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
Word count: 10,182

Abstract

Artificial intelligence (AI), natural language processing (NLP) and large language models (LLMs) are rapidly developing technologies, seeing constant advancements at a frequent basis. This project aims to leverage these new advancements, specifically in LLMs, to create a digital assistant to help new students of Birmingham City University get acquainted to their new environment. This is accomplished through the development of a chatbot web application which uses Retrieval-Augmented Generation (RAG) with OpenAI's gpt-4o-mini LLM on an embedded vector database of university information. The development process is thoroughly explored, with key elements of the chatbot's code being discussed in detail. The produced chatbot performs well, achieving 80% answer correctness on a dataset of testing questions evaluated using DeepEval's GEval metric with a gpt-4o LLM and manual verification.

Acknowledgements

I would like to primarily acknowledge my project supervisor, Dr. Atif Azad, as having a major positive influence throughout the development of this project. His expertise and advice had a significant effect throughout the entire project's development and report writing, and both would be of substantially lower quality were it not for his guidance.

Additionally, I would like to acknowledge my father, who continually motivated me to produce high-quality work throughout a deeply challenging year. Without the support of both of these people, this project would not have been feasible, and for this I thank them both greatly.

Glossary

Terminology	Description
AI	A field of computing dedicated to allowing computers to simulate human learning by training them on large amounts of data so that they can recognise patterns to classify or predict unknown data. AI can only be as good as the data it is trained upon ("Garbage in, garbage out"), and could become biased if it is fed too much data of a certain type.
Natural Language Processing (NLP)	NLP refers to the use of machine learning to encode and process text to understand it similarly to humans, which can be used to allow direct two-way conversation between users and computers.
LLMs	Large Language Models are a type of machine learning model dedicated to the recognition and generation of text. As suggested by their name, they are trained on enormous amounts of text data, which allows them to have active conversations with users. There are many different LLMs, and as their size and complexity increases, so too does the necessary processing power.
Retrieval-Augmented Generation (RAG)	The optimisation of the generated text output of an LLM, incorporating an external data source to enhance its contextual knowledge and the subject relevancy of its outputs.
Chatbot Conversational Agent	Software that simulates a natural conversation between the computer and end user. Many chatbots, including the one to be produced in this project, utilise recent developments such as Generative AI and natural language processing (NLP) to interpret and respond to user queries. (IBM, 2024d)

Contents

List of Figures

Introduction

1.1 Problem definition

New university students often face challenges in acclimating to their new university life (Oxford University, 2021). These challenges can stem from difficulty in locating buildings or understanding university policies, and traditional support systems, such as printed materials or static websites, can often be insufficient, as they require students to navigate complex and scattered resources. Additionally, relying on human staff for queries can become especially difficult during peak times, such as the start of the academic year.

1.2 Aim

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

1.3 Scope

This project will focus on the development of a chatbot with a knowledge base of university data to query and retrieve information from. This will be accomplished by using an OpenAI LLM via their API, which will be customised to enable Retrieval-Augmented Generation (RAG), a technology which enhances an LLM's knowledge by giving it access to additional information that was not originally present in their training data (P. Lewis et al., 2021), making it highly contextualised and usable for BCU-related queries.

The chatbot will have a graphical user interface (GUI) to ensure that users can quickly come to terms with how to use the chatbot, which will be hosted as a web application accessible on the local network. Creating a hosted website is not within this project's scope as this would introduce unnecessary cost and waste time.

This project will also involve the testing and evaluation of the chatbot to ensure its usability and accuracy with a variety of BCU-related questions.

1.4 Rationale

The rapid development of artificial intelligence (AI), particularly in natural language processing (NLP) and large language models (LLMs), provides an opportunity to address these challenges. Therefore, the problem also lies in creating a reliable, cost-effective, and user-friendly digital assistant capable of answering various university-related inquiries with a high level of accuracy, which this project aims to solve. Unlike human staff, a chatbot can theoretically run at all times and respond at a much faster rate, meaning students could get the information they need at any time in real-time.

Many LLMs already exist, such as ChatGPT (OpenAI, 2025a), DeepSeek (DeepSeek, 2025), and Gemini (Google, 2025). However, these LLMs do not possess specific BCU-related knowledge. This provides a significant opportunity for the assistance of BCU students, which this project aims to capitalise on.

1.5 Objectives

The project's objectives are to:

- Conduct a thorough literature review on the surrounding topics, especially AI, LLMs and RAG.
- 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 general information with a minimum 75% accuracy rate.
- Assess the accuracy and usability of an AI chatbot in answering university-related student queries.

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.

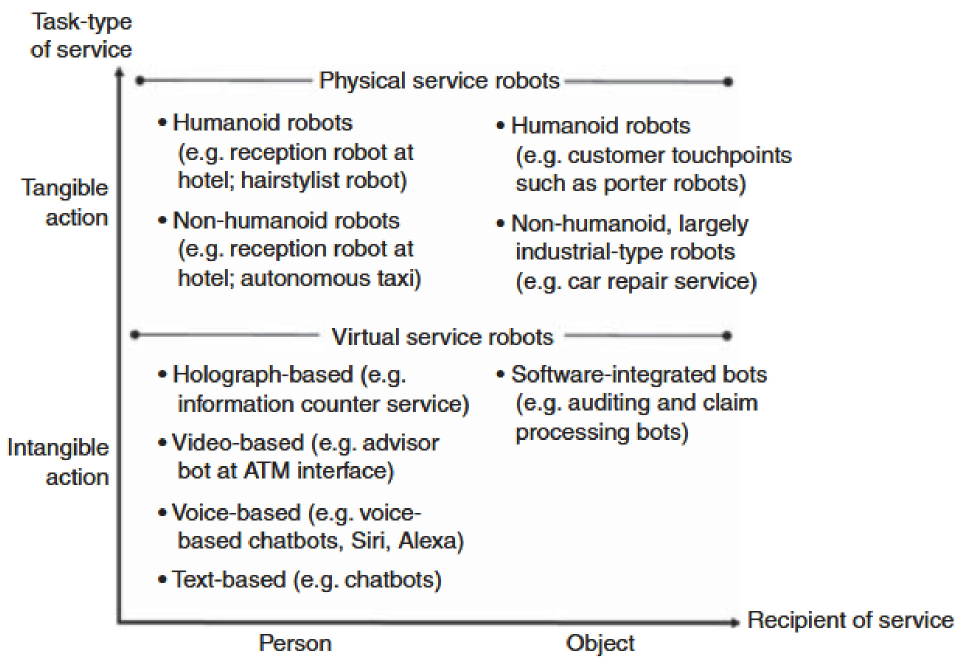

Figure 2.1: Service robots categorization by task-type and recipient of service (Wirtz et al., 2018).

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.

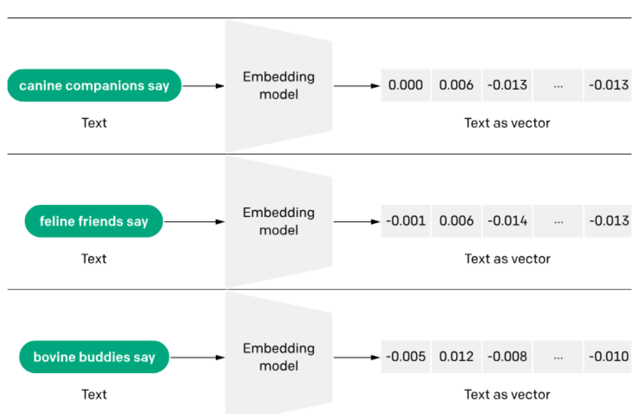

Figure 2.2: A basic overview of vectorisation (OpenAI, 2024a).

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.

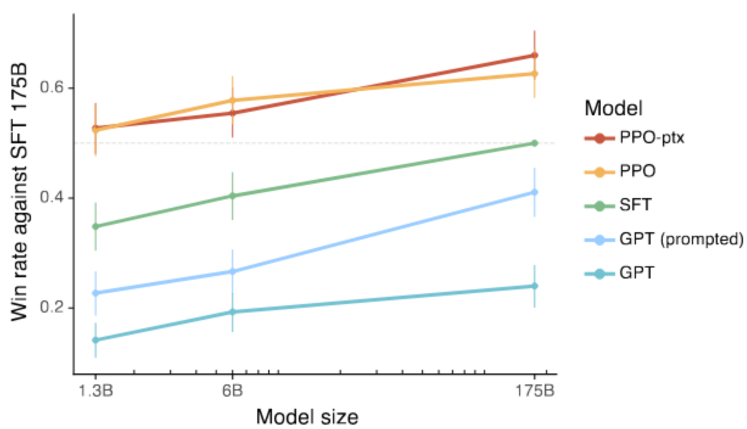

Figure 2.3: Human evaluations of the GPT models produced by Ouyang et al. (2022). PPO and PPO-ptx are their models.

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. (2023b), 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)).

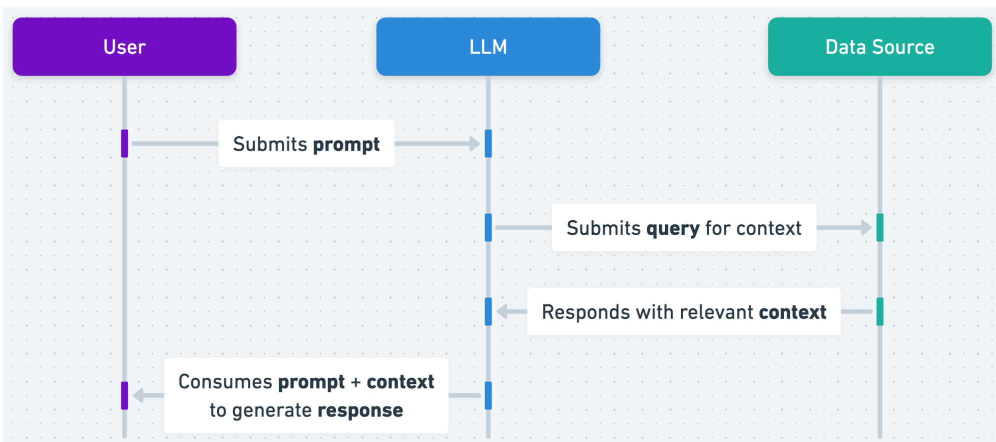

Figure 2.4: A basic overview of a RAG workflow (OpenAI, 2024b).

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.

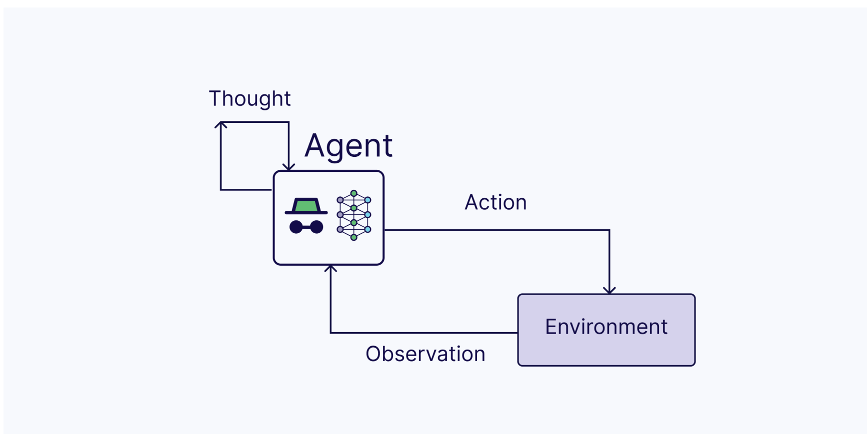

Figure 2.5: A basic ReAct (Reason + Act) agent workflow (Weaviate, 2024).

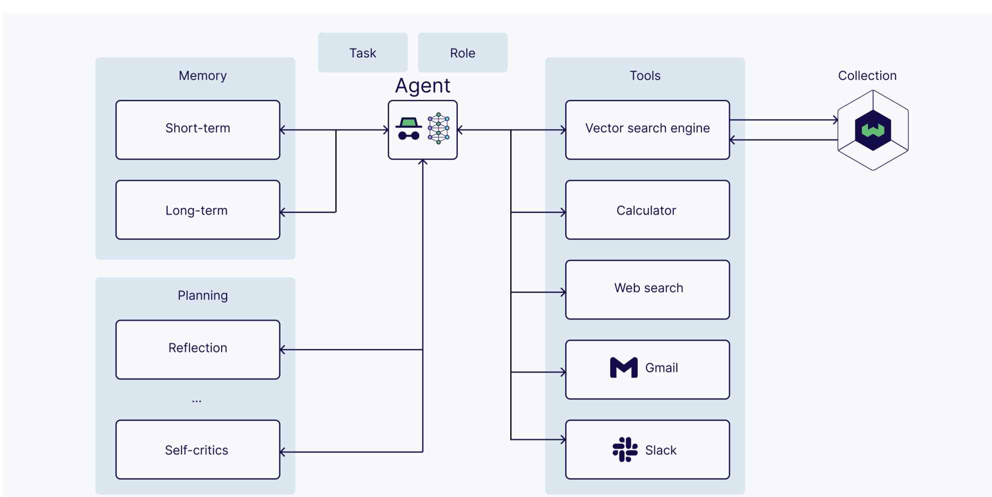

Figure 2.6: An advanced example agent workflow (Weaviate, 2024).

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).

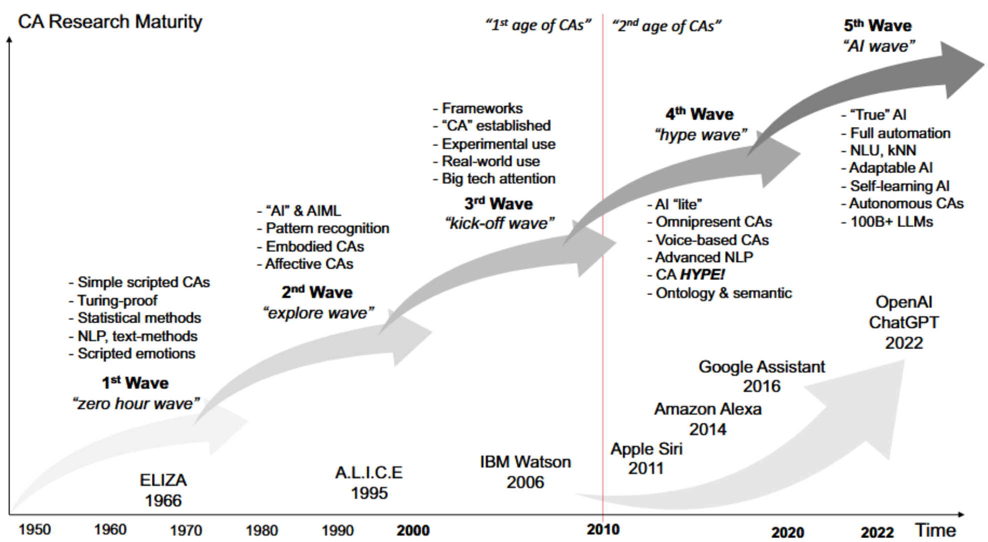

Figure 2.7: The five waves of conversational agent research (Schöbel et al., 2024).

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 FAISS vector database storing embeddings generated by OpenAI's text-embeddings-3-small model, and the overall framework used for this will be LangChain. This will keep the cost of the project low while maintaining a good level of quality in the chatbot's responses.

To evaluate the chatbot, DeepEval's GEval can be used to judge responses paired with manual human evaluation to ensure the reported results are accurate.

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.

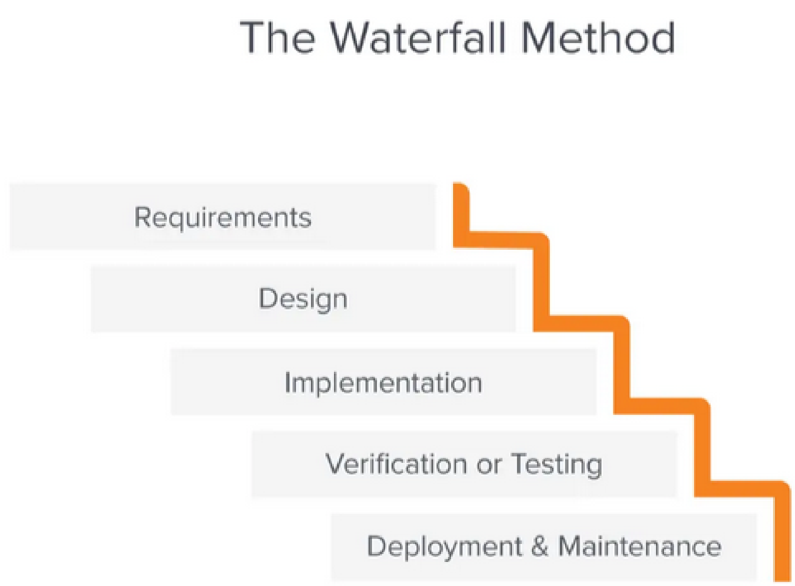

Figure 3.1: An overview of a Waterfall workflow (Adobe, 2025).

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 and mitigations

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.

To mitigate this risk, a Gantt chart was developed to model the overall project timeline, depicted in Figure 3.3

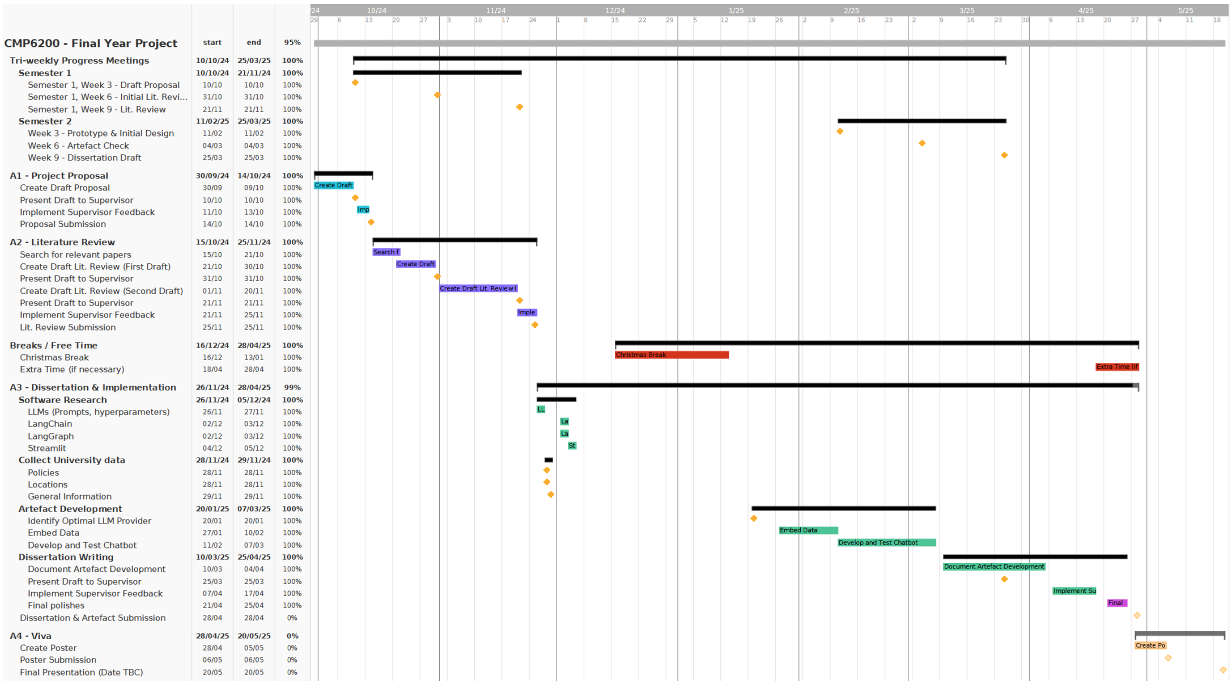

Figure 3.3: The overall project Gantt chart (TeamGantt, 2025).

The project's development was scheduled around finishing two weeks early, with those additional two weeks intended as a safety net for if uncontrollable circumstances negatively affect the project's development. In doing so, a solid plan was set in place that was resilient against setbacks.

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 input tokens (OpenAI, 2025c). Throughout the development and testing processes in each sprint, a cost will slowly begin to accrue.

Mitigating risks posed by potential cost constraints will be remedied by identifying the optimal model, which was decided to be gpt-4o-mini. The project has a limited budget due to it being a solo endeavour, though using a somewhat less intelligent model will greatly mitigate the risk of overspending while only compromising slightly on answer quality.

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.

To address this risk, time will be budgeted to allow for thorough research into the necessary tech stack to ensure that the project can be completed to a suitable standard.

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.

In an effort to mitigate any potential risk of LLM unpredictability, the chatbot's 'temperature' parameter will be set to 0, which will make its responses more static. This means that the chatbot should provide the same answer any time it is asked a certain question unless there are external circumstances (previous conversation history, etc.).

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 the current optimal methods for specific aims can be challenging.

Despite this risk, LangChain is recognised as a good framework for developing LLM-based applications and there are other resources that can be learned from that are not the documentation page, such as tutorial videos and blogs. Through using alternative resources alongside the official documentation, it should be possible to use LangChain to a good standard to produce an end product of suitable quality.

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 Concept diagrams

Figure 3.4 depicts a theoretical interaction with the chatbot from the frontend user's perspective created early in development.

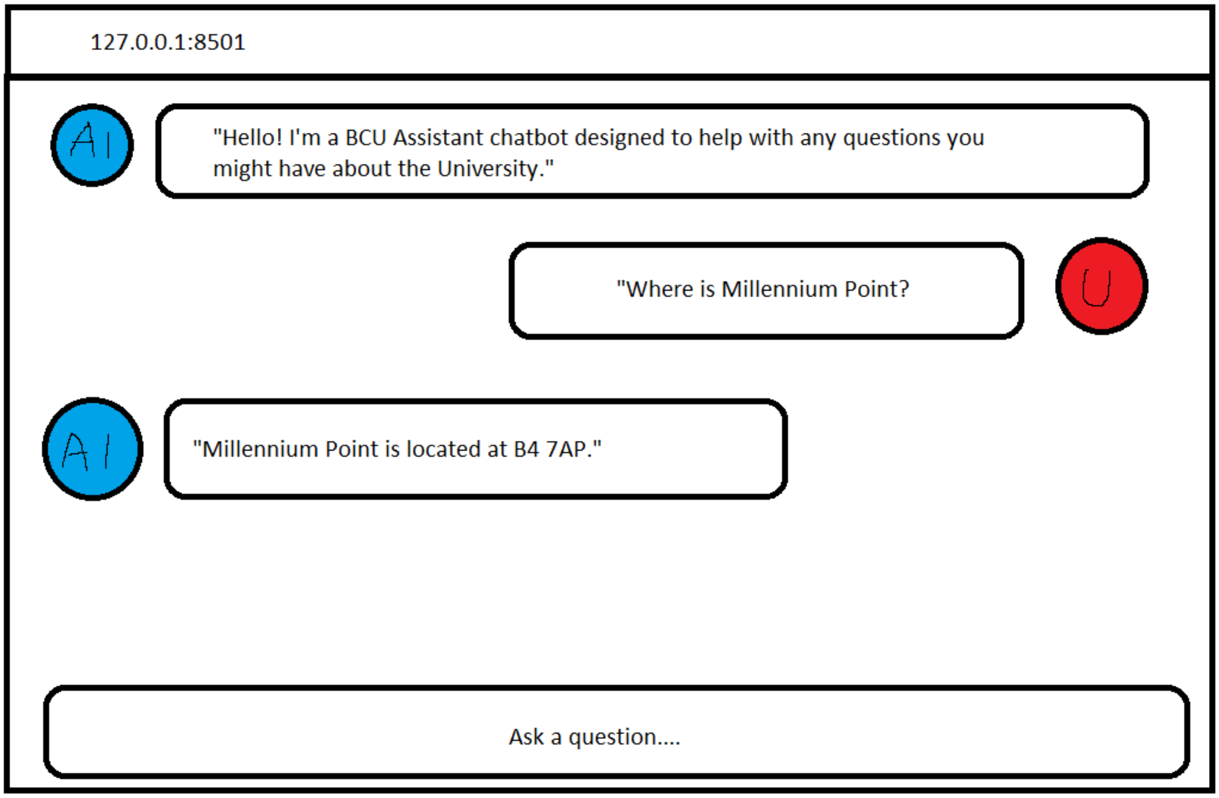

Figure 3.4: An early visualisation of a conversation.

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. This version of the chatbot would be hosted locally, though with additional time and resources it could instead be hosted on a dedicated domain or as part of another service.

Figures 3.5 and 3.6 depict how the backend of the chatbot should work, including the storage of BCU data and how the chatbot will query it.

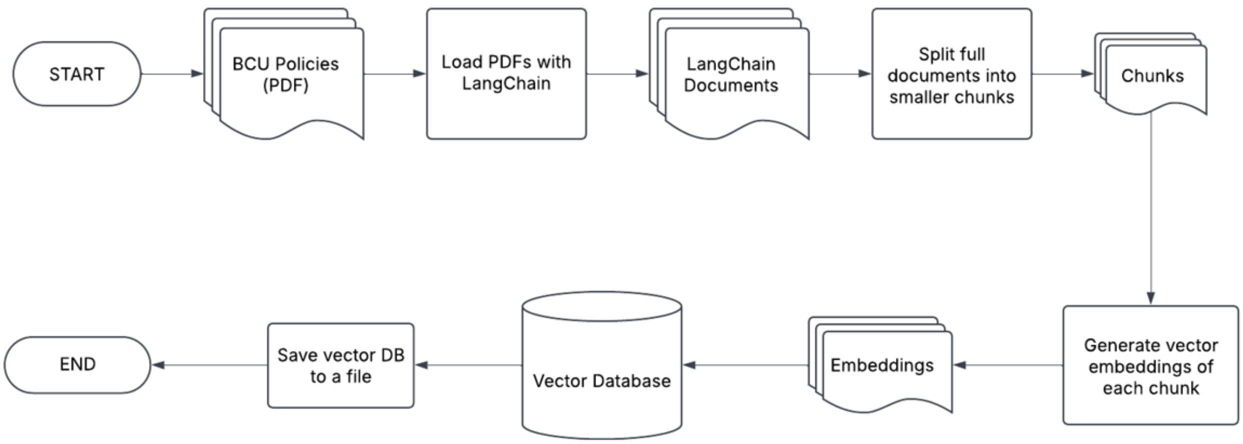

Figure 3.5: The flowchart for the PDF embedding procedure.

BCU's policies are publically available on their website (BCU, 2025), with policies valid during the 2024/25 academic year being used in this project. The planned use case for these policies was to download them locally and then process them using LangChain as a wrapper to split the PDFs into smaller chunks and embed them into a vector database. LangChain provides classes for direct PDF loading and conversion into its own "Document" format, and also provides functions to vectorise each chunk using any embedding model, with OpenAI's text-embedding-3-small being used.

After the embedded chunks are stored, the database generated from this process is stored locally. This means that the chatbot will operate both locally and on the cloud, with vector similarity searches being executed on the chatbot host device and the proprietary gpt-4o-mini LLM running on an OpenAI server.

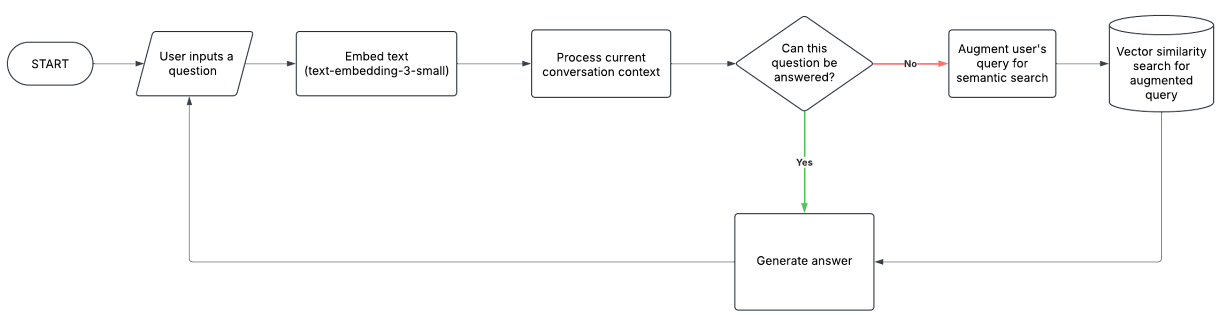

Figure 3.6: The flowchart for the answer generation procedure.

Users will input prompts as natural language to the chatbot, which will then vectorise them using the same embedding model used to embed the BCU policies. This is a necessary procedure as different embedding models will yield different outputs, meaning that if the same model was not used for the storage of policies and conversion of user queries, answers would be completely incorrect.

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. To do so, the chatbot will review the existing conversation when it receives a prompt. If it already has the answer to the user's question in previously retrieved context, there will be no need to perform another search and duplicate pre-existing context.

Another way that the chatbot may choose not to query the database is if the user's prompt can already be answered by the LLM. This may occur with very simple queries, such as "Hello!" or "My name is Lewis." In either of these example scenarios, the chatbot should not query the database, as the LLM should be capable of responding to generic greetings and remembering a user's name. Additionally, answers that do not require similarity searches should be answered considerably quicker, and would make the user experience feel much smoother if they are not having to wait a long time between responses as previously detailed in Section 3.3.1.

This decision functionality would be provided by LangGraph, and is detailed further in Section 3.4.3.

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 3.7.

```python
# Loads every PDF from the directory.
def loadPDFs():
    # 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(pdfPath,  glob = "*.pdf",
                             loader_cls = PyPDFLoader)
    documents = loader.load()
    return documents

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

    # 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 3.7: 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 saveToFAISS(chunks):
    # Clear out the database first if it exists.
    if os.path.exists(dbPath):
        shutil.rmtree(dbPath)

    # 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 = dbPath)

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

Figure 3.8: Code used to embed and store the chunks into a FAISS DB.

Firstly, any existing database in the specified directory is cleared using the rmtree method of shutil 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.

```
PS C:\Users\Lewis\Documents\University\CMP6200\CMP6
Split 220 documents into 385 chunks.
Saved 385 chunks to VectorStores/FAISS-HugeChunks.
```

Figure 3.9: The PDF to vector store conversion code successfully running.

3.4.3 Backend code

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 3.10.

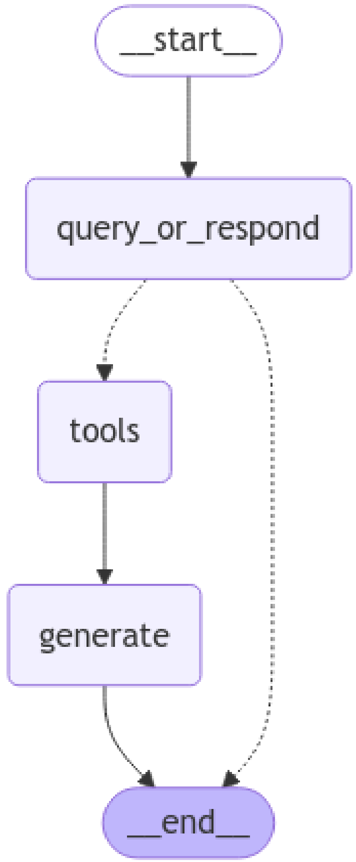

Figure 3.10: The graph for the chatbot.

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, 2025d), especially considering that 4o-mini performs suitably for the task at hand.

Figure 3.11 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.

```
# 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.
dbPath = "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 = dbPath,
                      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 3.11: Establishing prerequisite variables for the chatbot.

Thorough experimentation with the chunk sizes previously discussed in Figure 3.7 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 3.12: 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 3.10 by calling on the retriever tool denoted in Figure 3.13.

```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 3.13: 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.

```
# 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 3.14: Code used for the 'generate' node (1/2).

The 'generate' function is the largest function of the chatbot, and is split across Figures 3.14 and 3.15. 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.

```python
# 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 3.15: 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.

```python
# 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 3.16: 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 Frontend code

The chatbot's frontend GUI was created using the Streamlit package, which allows for the creation of visually appealing, dynamic and responsive web apps through simple Python code (Streamlit, 2021). Figure 3.17 shows the initial variables set for the page's structure.

```python
# Use Streamlit's wide layout, which will use the whole screen space rather than a small
# centre column. Also gives the page a title and icon which is shown in the
# browser tab view.
st.set_page_config(layout = 'wide', page_title = 'University Artifically Intelligent Chatbot',
                   page_icon='📚')

# When the UI first opens, this is the first message that will already be in the chat.
defaultMsg = "Hello! I'm an assistant chatbot designed to help answer any questions you have about BCU."

# If there's no message history (app just opened, or history just cleared), show the default message.
if 'message_history' not in st.session_state:
    st.session_state.message_history = [AIMessage(content=defaultMsg)]

# Organises the app so that the left and right columns are smaller than the main chat UI.
clearHistBtn, chatHist, queryLog = st.columns([1, 8, 1])
queries = []
```

Figure 3.17: The variables defining the frontend page's structure.

Streamlit provides many helpful methods and variables for creating the frontend GUI, allowing for the page layout to be quickly defined using 'set_page_config', where the wide layout was used to ensure the app fills the screen, and the title and an icon for the browser tab are also provided.

When the app initially opens, it would by default open to a mostly blank page. To remedy this, a default message was provided explaining what the chatbot is, which will always show as the first message in the conversation. This is performed by validating that the current session's message history is empty before inserting the default message.

Following this, three columns are set up on the page: a small column containing a button to clear the current message history, a large column containing the main chat UI, and another small column showcasing the queries being given to the FAISS database by the chatbot, which would be none at the time of startup, so an empty list is defined.

After the page's structure is defined, the functionality of each of the three columns is established as depicted in Figures 3.18, 3.19 and 3.20.

```
# In the left column, a button is placed that clears the chat history when clicked.
# It resets the entire conversation back to the original "Hello!" prompt.
with clearHistBtn:
    if st.button('Clear history'):
        st.session_state.message_history = [AIMessage(content=defaultMsg)]
        queries = []

# In the right column, the queries being sent by the retrieval agent to the DB are logged.
with queryLog:
    st.title("Agent's queries to vector DB")
    st.write(queries)
```

Figure 3.18: Defining the left and right column functions.

The left column ('clearHistBtn') has a button placed in it which will clear the current conversational history when clicked, and also empty the list of queries to the database. This button effectively resets the app to its initial opening state.

The right column ('queryLog') outputs all the agent's queries which it has given to the vector database. The queries themselves are retrieved within the main column of the app, 'chatHist'.

```
# The main UI is in this column.
# The user inputs their prompt into a text box which is then sent off to the chatbot.
with chatHist:
    userInput = st.chat_input("Ask anything about BCU!")

    # When an input is confirmed, add it to the conversation history and get a response.
    if userInput:
        st.session_state.message_history.append(HumanMessage(content=userInput))

        # Prompts the LLM and RAG tool with the current conversation history.
        response = graph.invoke({
            'messages': st.session_state.message_history
        })

        # The response that gets returned still contains the whole history,
        # but also appends the latest LLM response. Therefore, make that the new
        # message history.
        st.session_state.message_history = response['messages']
```

Figure 3.19: Defining the main column function (1/2).

The user's input box is established, and when an input is given, it is added to Streamlit's active conversation history, which is then used to invoke the chatbot's LangGraph. When the chatbot responds, it returns the entire conversation history once again, this time containing the chatbot's latest response. Therefore, this is used to overwrite the existing Streamlit history.

With the core prompting functionality set, the Streamlit UI for this column then needed to be established as depicted in Figure 3.20.

```python
# Shows the running conversation history after any message is sent.
# Iterates over the whole message history, showing HumanMessages for the user,
# and AIMessages for the LLM.

for i in range(1, len(st.session_state.message_history) + 1):
    currentMsg = st.session_state.message_history[-i]

    # ? If the message was written by the LLM:
    if isinstance(currentMsg, AIMessage):
        # The tool call is an AIMessage with no content, as the call is instead done within the metadata,
        # so it would show a blank box. To fix this, the message is checked to see if it's blank first.
        if currentMsg.content != "":
            # ? Use a robot profile picture.
            message_box = st.chat_message('assistant')
            message_box.markdown(currentMsg.content)

        # For logging purposes, the tool call used in the message is stored.
        toolCalls = currentMsg.additional_kwargs.get("tool_calls")

        if toolCalls is not None:
            # The tool call is an array containing a dictionary of dictionaries, but I'm only looking
            # for the query given to the vector DB, stored in function.arguments.
            toolQuery = toolCalls[0].get("function").get("arguments")

            # Strangely, the query is formatted like a dictionary but is actually a string,
            # so I remove the text "query":" and also the closing speech marks.
            queries.append(toolQuery[10 : len(toolQuery)-2])

    # ? Alternatively, if the user wrote it:
    elif isinstance(currentMsg, HumanMessage):
        # ? Use a person profile picture.
        message_box = st.chat_message('user')
        message_box.markdown(currentMsg.content)
```

Figure 3.20: Defining the main column function (2/2).

By iterating over each message in the conversation, logic is established to determine whether the message should use a human profile picture or a robot profile picture, representing the user and chatbot respectively. Initially, a bug was present where the chatbot would send two messages at a time, with one message being blank. It was previously established that this was due to the 'query_or_respond' node invoking the retrieval tool, which it does in the form of a blank message with metadata. However, Streamlit would still detect this blank message and output it. Therefore, the chatbot's message is checked to ensure it is not blank before being displayed. If it is blank (i.e. a tool call), it will not be displayed to the user, as there would be no reason for this, and it would only serve to cause confusion.

Logging the tool calls for the 'queryLog' column proved to be somewhat challenging. The metadata of an AIMessage ('additional_kwargs') is structured like a dictionary, though cannot entirely be parsed in the same way. Therefore, after parsing as far as possible using the standard dictionary 'get' method, extracting the actual query itself is performed with basic string manipulation, to exclude irrelevant details from the query string. This sanitised string is then logged as a query which will be automatically detected by the 'queryLog' column and output.

3.4.5 Running the chatbot

After ensuring all prerequisite packages are installed, the chatbot itself is run through the 'Streamlit Frontend' file. Because the file is executed by Streamlit rather than the generic Python interpreter, it is run slightly differently as depicted in Figure 3.21.

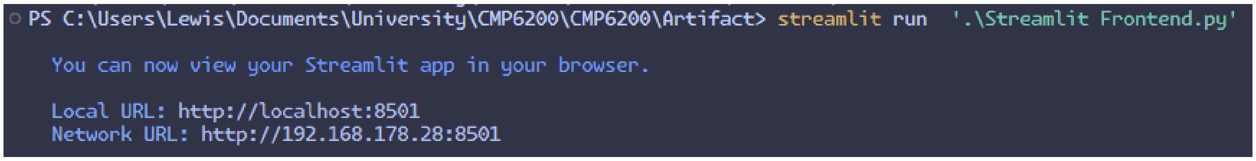

Figure 3.21: Running the chatbot from the terminal with Streamlit.

Then, by navigating to the given URL, the chatbot itself can be accessed.

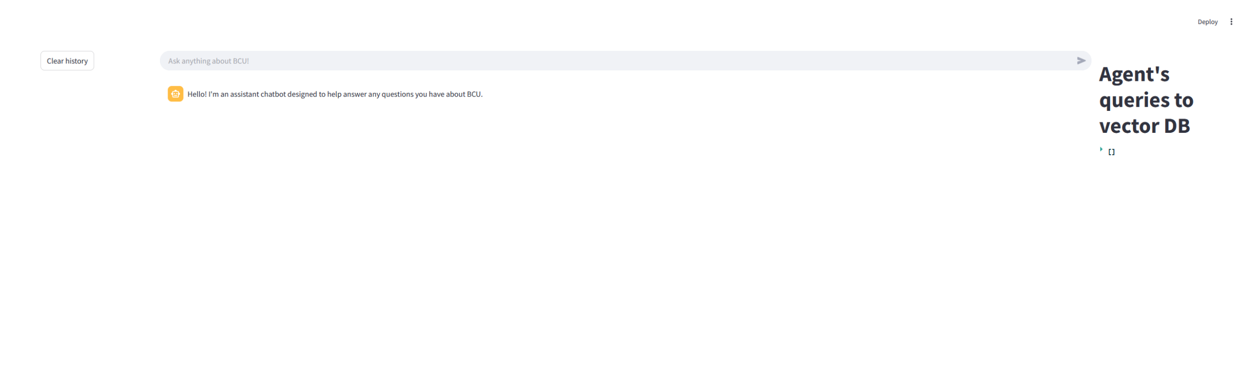

Figure 3.22: The chatbot's GUI at its initial state.

The previously defined columns are clearly visible: 'clearHistBtn' as the 'Clear history' button on the left, 'chatHist' as the main central column with the text box and default chatbot message, and 'queryLog', currently showing the empty list of queries.

The chatbot can now be queried by simply giving a prompt in the main text entry field as shown in Figure 3.23, which is slightly zoomed in for demonstrative purposes.

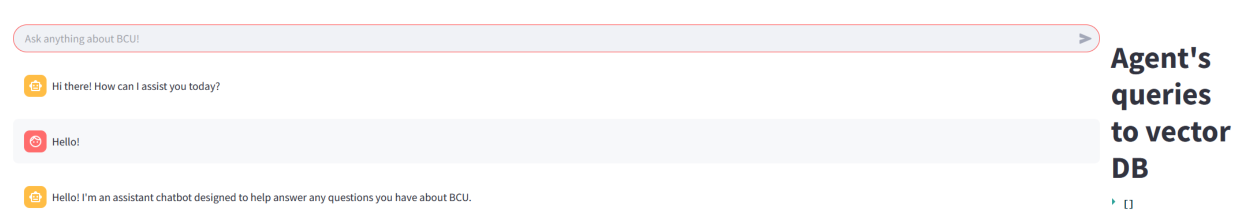

Figure 3.23: Prompting the chatbot with 'Hello!'.

This simple query demonstrates the conditional branch in LangGraph. With 'Hello' being such a simple query, the chatbot deems it unnecessary to retrieve context for, as it can already provide a suitable answer. This can be seen from the 'queryLog' column still remaining empty. Instead, the chatbot can now be asked a BCU-related question which it will require context for.

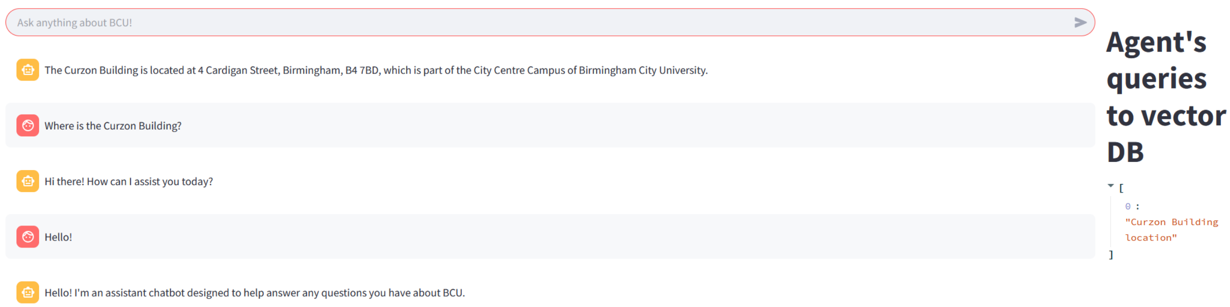

Figure 3.24: Prompting the chatbot with 'Where is the Curzon Building?'.

The chatbot is able to answer correctly with no hallucination, as it performed a search for 'Curzon Building location' on the FAISS database as shown in the 'queryLog' column. From this, it was able to parse the retrieved chunks, locate the address, and return it to the user. To expand on this functionality, another relevant question can be asked:

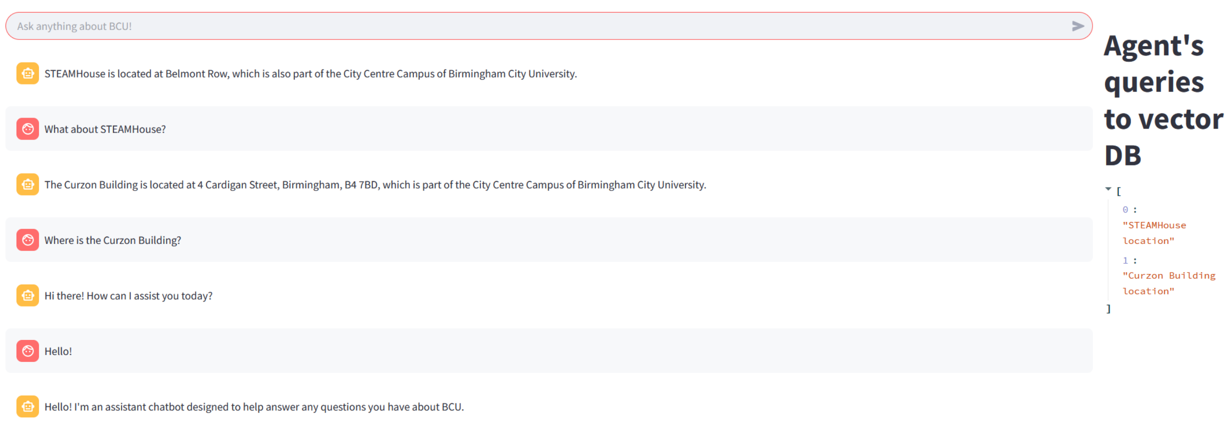

Figure 3.25: Prompting the chatbot with 'What about STEAMHouse?'.

Because the chatbot is invoked with the conversation history, it is able to interpret what the user means by 'What about', referring to another building's location. As such, it uses another similar query on the vector database, this time retrieving information for the STEAMHouse building, which it is able to successfully return to the user. Additionally, its response states 'which is also part of the City Centre Campus', showing that the previous response it gave was factored into its new response, as per the use of 'also'.

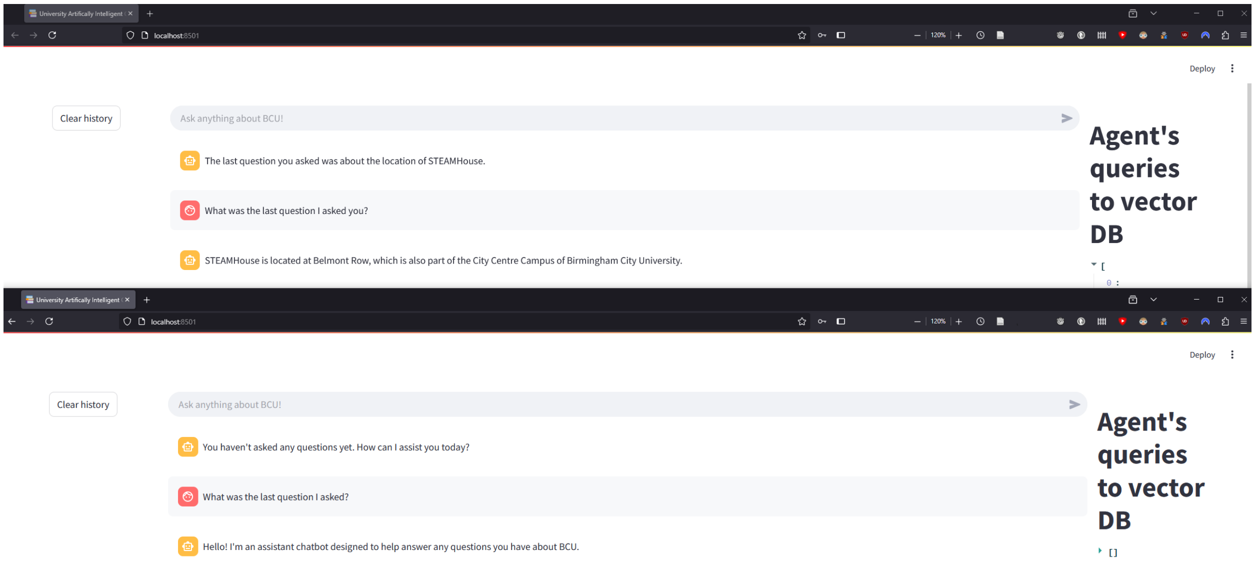

Figure 3.26: Two separate chatbot instances, which run without memory overlap.

Figure 3.26 depicts how multiple instances of the chatbot can run at the same time without influencing each other, which is a key element in terms of user privacy. In the figure, it can be seen that two separate windows of the chatbot are running, where one contains the conversation in Figure 3.25 and another that had just been started. Both chatbots are asked what the last question asked of them was, which the new instance cannot answer, but the old instance can. This shows how the second chatbot instance runs independently of the first.

Evaluation

In this chapter, the produced artefact will be evaluated against its original requirements. In addition, the development process itself will be reflected upon to identify where issues arose and could have been prevented.

4.1 Methodology

To properly evaluate the chatbot, it is best to use direct evaluation metrics. While reviewing literature in Section 2.1.3, an evaluation platform known as DeepEval was considered, with the ability to use an LLM as a judge of other LLMs (DeepEval, 2024), known as 'G-Eval', originally created by Liu et al. (2023a). This means that the metric is directly programmable using a natural language system prompt as seen with the chatbot itself.

To evaluate the chatbot, an evaluation dataset of questions and their expected answers was provided, with the chatbot's actual response being compared to the expected answer which is known to be true (referred to as a 'golden answer'.). It is common practice when using G-Eval to use a superior LLM for evaluation than the one that originally generated the answer. Therefore, for G-Eval, gpt-4o was used, making the evaluation process the most expensive part of this project's development cycle.

4.1.1 Dataset and evaluation metrics

Table 4.1 depicts the golden dataset[1] used to evaluate the chatbot's performance, produced by myself after my own in-depth analysis of each policy in the vector store.

ID	Question	Expected answer
1	What happens if I submit my assignment 3 days late?	If an assignment is submitted 3 days late, your mark will be reduced by 10%.
2	What happens if I submit my assignment 3 **minutes** late?	If an assignment is submitted 3 minutes late, it is not considered a late submission, and your mark will not be reduced.
3	What is an EC claim?	An Extenuating Circumstances claim can be made by a student if there are circumstances that affect their ability to submit assessments on time, complete assessments to a good standard or attend in-person assessments.

[1] BCU's extenuating circumstances policy was updated during the development of the chatbot to where the fourth question's answer is not entirely correct. However, the chatbot is unaware of this, and the objective was to assess the chatbot's performance against its knowledge base sourced in February 2025, so this is not a major issue.

4	What circumstances will be accepted as extenuating circumstances?	Serious short-term illness or injury, worsening of an ongoing illness or disability, symptoms of a harmful infectious disease, death or significant illness of a close family member or friend, unexpected caring responsibilities, significant personal or family crises, witnessing or experiencing a traumatic incident, a crime which has had a substantial impact on you, an accommodation crisis such as eviction.
5	When must I enrol?	Students must enrol at the start of their programme and enrol for each level by the Friday of week four from the start date of their course unless a Break in Study has been approved.
6	What is the pass mark for a module?	For an undergraduate course, the pass mark is 40%. On a postgraduate course, it is instead 50%.
7	What happens if I fail a module?	The first time you fail a module, you can be re-assessed for failed assessments, which is known as a resit. Your grade will be capped at the pass mark. You cannot be reassessed for assessments that you passed.
8	What are the degree classifications?	Achieving an average of 70% or above grants you first-class honours. 60-69% is an upper second (2:1), 50-59% is a lower second (2:2), 40-49% is third-class honours, and anything below 40% is a fail.
9	How many BCU students are there?	BCU has 31,300 students as of 2022-23.
10	What is BCUSU?	The Birmingham City University Student Union represent you as a student, and work together with the university to make change. They can help with various academic topics and offer societies to help people make friends.

Table 4.1: The golden dataset for chatbot evaluation.

The golden dataset was then stored as a nested array of question/answer pairs where for each item in the array, index 0 was the question and index 1 was its respective answer.

```python
# Nested list of the question/answer pairs to be converted to DeepEval's LLMTestCases.
qaPairs = [
    ["What happens if I submit my assignment 3 days late?", "If an assignment is submitted 3 days late, your mark will be reduced by 10%."],
    ["What happens if I submit my assignment 3 minutes late?", "If an assignment is submitted 3 minutes late, it is not considered a late
    submission, and your mark will not be reduced."],
    ["What is an EC claim?", "An Extenuating Circumstances claim can be made by a student if there are circumstances that affect their ability to
    submit assessments on time, complete assessments to a good standard or attend in-person assessments."],
    ["What circumstances will be accepted as extenuating circumstances?", "Serious short-term illness or injury, worsening of an ongoing illness or
    disability, symptoms of a harmful infectious disease, death or significant illness of a close family member or friend, unexpected caring
    responsibilities, significant personal or family crises, witnessing or experiencing a traumatic incident, a crime which has had a substantial
    impact on you, an accommodation crisis such as eviction."],
    ["When must I enrol?", "Students must enrol at the start of their programme and enrol for each level by the Friday of week four from the start
    date of their course unless a Break in Study has been approved."],
    ["What is the pass mark for a module?", "For an undergraduate course, the pass mark is 40%. On a postgraduate course, it is instead 50%."],
    ["What happens if I fail a module?", "The first time you fail a module, you can be re-assessed for failed assessments, which is known as a
    resit. Your grade will be capped at the pass mark. You cannot be reassessed for assessments that you passed."],
    ["What are the degree classifications?", "Achieving an average of 70% or above grants you first-class honours. 60-69% is an upper second (2:1),
    50-59% is a lower second (2:2), 40-49% is third-class honours, and anything below 40% is a fail."],
    ["How many BCU students are there?", "BCU has 31,300 students as of 2022-23."],
    ["What is BCUSU?", "The Birmingham City University Student Union represent you as a student, and work together with the university to make
    change. They can help with various academic topics and offer societies to help people make friends."]
]

testCases = []
```

Figure 4.1: The golden dataset as represented in the evaluation script.

To begin testing the chatbot, the 'testChatbot' function was created to invoke the chatbot with a sample conversation that is the same as if the Streamlit app had been opened for the first time, with the only messages being the introductory message and the question from the golden dataset being tested.

```python
def testChatbot(question):
    # I'm not aiming to test the conversational memory, but rather it's retrieval ability uninfluenced by existing conversation.
    # Therefore, every time the chatbot is invoked, it'll only know this message, which is the same as when it is initially opened in Streamlit.
    evalConversation = [AIMessage(content="Hello! I'm an assistant chatbot designed to help answer any questions you have about BCU."),
                        HumanMessage(content = question)]

    response = graph.invoke({
        "messages": evalConversation
    })

    # Returns the most recent message, which is the chatbot's response.
    return response["messages"][-1].content
```

Figure 4.2: The testChatbot function to invoke the chatbot with a new conversation.

The question/answer pairs were then used to generate DeepEval's own 'LLMTestCase' classes for each item, with the input being the question, expected output being the golden answer, and the actual output using the testChatbot function to invoke the chatbot.

```python
# Creates DeepEval's LLMTestCases for each question and answer pair for input and expected output.
# For the actual output, the chatbot is used.
for qa in qaPairs:
    testCases.append(
        LLMTestCase(input = qa[0],
                    actual_output = testChatbot(qa[0]),
                    expected_output = qa[1])
    )
```

Figure 4.3: Iteratively creating DeepEval LLMTestCases for each question/answer pair.

The test cases were then added into a DeepEval 'EvaluationDataset' for later use. Then, the GEval metric was defined.

```
# Create an evaluation dataset composed of the test cases.
dataset = EvaluationDataset(test_cases = testCases)

correctness_metric = GEval(
    name="Correctness",
    criteria = "Determine whether the actual output is factually correct based on the expected output. Any additional scenarios or details not present in the expected output are OK.",
    model="gpt-4o",
    evaluation_params=[
        LLMTestCaseParams.EXPECTED_OUTPUT,
        LLMTestCaseParams.ACTUAL_OUTPUT
    ]
)

# Evaluate the dataset using the defined GEval metric.
evaluation_results = evaluate(dataset, metrics = [correctness_metric])
```

Figure 4.4: Creating the evaluation dataset and GEval metric.

The criteria field of the GEval metric acts as the system prompt for the gpt-4o model. In the criteria, the LLM is told to evaluate the 'correctness' of the actual output by comparing it against the expected output. An additional note was added to ensure that the chatbot was not penalised for giving more information than strictly necessary, as GEval would originally fail some test cases as the actual output was sometimes more informative than the expected output.

Using an LLM for evaluation in this way allows for automated testing to be significantly easier and somewhat more reliable to perform, as there is no need for implementations such as word matching. This is because the LLM can automatically infer the semantic similarities between the expected answer and the intended answer, known as Semantic Answer Similarity.

Following all of these initial steps, the primary DeepEval 'evaluate' function is called using the created EvaluationDataset and GEval metric.

4.2 Baseline systems

Multiple vector stores were created for the chatbot, each using a differing chunk size and overlap. A total of four datasets were made, all of which used FAISS as the backend engine and PyPDFLoader to load each PDF prior to chunking.

These vector stores were:

- FAISS
  - The original vector store used throughout early development.
  - Chunk size: 1000
  - Overlap: 200
- FAISS-SmallChunks
  - As suggested by its name, used a lower chunk size and overlap than others.
  - Chunk size: 500
  - Overlap: 100
- FAISS-BigChunks

  - Used bigger chunks and overlap than the original vector store.
  - Chunk size: 1500
  - Overlap: 300
- FAISS-HugeChunks
  - The vector store used in the final product.
  - Chunk size: 2000
  - Overlap: 500

4.3 Results

Table 4.2 depicts how the chatbot answered each question for each vector store. Refer to Table 4.1 for the corresponding question to each ID.

ID	FAISS	SmallChunks	BigChunks	HugeChunks
1	Wrong	Wrong	Wrong	Correct
2	Wrong	Wrong	Wrong	Wrong
3	Correct	Wrong	Correct	Correct
4	Correct	Questionable	Correct	Correct
5	Wrong	Wrong	Wrong	Correct
6	Correct	Correct	"I don't know"	Correct
7	Questionable	Wrong	Wrong	Correct
8	Wrong	Wrong	Correct	Wrong
9	Wrong	Wrong	Wrong	Correct
10	Correct	Correct	Correct	Correct
	40%	**20%**	**40%**	**80%**

Table 4.2: The performance of the chatbot with each vector store.

G-Eval assigns a score to each of the chatbot's answers to the golden dataset questions, where scores under 0.5 are deemed to be incorrect. However, answers marked as 'Questionable' were answers that were technically correct after manual inspection despite G-Eval labelling them as incorrect. This showcases in part the occasional unreliability of LLMs even in judging contexts such as G-Eval.

The questionable judgments were with Questions 3 and 4 on the smallest chunk-size vector store, depicted in Figures 4.5 and 4.6.

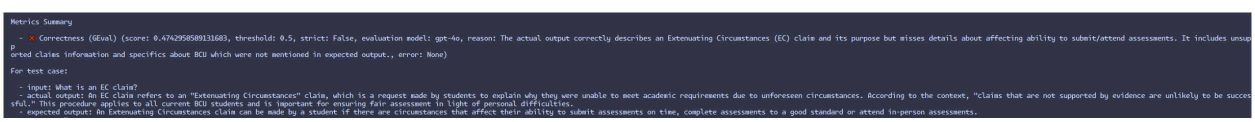

Figure 4.5: The G-Eval judgment of the chatbot's response to Question 3 using the FAISS-SmallChunks vector store.

The answer provided here was very narrowly under G-Eval's 0.5 judgment threshold meaning that it was deemed to be incorrect, with the gpt-4o evaluator LLM stating that unsupported information was introduced. However, upon manual inspection, the information provided by the chatbot in response to this question is correct.

There is a similar occurrence in the response to Question 4 using this vector store, as depicted in Figure 4.6.

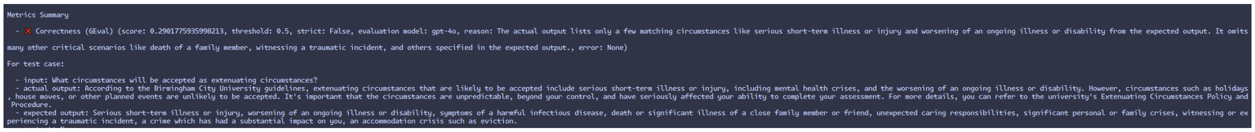

Figure 4.6: The G-Eval judgment of the chatbot's response to Question 4 using the FAISS-SmallChunks vector store.

G-Eval uses the expected answer verbatim, meaning that with the long list of options provided in the expected answer to Question 4, G-Eval expected the chatbot to respond with every option as stated in its response: "It omits many other critical scenarios ... specified in the expected output". While this is expected behaviour of G-Eval, it proves that manual evaluation can still be more suitable in specific scenarios, as the answer given by the chatbot here was manually deemed to be of good quality despite not listing every possible scenario.

There were also two more noteworthy observations, both with the FAISS-BigChunks vector store, with its response to Question 6 being "I don't know", and the response to Question 8 being the only correct response of all four vector stores, even including the best-performing vector store.

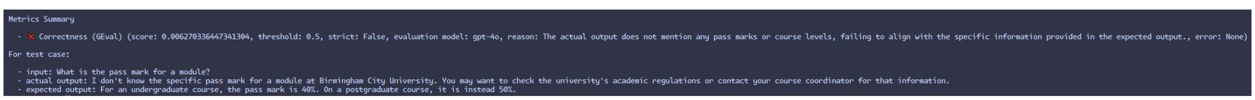

Figure 4.7: The G-Eval judgment of the chatbot's poor response to Question 6 using the FAISS-BigChunks vector store.

The chatbot's total failure to answer the question using this vector store signifies a potential issue in the retrieval tool, yet every other vector store could successfully retrieve the passing grades in question. It is possible that while the PDFs were being split into chunks for this vector store specifically, the information required for this question was split over multiple chunks despite the large size and overlap, meaning the similarity search could not find a suitable answer. The chatbot's system prompt states that it must respond with "I don't know" if it is unable to retrieve relevant context, and this is displayed clearly within this failure.

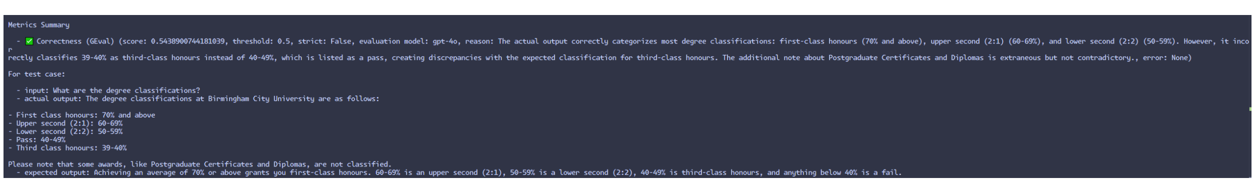

Figure 4.8: The G-Eval judgment of the uniquely good response to Question 8.

Intriguingly, this particular vector store was the only one where the information needed to answer the question about degree classifications was successfully retrieved for a response. It is possible that the question itself may have been misinterpreted when performing semantic similarity searches on the other vector stores, leading them to instead fail on this question as shown in Figure 4.11.

4.3.1 Best-performing system

Images of the G-Eval outputs for each incorrect answer for all vector stores can be found in the project's Github repository, though this section will focus on the best-performing vector store, which was FAISS-HugeChunks.

```
=========================================================================

Overall Metric Pass Rates

Correctness (GEval): 80.00% pass rate

=========================================================================
```

Figure 4.9: Overall G-Eval results against the "FAISS-HugeChunks" vector store.

On the ten test cases on various academic policies and information used in evaluating the chatbot, 80% were answered correctly according to G-Eval. Figures 4.10 and 4.11 depict the two queries where the chatbot failed to give a suitable answer.

```
Metrics Summary

  - ✗ Correctness (GEval) (score: 0.39708257277880243, threshold: 0.5, strict: False, evaluation model: gpt-4o, reason: The actual output correctly states that a 3-minute late submission does not fall under penalties, but it introduces contradictory details about penalties for submissions between 1 and

24 hours late and non-submission definitions., error: None)

For test case:

  - input: What happens if I submit my assignment 3 minutes late?
  - actual output: If you submit your assignment more than 1 hour after the published deadline, it will be recorded as a non-submission, and you will receive a mark of 0. According to the Late Submission of Assessment Policy, submissions made between 1 and 24 hours late will incur a 5% reduction in the
original mark. However, since your submission is only 3 minutes late, it would fall under the non-submission rule if it's more than 1 hour late. Be sure to check the specific guidelines in your Module Guide for any additional details!
  - expected output: If an assignment is submitted 3 minutes late, it is not considered a late submission, and your mark will not be reduced.
```

Figure 4.10: The first incorrect answer, with the chatbot answering incorrectly.

This question was answered incorrectly due to the chatbot's misinterpretation of the related policy. Work submitted **up to** 1 hour after a deadline does **not** receive any grade penalty, though the chatbot likely read that work submitted **between 1 hour and 24 hours** after a deadline **does** incur a penalty. As such, this misinterpretation lead to the question being answered incorrectly.

```
Metrics Summary

  - ✗ Correctness (GEval) (score: 0.046060901631594355, threshold: 0.5, strict: False, evaluation model: gpt-4o, reason: The actual output describes postgraduate classifications which contradicts the undergraduate classifications in the expected output. Both use similar percentage ranges but differ in
t
itles; Expected mentions 'first-class honours', 'upper second', etc., Actual mentions 'Distinction', 'Merit', etc., error: None)

For test case:

  - input: What are the degree classifications?
  - actual output: At Birmingham City University, the degree classifications for postgraduate awards are as follows:

- 70% and above: Distinction
- 60-69%: Merit
- 50-59%: Pass
- 40-49%: Fail

Please note that Postgraduate Certificates (PGCert) and Postgraduate Diplomas (PGDip) are not classified.
  - expected output: Achieving an average of 70% or above grants you first-class honours. 60-69% is an upper second (2:1), 50-59% is a lower second (2:2), 40-49% is third-class honours, and anything below 40% is a fail.
```

Figure 4.11: The second incorrect answer, with incorrect information retrieval.

The chatbot retrieving information that isn't directly relevant for this question implies an error in the retrieval tool. This could likely be due to the format of each policy document, with the information requested here (degree thresholds) being stored in a table, depicted in Figure 4.12.

Classification of awards

7.6.5 The table below shows the classification bands for the University's awards.

Classification band	Postgraduate awards[1]	Honours degree Integrated master's	Foundation degree HNC/HND DipHE
70% and above	Distinction	First class honours	Distinction
60-69%	Merit	Upper second (2:1)	Merit
50-59%	Pass	Lower second (2:2)	Pass
40-49%	Fail	Third class honours	Pass
Below 40%	Fail	Fail	Fail

Figure 4.12: The snippet of the Academic Regulations that should have been referenced. (BCU, 2025)

The 'PyPDFLoader' class in LangChain, which was used when storing all University data, can sometimes misinterpret tables. This may in turn have created issues with the semantic search performed on the FAISS DB, leading to this question going unanswered as the search was unable to identify each degree classification.

4.4 Discussion

This section aims to discuss the evaluation results as well as the project's overall success against its initial stated requirements.

4.4.1 Evaluation results

Vector store performance

The vector store with the best performance against the training dataset being the one with the largest chunk size and overlap (FAISS-HugeChunks) was expected behaviour, as the chatbot is able to obtain more information for each database query performed. The reason that the other three options were considered was to reduce the token usage of the chatbot, which in turn would speed up responses and reduce prompt cost.

The significant failures of the smallest-chunk vector store (FAISS-SmallChunks) were unexpected, however. It appears as though semantic searches performed on the vector store when each chunk stored within it is small work poorly or not at all in many cases.

Limitations of G-Eval

While the FAISS-SmallChunks vector store did perform the worst according to G-Eval, the analysis of the reported incorrect answers in Figures 4.5 and 4.6 indicate that G-Eval's results cannot be taken entirely at face value, and should perhaps instead act as a guideline. The two answers deemed to be questionably judged were found to be correct after manual verification, even if they did lack some minor information.

Therefore, as initially theorised in the literature review, manual verification does indeed appear to be the most optimal evaluation solution for smaller LLM-based projects such as this one. However, when dealing with projects of a much greater scope, tools such as G-Eval can be of great use despite their occasional shortcomings.

4.4.2 Functional requirements

The functional requirements, and how they were met, were as follows:

- The chatbot must interpret and respond to answers in English.
  - This requirement was fully met with no particular involvement from myself. OpenAI's models can automatically interpret and respond with English text, as well as other languages, though other languages were not tested as I cannot personally verify them.
- The chatbot must accept text queries.
  - This was automatically met through the use of OpenAI models.
- The chatbot must respond using text.
  - This was automatically met through the use of OpenAI models.
- The chatbot must be accessible at all times.

  - When the Streamlit application is running, the chatbot can always be accessed by any device connected to the same network, as long as they connect with the IP and port which Streamlit specifies. While this is technically a constraint, it is not a strain on system resources to leave the Streamlit app running in the background indefinitely, meaning this requirement can be considered fulfilled.

- The chatbot must supply BCU-related information.
  - A vector database using FAISS was created containing a wide variety of BCU policies and miscellaneous information. Using this database, the chatbot had access to a retrieval tool which would perform a semantic search on the database to retrieve BCU information relating to the user's query.

- The chatbot must answer at least 75% of BCU-related queries correctly.
  - G-Eval reported an accuracy of 80% with the most optimal vector store against the manually produced golden answers. This is over 75%, though perhaps not by a satisfactory amount. A larger testing dataset would have helped to provide a more accurate picture in future.

- The chatbot must have a GUI for ease of use and accessibility.
  - Streamlit acts as the chatbot's frontend, providing a responsive and sleek UI that adapts to desktop web browsers and mobile devices. The UI is simple to understand and can be navigated with the Tab key on a keyboard for people unable to use a mouse.

- Multiple users must be able to use the chatbot at the same time.
  - Streamlit facilitates this functionality, creating isolated instances of the chatbot which do not interact with each other.

All functional requirements of the chatbot's original scope were met, producing a usable product which students can use to get BCU-related information at a satisfactory level of accuracy.

4.4.3 Non-functional requirements

The non-functional requirements, and how they were met (or failed to be met) are as follows:

- The chatbot should respond to queries within 10 seconds.
  - All conversations with the chatbot throughout testing would gather responses in fewer than 10 seconds. Queries that used RAG took significantly longer than those which did not. The overall 'feel' of the app could be made faster by allowing the LLM to stream text rather than output a full message, which will show the message being procedurally written rather than a buffer before a full message suddenly appears.

- The chatbot could allow for voice input and output.

  - This requirement **was not met.** This was mostly due to time constraints, as implementing this functionality would have taken substantial research that would likely not have been possible to perform while meeting project deadlines. Unfortunately, this does make the app less accessible, forcing users to be able to use a keyboard or have third-party voice software to input text.

- The chatbot could be deployed on an existing messaging service such as Teams.
  - This requirement **was not met.** As with the voice requirement, this would have taken additional research and a possible redesign of the app's backend to provide an API compatible with a messaging service chatbot. Given that Streamlit already provides a usable and modern UI, this requirement was instead considered unnecessary.

Only one of the three non-functional requirements was met due to time constraints which plagued development. Even without the implementation of these features, however, the chatbot is still a very usable product.

4.4.4 Development process reflection

Positives

All functional requirements stated in Section 3.3.1 for the final product, as well as the original aims and objectives of the project, were successfully met with a working chatbot with good accuracy on BCU-related topics being produced in a timely fashion.

Furthermore, with the project being a solo endeavour, a comprehensive understanding of the project management life cycle was obtained from conception to completion. As a result, I believe my problem-solving and decision-making skills have greatly improved.

Negatives

As identified previously in Section 3.2, the most significant limitation throughout the development process was the amount of time available. Over the course of the project's development, significant extenuating circumstances occurred leading to the lack of some desired features and lower quality of others.

Furthermore, balancing the production of this project alongside four other university modules simultaneously proved to be an arduous task that I was unable to efficiently solve to a level I would have preferred.

Cost proved to be a much lesser restriction than initially anticipated, due to the cost efficiency provided through the identification of OpenAI's lower-end models through thorough research.

However, the other limitations specified in Section 3.2 also played key roles of their own, though less significant than the time restrictions. Most notably of these was my own lack of experience with LLMs. Developing a product using a tech stack I was entirely unfamiliar with prior to development proved to be highly difficult.

Overall accomplishments

Overall, it is safe to say that the project can be considered a success, though it certainly is not without flaw. It was previously mentioned that the chatbot had an accuracy of 80% according to GEval, though this was only against a dataset of 10 questions. It would be much more suitable to expand this training dataset, alongside gathering actual user feedback.

Additionally, my own limitations in knowledge when it came to LangChain and LangGraph meant that I was unable to optimally refine the retrieval tool to work on the failed questions in time, and I was also unable to successfully implement a ReAct agent as researched in the literature review.

Despite the project's few failures, there were many successes. I have hugely increased my own knowledge of Python, LLMs, and RAG. These are three critical skills to have if to work in software development with recent trends of companies becoming more reliant on LLMs.

Furthermore, through developing the chatbot and performing the extensive research required, I believe my skills as an overall software developer have enhanced in a way that is not exclusive to Python; I have become much more aware of how to interpret API references and documentation, meaning that my ability to adapt to new tech stacks as seen in this project should now be a much faster process.

The achievement I am most proud of with the chatbot is the cost-saving effort of the conditional branches. Originally, the chatbot would query the database for every prompt given, even if it was something as simple as 'Hello!'. This would lead to 6,000 characters worth of university data which would not be relevant to the query being given to the LLM, wasting processing time and money through the greatly increased token cost of such a prompt, as well as resulting in a strange and irrelevant response that would only serve to confuse the user.

Conclusions

In conclusion, this report has thoroughly detailed the design and development process of a chatbot aimed to assist new university students to become acclimated to their new surroundings through an interactive conversational medium to learn key information about the university such as its governing policies.

A thorough literature review on surrounding topics and the current state-of-the-art was conducted to allow for the chatbot's development and documentation of the development process and final product. The final product itself performed to a suitable degree, answering 80% of its evaluation golden dataset questions correctly in relation to its expected answers, though improvements could have been made to increase this percentage.

The final product performs well, meeting and in some cases exceeding its stated functional requirements, though was not able to meet all of its non-functional requirements. Despite these shortcomings, the chatbot is functional and a very viable source of BCU-related information using natural language prompts for specific information rather than having to manually consult long policies and web searches.

Recommendations for future work

6.1 Bleeding-edge LLMs

As originally mentioned in the literature review, the LLM space is rapidly expanding and continued to do so even throughout this project's development. Throughout the main development process, the best identified model in terms of performance to cost was gpt-4o-mini. However, OpenAI has since gone on to release more intelligent models at similar price points, namely gpt-4.1-mini, and gpt-4.1-nano. The 4.1 mini model is 2.6x more expensive than 4o-mini ($0.40/1M input tokens) (OpenAI, 2025c), though its benchmark scores are only slightly lower than the full gpt-4o model for a fraction of its cost ($2.50/1M input tokens).

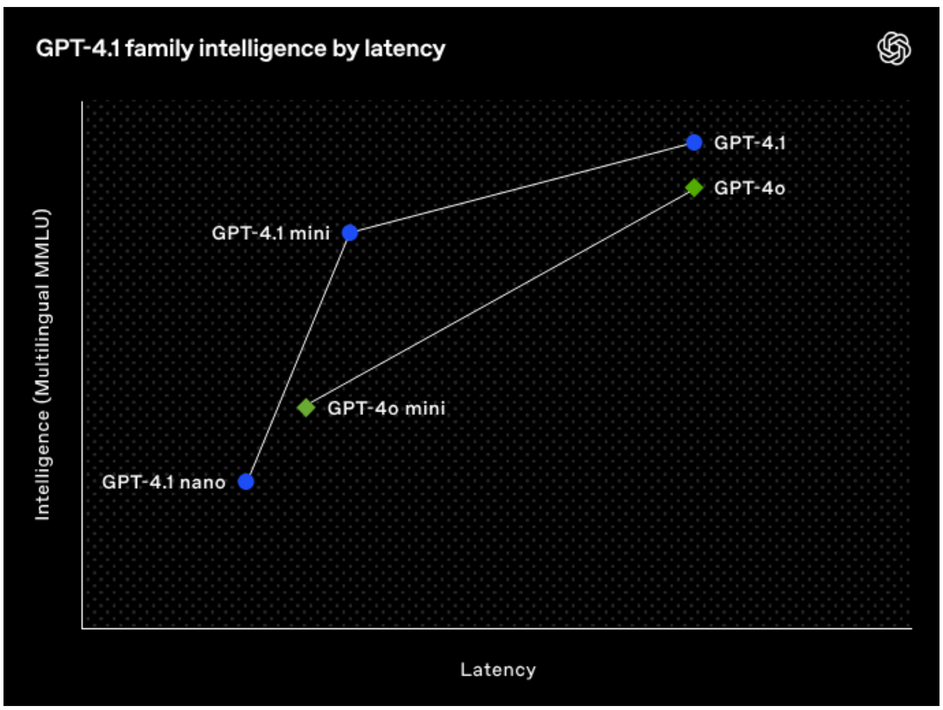

Figure 6.1: A relative performance chart of GPT models on the MMLU benchmark (OpenAI, 2025b).

gpt-4.1-nano is observed to have lower performance than 4o-mini, though this is also reflected in its extremely low latency and 50% reduced cost of only $0.10 per million tokens. The full GPT-4.1 model is shown to be extremely intelligent according to OpenAI's graph, though its pricing is simply too high in the scope of this project ($2/1M input tokens, $8/1M output tokens).

In summary, it would likely be beneficial for future similar works to utilise the latest developments in the LLM space, and remain observant of upcoming releases to capitalise as soon as they become available.

6.2 Agentic RAG

While the produced chatbot was able to answer the majority of questions correctly, it is possible that the implementation of an agentic RAG solution could have enhanced its results even further through an iterative corrective RAG cycle of answer evaluation. This was attempted during development, though repeatedly proved unsuccessful in this specific use case. To avoid jeopardising the overall project schedule, it was therefore not present in the final product.

Therefore, future similar works should plan extensively ahead of time for how they could implement such a solution to ensure maximum answer accuracy. However, they should be mindful that the latency will greatly increase if the chatbot repeatedly deems its own answers to be incorrect, and users may perceive this as a system failure.

6.3 User feedback

Uncontrollable external circumstances meant that user feedback could not be gathered at key stages throughout the product's iterative development cycle, with key observations such as those recorded in Chapter 4 being performed single-handedly. This is a key element of the Agile development process, and the failure to do so likely had negative consequences on the final product.

As such, any future works should always endeavour to obtain constant user feedback to ensure their work is of a constant good standard.

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).

DeepSeek (2025). *DeepSeek*. URL: https://chat.deepseek.com (visited on 24/03/2025).

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).

Google (2025). *Gemini – Chat to Supercharge Your Ideas*. Gemini. URL: https://gemini.google.com (visited on 24/03/2025).

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 2023a). *G-Eval: NLG Evaluation Using GPT-4 with Better Human Alignment*. DOI: 10.48550/arXiv.2303.16634. arXiv: 2303.16634 [cs]. URL: http://arxiv.org/abs/2303.16634 (visited on 24/03/2025).

Liu, Yang, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu and Chenguang Zhu (23rd May 2023b). *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.

M. Bran, A., S. Cox, O. Schilter, C. Baldassari, A.D. White and P. Schwaller (2024). 'Augmenting Large Language Models with Chemistry Tools'. In: *Nature Machine Intelligence* 6 (5), pp. 525–535. DOI: 10.1038/s42256-024-00832-8.

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).

Meta (29th Mar. 2017). *Faiss: A Library for Efficient Similarity Search*. Engineering at Meta. URL: https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/ (visited on 10/03/2025).

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 Conference 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 (2025a). *ChatGPT*. URL: https://chatgpt.com (visited on 24/03/2025).

OpenAI (2025b). *Introducing GPT-4.1 in the API*. URL: https://openai.com/index/gpt-4-1/ (visited on 23/04/2025).

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

OpenAI (2025c). *Pricing (April 23rd 2025)*. URL: https://platform.openai.com (visited on 23/04/2025).

OpenAI (2025d). *Pricing (March 23rd 2025)*. 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.

Oxford University (2021). *Supporting Academic Transition: A Focus on First-Year Teaching*. URL: https://www.ctl.ox.ac.uk/teaching-and-supporting-first-year-students (visited on 24/03/2025).

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.

Streamlit (14th Jan. 2021). *Streamlit • A Faster Way to Build and Share Data Apps*. URL: https://streamlit.io/ (visited on 24/03/2025).

TeamGantt (2025). *TeamGantt: The Easiest FREE Gantt Chart Maker Online*. URL: https://www.teamgantt.com/ (visited on 26/04/2025).

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.

Weaviate (5th Nov. 2024). *What Is Agentic RAG | Weaviate*. URL: https://weaviate.io/blog/what-is-agentic-rag (visited on 10/03/2025).

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.

Woo, J.J., A.J. Yang, R.J. Olsen, S.S. Hasan, D.H. Nawabi, B.U. Nwachukwu, R.J. Williams III and P.N. Ramkumar (2025). 'Custom Large Language Models Improve Accuracy: Comparing Retrieval Augmented Generation and Artificial Intelligence Agents to Non-custom Models for Evidence-Based Medicine'. In: *Arthroscopy - Journal of Arthroscopic and Related Surgery* 41 (3), 565–573.e6. DOI: 10.1016/j.arthro.2024.10.042.

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).

Higgins, Lewis (2025). *CMP6200 - GitHub*. URL: https://github.com/LewGoesBOOM/CMP6200 (visited on 28/04/2025).

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 a chatbot?* URL: https://www.ibm.com/topics/chatbots (visited on 28/10/2024).

IBM (2024e). *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).