia J. Fu, Jessica Chow and Christopher C. Teng (1st Mar. 2024). 'Development and Evaluation of Aeyeconsult: A Novel Ophthalmology Chatbot Leveraging Verified Textbook Knowledge and GPT-4'. In: *Journal of Surgical Education* 81 (3), pp. 438–443. ISSN: 1931-7204. DOI: 10.1016/j.jsurg.2023.11.019.

Siriwardhana, Shamane, Rivindu Weerasekera, Elliott Wen, Tharindu Kaluarachchi, Rajib Rana and Suranga Nanayakkara (12th Jan. 2023). 'Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering'. In: *Transactions of the Association for Computational Linguistics* 11, pp. 1–17. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00530.

Srivastava, Saurabh and T.V. Prabhakar (Sept. 2020). 'Desirable Features of a Chatbot-building Platform'. In: *2020 IEEE International Conference on Humanized Computing and Communication with Artificial Intelligence (HCCAI)*. 2020 IEEE International Conference on Humanized Computing and Communication with Artificial Intelligence (HCCAI), pp. 61–64. DOI: 10.1109/HCCAI49649.2020.00016.

Turing, A. M. (1st Oct. 1950). 'I.—COMPUTING MACHINERY AND INTELLIGENCE'. In: *Mind* LIX (236), pp. 433–460. ISSN: 1460-2113, 0026-4423. DOI: 10.1093/mind/LIX.236.433.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser and Illia Polosukhin (4th Dec. 2017). 'Attention is all you need'. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., pp. 6000–6010. ISBN: 978-1-5108-6096-4. DOI: 10.48550/arXiv.1706.03762.

Vrontis, Demetris, Michael Christofi, Vijay Pereira, Shlomo Tarba, Anna Makrides and Eleni Trichina (26th Mar. 2022). 'Artificial intelligence, robotics, advanced technologies and human resource management: a systematic review'. In: *The International Journal of Human Resource Management* 33 (6). Publisher: Routledge, pp. 1237–1266. ISSN: 0958-5192. DOI: 10.1080/09585192.2020.1871398.

Wang, Jianguo, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei and Charles Xie (18th June 2021). 'Milvus: A Purpose-Built Vector Data Management System'. In: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD '21. New York, NY, USA: Association for Computing Machinery, pp. 2614–2627. ISBN: 978-1-4503-8343-1. DOI: 10.1145/3448016.3457550.

Wang, Zhaoxia, Chee Seng Chong, Landy Lan, Yinping Yang, Seng Beng Ho and Joo Chuan Tong (Dec. 2016). 'Fine-grained sentiment analysis of social media with emotion sensing'. In: *2016 Future Technologies Conference (FTC)*. 2016 Future Technologies Conference (FTC), pp. 1361–1364. DOI: 10.1109/FTC.2016.7821783.

Weizenbaum, Joseph (1st Jan. 1966). 'ELIZA—a computer program for the study of natural language communication between man and machine'. In: *Commun. ACM* 9 (1), pp. 36–45. ISSN: 0001-0782. DOI: 10.1145/365153.365168.

Wirtz, J., P.G. Patterson, W.H. Kunz, T. Gruber, V.N. Lu, S. Paluch and A. Martins (2018). 'Brave new world: service robots in the frontline'. In: *Journal of Service Management* 29 (5), pp. 907–931. DOI: 10.1108/JOSM-04-2018-0119.

Xie, Xingrui, Han Liu, Wenzhe Hou and Hongbin Huang (Dec. 2023). 'A Brief Survey of Vector Databases'. In: *2023 9th International Conference on Big Data and Information Analytics (BigDIA)*. 2023 9th International Conference on Big Data and Information Analytics (BigDIA). ISSN: 2771-6902, pp. 364–371. DOI: 10.1109/BigDIA60676.2023.10429609.

Zamfirescu-Pereira, J.D., Richmond Y. Wong, Bjoern Hartmann and Qian Yang (19th Apr. 2023). 'Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts'. In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. CHI '23. New York, NY, USA: Association for Computing Machinery, pp. 1–21. ISBN: 978-1-4503-9421-5. DOI: 10.1145/3544548.3581388.

Zhang, Yue, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi and Shuming Shi (24th Sept. 2023). *Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models*. DOI: 10.48550/arXiv.2309.01219. arXiv: 2309.01219.

Bibliography

AWS (2024). *What is RAG? - Retrieval-Augmented Generation AI Explained - AWS*. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/retrieval-augmented-generation/ (visited on 23/11/2024).

Cambridge Dictionary (2024). *Meaning of user experience in English*. URL: https://dictionary.cambridge.org/dictionary/english/user-experience (visited on 28/10/2024).

Cloudflare (2024). *What is a large language model (LLM)?* URL: https://www.cloudflare.com/en-gb/learning/ai/what-is-large-language-model/ (visited on 28/10/2024).

Confident AI (2024). *LLM Evaluation Metrics: The Ultimate LLM Evaluation Guide - Confident AI*. URL: https://www.confident-ai.com/blog/llm-evaluation-metrics-everything-you-need-for-llm-evaluation (visited on 24/11/2024).

Databricks (18th Oct. 2023). *Retrieval Augmented Generation*. Databricks. URL: https://www.databricks.com/glossary/retrieval-augmented-generation-rag (visited on 23/11/2024).

Elastic (2024). *What is Semantic Search? | A Comprehensive Semantic Search Guide*. URL: https://www.elastic.co/what-is/semantic-search (visited on 24/11/2024).

IBM (16th Aug. 2024b). *What is AI?* URL: https://www.ibm.com/topics/artificial-intelligence (visited on 28/10/2024).

IBM (11th Aug. 2024c). *What Is NLP (Natural Language Processing)? | IBM*. URL: https://www.ibm.com/topics/natural-language-processing (visited on 04/11/2024).

IBM (2024d). *What is generative AI?* URL: https://research.ibm.com/blog/what-is-generative-AI (visited on 28/10/2024).

ICO (2024). *Definitions*. URL: https://ico.org.uk/for-organisations/uk-gdpr-guidance-and-resources/artificial-intelligence/explaining-decisions-made-with-artificial-intelligence/part-1-the-basics-of-explaining-ai/definitions/ (visited on 28/10/2024).

MIT (9th Nov. 2023). *Explained: Generative AI*. URL: https://news.mit.edu/2023/explained-generative-ai-1109 (visited on 28/10/2024).