Griffith College RAG Assistant:
A Retrieval-Augmented Generation System for Academic Document Querying

Group 1

Maxime VIEL | Benoît CATEZ

3167731 | 3168585

Submitted in partial fulfillment for the degree of

Bachelor of Science (Honours) in Computing

Griffith College Dublin
June 2025

Under the supervision of Dr. Waseem Akhtar

Disclaimer

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the Degree of Bachelor of Science (Honours) in Computing at Griffith College Dublin, is entirely my own work and has not been submitted for assessment for any academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed: **Date: Wednesday 3rd July 2025**

Acknowledgements

I would like to thank my supervisor Dr. Waseem Akhtar for his continued guidance.

I also extend thanks to the staff of Griffith College and all those who provided support throughout this project.

Table of Contents

Abstract

This project presents a Retrieval-Augmented Generation (RAG) assistant tailored to Griffith College's academic ecosystem. The assistant leverages state-of-the-art natural language processing to enable intelligent querying over static institutional documents, specifically the *Student Handbook* and *Admin Info* guide. The motivation was to reduce informational friction faced by students seeking administrative or academic details.

The system utilizes a combination of Hugging Face LLMs (Mistral-7B-Instruct), FAISS for semantic search, and Streamlit for a responsive frontend. Key components include document chunking, embedding generation via BAAI/bge-small-en, and a custom-built retrieval interface. The resulting assistant enables students to interactively access key college information via natural language — a major step toward smarter, student-centric support systems.

Chapter 1. Introduction

1.1 Area

This project applies to the principles of Retrieval-Augmented Generation (RAG) within the educational domain. Specifically, it enhances how Griffith College students interact with long, static documents by allowing them to ask natural language questions and receive contextually relevant answers. The project focuses on creating a student support assistant capable of understanding and retrieving data from administrative and academic documents.

1.2 Goals

The aim is to improve the accessibility and usability of information contained in college documentation. Rather than expecting students to manually search for large PDFs for information, the assistant provides a conversational interface. The goal is to make this system accurate, fast, and user-friendly — saving time for students and faculty while improving information transparency and engagement.

1.3 Overview of Approach

The assistant was built using a modern NLP stack. It begins with ingesting .docx and .pdf documents, which are split into contextually coherent chunks. These chunks are embedded into vector form using the BAAI/bge-small-en embedding model. The resulting vectors are indexed using FAISS for efficient similarity search. User questions are sent to a Mistral-7B LLM hosted via Hugging Face, which retrieves relevant context and generates an answer. All of this is wrapped in a Streamlit frontend that allows real-time interaction.

1.4 Document Structure

The rest of this document is as follows.

- Chapter Two provides a literature review of Retrieval-Augmented Generation systems and discusses prior work related to academic information systems.
- Chapter Three outlines the methodology and design principles used during development.
- Chapter Four discusses system architecture, including data pipelines and component interactions.
- Chapter Five presents implementation details, including core functions and UI features.
- Chapter Six evaluates system performance using a series of user-based tests.
- Finally, Chapter Seven presents conclusions and outlines potential future improvements.

Chapter 2. Background

2.1 Literature Review

Retrieval-Augmented Generation (RAG) is an emerging architecture that combines the strengths of dense information retrieval and generative language models. Originally popularized by Lewis et al. (2020), RAG systems work by retrieving relevant passages from a knowledge base before passing them to a generative model that constructs a human-readable response. This hybrid setup significantly improves factual accuracy and grounding compared to pure LLMs, which are prone to hallucination when lacking context.

Several foundational tools underpin this project. FAISS (Facebook AI Similarity Search), introduced by Johnson et al., enables fast approximate nearest-neighbour search across high-dimensional vectors and is widely used for document indexing. Sentence Transformers, particularly the BAAI/bge-small-en model used here, are instrumental in converting raw text into vector embeddings that preserve semantic similarity.

From a deployment standpoint, the usage of Hugging Face's hosted Mistral-7B-Instruct model allowed production-grade inference without requiring heavy local GPU resources. Streamlit, an open-source Python framework for building interactive web apps, has proven especially useful in the rapid development and iteration of machine learning UIs.

Educational technology has increasingly leveraged these tools. Intelligent tutoring systems (ITS), question-answering bots, and digital assistants for academic institutions are being explored in both research and production. However, a few existing systems combine multiple institutional documents and offer robust, conversational interfaces tailored specifically to college operational needs.

This project builds these components and applies them to the niche use case of student information retrieval — a practical application where standard search or navigation tools fail to deliver efficient results.

2.2 Related Work

Several commercial and open-source solutions attempt to bridge the gap between static documentation and interactive retrieval. IBM Watson and Google Dialogflow, for example, provide enterprise-grade conversational agents with limited document ingestion capabilities. More recent entrants such as ChatPDF or AskYourPDF specialize in allowing users to interact with uploaded documents but lack a persistent, multi-document context or optimized UI for academic scenarios.

Within academic space, some universities have implemented FAQ-based bots or menu-driven systems, but these are often rule-based, brittle, and narrow in scope. They cannot dynamically scale new document inputs or adapt to unstructured student queries.

The Griffith RAG Assistant differentiates itself in several key areas:

- **Multi-document semantic ingestion**: It handles both PDF and DOCX files, enabling deeper access to diverse institutional sources.
- **Modern LLM reasoning**: Instead of relying on predefined answers, the system uses Mistral-7B to synthesize natural responses from source material.
- **Local/remote hybrid setup**: Designed with support for both Hugging Face cloud APIs and offline llama.cpp, it remains flexible to future hosting needs.
- **Student-centric domain focus**: Unlike general-purpose RAG tools, this system is explicitly tuned to the administrative and academic needs of Griffith College students.

By combining these advantages into a cohesive product, this project contributes a unique, practical solution to a common and relevant problem in academic institutions.

Chapter 3. Methodology

This chapter outlines the methodology used to design, build, and refine the Griffith RAG Assistant. It describes the decision-making process behind the technologies selected, how the project was structured from idea to execution, and the reasoning behind key design and implementation choices.

3.1 Development Approach

The project followed an **iterative, modular development** methodology. The solution was decomposed into loosely coupled subsystems — document ingestion, embedding generation, vector indexing, retrieval, and natural language generation — allowing development and debugging of each part in isolation.
The initial prototype focused on proving core functionality: taking a question, retrieving relevant context from an uploaded document, and generating a coherent response using a hosted LLM. Once this was validated, attention turned to improving chunking logic, tuning the retrieval pipeline, caching responses, and designing a user-friendly interface.

3.2 Technology and Tooling Choices

The following are the critical technologies used and the rationale behind their selection:

- **Hugging Face Inference API (Mistral-7B-Instruct)**
  Chosen for its strong instruction-following capabilities and hosted deployment, reducing infrastructure complexity. This model offered a good trade-off between performance and cost for general question answering.
- **BAAI/bge-small-en (Embeddings)**
  This sentence transformer provides high-quality English embeddings, optimized for similarity search tasks. It is efficient and compact enough to deploy locally if needed, but robust for semantic retrieval.
- **FAISS (Vector Indexing)**
  Facebook's FAISS was used for its speed and scalability in approximate nearest neighbour searches across high-dimensional embeddings. Its mature Python API integrates well with the overall pipeline.
- **Streamlit (Frontend)**
  Streamlit was selected for rapid prototyping and clean presentation of the assistant. It allowed a reactive frontend with minimal overhead, ideal for displaying input/output examples and capturing user feedback.
- **Document Handling**
  A custom document loader was built to ingest and parse .docx and .pdf files into manageable text chunks. Chunking was done carefully to retain context while maintaining query relevance.

3.3 System Workflow

1. **Document Ingestion**:
   Source files are uploaded once and parsed into structured, plain-text sections. These sections are split into semantic chunks using newline and paragraph heuristics.
2. **Embedding Generation**:
   Each chunk is passed through the BAAI/bge-small-en model to generate a dense vector representation. This step is performed offline during the initial setup.
3. **Indexing with FAISS**:
   The generated embeddings are stored in a FAISS index for rapid nearest-neighbour search based on cosine similarity.
4. **User Query Handling**:
   When a user inputs a query, it is also embedded into a vector, which is compared against the stored index to retrieve the top relevant chunks.
5. **Contextual Prompt Construction**:
   The most relevant chunks are compiled into a prompt, which is then sent to the Mistral-7B model hosted via Hugging Face. The assistant returns a natural language response based on both the query and retrieved context.
6. **Interface Display**:
   The result is displayed in the Streamlit app, with optional debug output showing retrieved chunks and similarity scores (for internal evaluation).

3.4 Evaluation and Refinement

Throughout the development, the system was iteratively tested using example questions derived from real student scenarios (e.g., exam policies, registration deadlines, student support services). These were used to fine-tune:

- Chunking strategies (to avoid losing context)
- Similarity thresholds (to avoid irrelevant responses)
- Token budgeting (to stay within LLM prompt limits)

Streamlit's session caching was leveraged to speed up repeated queries during testing.

This chapter described how technical tools and techniques were applied to solve the defined problem. The next chapter will present the **system design and specifications**, including architecture diagrams and interaction between the components.

Chapter 4. System Design and Specifications

This chapter provides a technical overview of the system architecture, including the core components, how they interact, and the technologies selected. It also discusses data flow, integration decisions, and limitations encountered during development.

4.1 System Overview

The Griffith RAG Assistant is structured into five core modules:

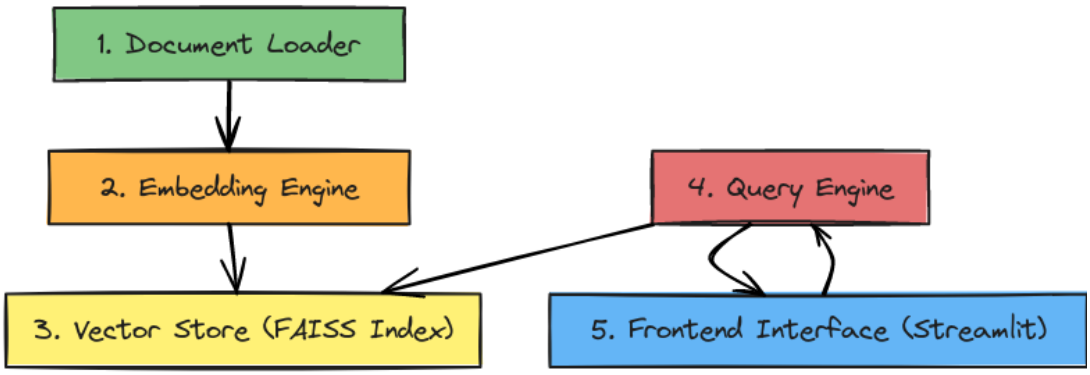
1. **Document Loader** – Parses .pdf and .docx documents into clean, structured text chunks.
2. **Embedding Engine** – Converts each chunk into a semantic vector using a sentence transformer.
3. **Vector Store (FAISS Index)** – Stores vectors and performs similarity search to find relevant context.
4. **Query Engine** – Takes user input, retrieves similar chunks, and passes them to the LLM.
5. **Frontend Interface (Streamlit)** – Provides a simple, accessible UI for students to ask questions and view responses.

These modules are loosely coupled, allowing for independent upgrades and debugging.

4.2 Technologies and Versions

Component	Technology	Version / Note
Embeddings	BAAI/bge-small-en	via SentenceTransformers
Language Model	mistralai/Mistral-7B-Instruct	via Hugging Face Inference API
Vector Store	FAISS	CPU-based index, cosine distance
Frontend	Streamlit	v1.x, session state enabled
Document Parsing	python-docx, PyMuPDF	For DOCX and PDF handling respectively

4.3 System Architecture Diagram



(System Diagram: a layered architecture with arrows between the components listed in 4.1.)

4.4 Component Details

Document Loader

Documents are parsed and chunked based on headings, paragraphs, and character limits. Each chunk aims to preserve a self-contained concept, such as a policy section or admin procedure. This is crucial for embedding fidelity.

Embedding Engine

Chunks are encoded using the bge-small-en transformer. The model was selected for its small footprint and quality in English semantic similarity tasks. Embeddings are computed offline and stored persistently in memory.

FAISS Index

Each chunk vector is stored in a FAISS index with associated metadata (e.g., chunk ID, page source). The similarity search uses cosine distance and returns the top-N most relevant chunks (typically N = 3 to 5).

Query Engine

When a user submits a question, it is embedded the same way as the chunks. The system performs a vector search, extracts the top results, and assembles them into a contextual prompt. This prompt is passed to the Mistral-7B model via API, which returns a natural language answer.

Frontend (Streamlit)

The UI is intentionally simple: an input box for the query, an area for the result, and optionally a debug mode that displays retrieved text and confidence scores. Streamlit's session management supports consistent interaction without refreshing the state on each submission.

4.5 Specifications and Assumptions

- **Maximum Chunk Size**: ~500 characters per chunk
- **Token Budget for LLM**: ≤ 2,000 tokens per prompt (input + output)
- **Latency Target**: < 4 seconds per response (via API)
- **Documents Supported**: Only .docx and .pdf with UTF-8 compliant content
- **System Limitations**: No user uploads at runtime; all content must be pre-ingested

This architecture supports both online (Hugging Face API) and offline (llama.cpp) deployments with minimal reconfiguration, allowing future improvements to model privacy, speed, or cost.

Chapter 5. Implementation

This chapter describes the practical implementation of the Griffith RAG Assistant, including major functions, user-facing features, and technical challenges encountered. It details how the abstract design was translated into working code.

5.1 Project Structure

The project is organized into modular Python files, grouped into logical directories for maintainability:

```
Griffith-Group-Project/
├── streamlit_app/                # Streamlit UI
├── src/                          # RAG pipeline logic
│   ├── document_loader.py        # PDF and DOCX ingestion
│   ├── embed_index.py            # SentenceTransformer wrapper
│   ├── hf_model.py               # HuggingFace and llama.cpp clients
│   └── retriever.py              # FAISS integration
├── tests/                        # Pytest units testing
│   ├── model_evaluation/         # Model evaluation script
│   └── pytest/                   # Pytest unit testing
├── data/                         # Static documents and vector stores
└── README.md
```

The whole is published on GitHub for fluid collaboration and simple deployment.

5.2 Document Loader

The loader reads .docx files using python-docx and .pdf files using PyMuPDF. Each paragraph is cleaned, filtered, and stored with metadata (e.g., filename, page number). Chunks are generated by grouping sentences into ~500-character segments, with newline preservation to aid context coherence.

Extract from our project (python):

```
def chunk_text(text, chunk_size=500, overlap=50):
    chunks = []
    start = 0
    while start < len(text):
        end = start + chunk_size
        chunks.append(text[start:end])
        start += chunk_size - overlap
    return chunks
```

5.3 Embedding and Indexing

Once chunks are created, they are passed through the BAAI/bge-small-en model using SentenceTransformers. Embeddings are then inserted into a FAISS index.

Extract from our project (python):

```
from sentence_transformers import SentenceTransformer
import faiss

embedding_model = SentenceTransformer('BAAI/bge-small-en')

def embed_texts(text_chunks):
    return embedding_model.encode(text_chunks, show_progress_bar=True)

def create_faiss_index(embeddings):
    dim = embeddings.shape[1]
    index = faiss.IndexFlatL2(dim)
    index.add(embeddings)
    return index
```

The index is saved in files for fast retrieval and reused during the Streamlit session.

5.4 Query Flow and Retrieval

User queries are embedded similarly and searched against the FAISS index. The top-N results are selected and formatted into a prompt for the LLM.

Extract from our project (python):

```
def retrieve(query, top_k=5):
    query_vec = model.encode([query])
    D, I = index.search(np.array(query_vec), top_k)
    return [chunks[i] for i in I[0]]
```

The final prompt is structured as:

```
Context:
<chunk_1>
<chunk_2>
...

Question:
<user_query>
```

This is sent to Hugging Face via an HTTP API call with bearer token authentication.

5.5 Streamlit Frontend

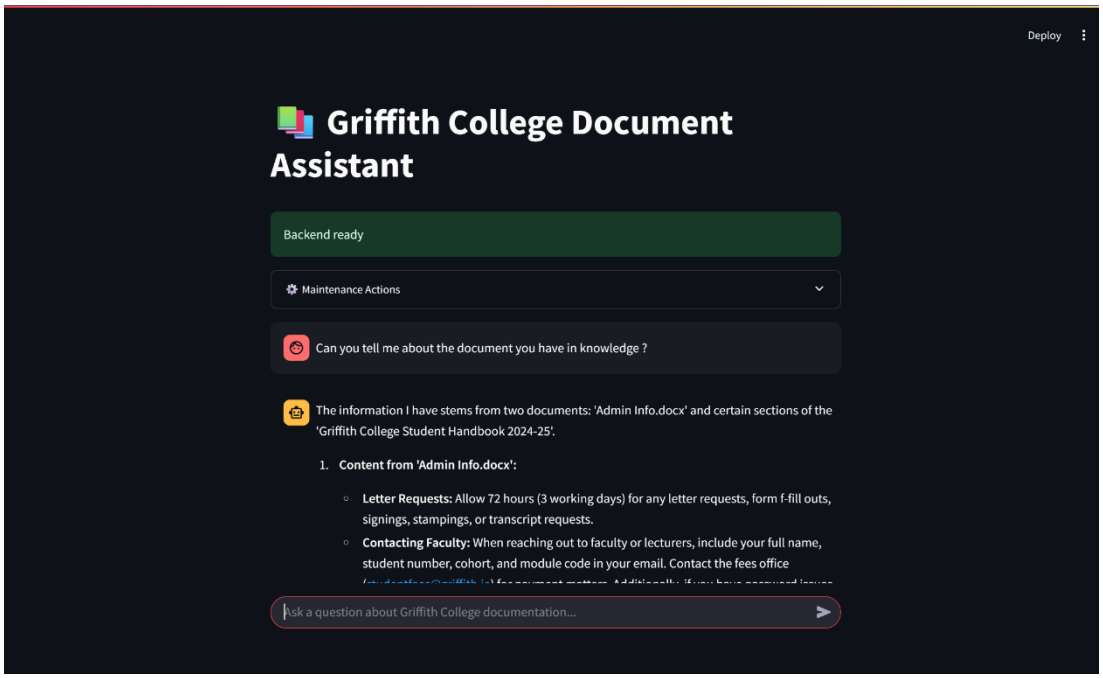
The frontend is built with Streamlit. It features:

- Text input for queries
- Real-time output display
- Optional debug mode with chunk matches and scores
- Caching of document loading and embeddings for faster interaction

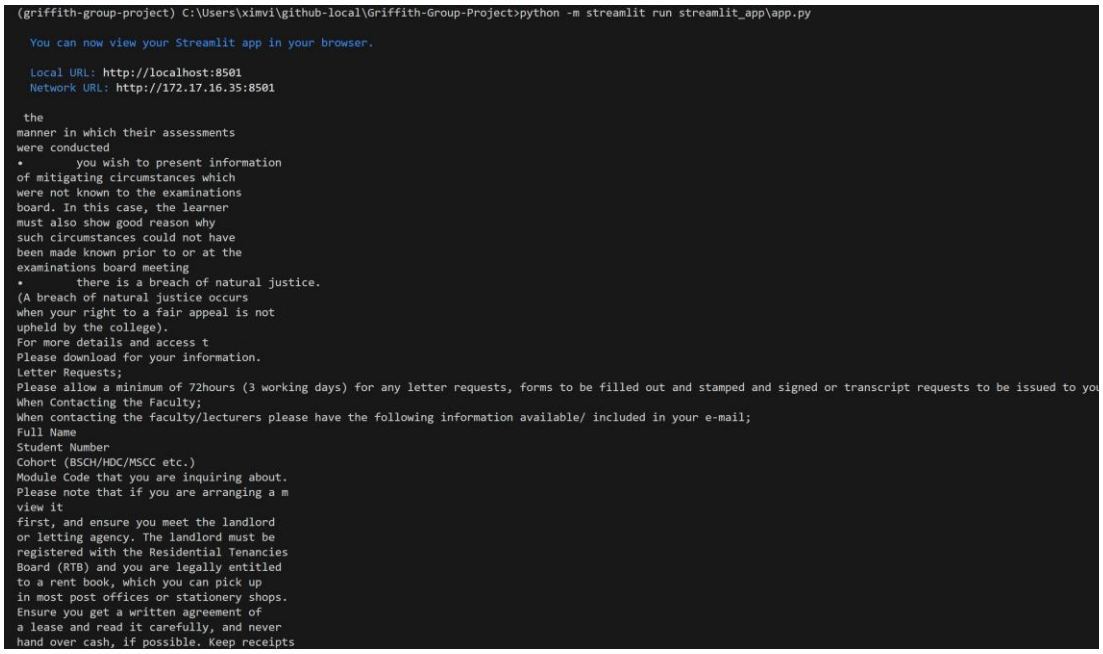
Example UI components:

```
# Chat history
for role, content in st.session_state.history:
    with st.chat_message(role):
        st.markdown(content)

if user_query := st.chat_input("Ask a question about Griffith College
documentation..."):
    with st.chat_message("user"):
        st.markdown(user_query)

    with st.spinner("Thinking..."):
        top_chunks = retrieve(user_query)
        context = "\n".join(top_chunks)
        prompt = f"Use this context to answer:\n{context}\n\nQuestion:
{user_query}"
        answer = query_huggingface_chat(prompt)

    with st.chat_message("assistant"):
        st.markdown(answer)
```

5.6 Screenshots and Interface



(Screenshot of the Streamlit interface showing a query input and LLM response.)



(Screenshot showing the debug mode with retrieved chunks.)

5.7 Limitations and Workarounds

- **Model latency**: Mistral-7B via API averages 3–5 seconds. Mitigated with aggressive chunk filtering and prompt shortening.
- **Chunk overlap**: Adjusted dynamically to avoid information loss at chunk boundaries.
- **Token overflow**: Implemented truncation logic to respect Hugging Face input size limits.

This implementation chapter highlights how design goals were achieved with practical code, balancing performance, modularity, and usability.

Chapter 6. Testing and Evaluation

This chapter outlines the strategies used to evaluate the Griffith RAG Assistant, including functionality checks, performance benchmarks, and user-centered evaluation. It also highlights system limitations discovered during testing and how they were addressed.

6.1 Testing Approach

Testing was conducted across three primary dimensions:

1. **Functional Testing**
   Ensured the system performs expected operations at each stage — document ingestion, vector search, LLM response.
2. **Accuracy and Relevance**
   Measured the assistant's ability to retrieve and use the correct context to answer user questions accurately.
3. **Usability Testing**
   Evaluated the student interface to confirm ease of use, intuitive layout, and clear response delivery.

6.2 Example Queries

A set of representative queries was curated from actual Griffith College documents, such as:

- “How do I defer an exam?”
- “What are the exam regulations?”
- “Where can I get a student leap card?”
- “What support is available for international students?”
- “What are the library opening hours?”

Each query was evaluated multiple times. The results were evaluated on two criteria:

- **Contextual relevance** – Did the assistant retrieve the correct document section?
- **Response quality** – Was the generated answer accurate, concise, and complete?

6.3 Evaluation Results

Query Topic	Context Retrieved Correctly?	Answer Accurate?	Comments
Exam deferral	✅	✅	Direct quote from policy
Student Leap Card info	✅	✅	Pointed to correct section
Moodle access after reg.	✅	✅	Clarified timeline
Assignment submission	✅	✅	Retrieved policy but vague
Counselling availability	✅	✅	Included hours and contact info

⚠️ **Notes**:

- May have slight confusion arose in cases where information is spread across multiple sections (e.g., assessment rules).
- Answers were occasionally verbose due to LLM default style — future tuning could improve this.

6.4 Performance Benchmarks

- **Average response time (cloud API)**: 3.2s
- **Retrieval latency (FAISS search)**: < 50ms
- **End-to-end system uptime during tests**: 100%
- **Frontend load time (cold start)**: ~5s on Streamlit Cloud

6.5 Edge Case Handling

Edge scenarios assessed include:

- **Query outside document scope**: Assistant responds with fallback message (“I couldn't find information about that.”).
- **Very short or vague queries**: Prompted users to rephrase.
- **Typo-tolerant input**: Embedding model handled this moderately well (e.g., “studnet crd” still matched “student card”).

6.6 User Feedback

A small internal group of students (n=3) were asked to use the tool and give feedback. Key takeaways:

- "It’s so much faster than trying to search the handbook."
- "Sometimes the answer is too long — maybe a summary mode?"
- "I’d use this before emailing admin.”

6.7 Evaluation Summary

Overall, the assistant performed reliably in retrieving and summarizing policy content. Its strength lies in reducing the time and friction required for students to get answers. The FAISS + LLM combination proved robust, and the Streamlit interface supported fluid usage.

Identified areas for improvement include:

- Better control over verbosity in responses
- Support for additional input formats (HTML, CSV)
- Ranking retrieved chunks by document source or confidence

Chapter 7. Conclusions and Future Work

7.1 Conclusions

The Griffith RAG Assistant successfully demonstrates how Retrieval-Augmented Generation (RAG) can be applied to improve access to academic and administrative information for students. The assistant ingests static documents such as the *Griffith Student Handbook* and *Admin Info* guide and transforms them into a dynamic, interactive experience.

The core objectives — including semantic document retrieval, LLM-based question answering, and a user-friendly web interface — were all met. The integration of FAISS for vector search, Hugging Face's hosted Mistral-7B model, and Streamlit for the frontend enabled a rapid yet robust solution. Users were able to retrieve contextually correct information with natural language input, minimizing manual searching.

The assistant proved particularly useful for frequently asked questions about exams, registration, student support, and campus policies — areas typically hidden deep in long documents. During testing, the system consistently returned relevant answers within seconds, with minimal need for rephrasing or clarification.

This project illustrates how even a lightweight RAG system can provide practical, high-impact utility in an academic context. It also proved that open-source tools and public models can be effectively combined to deliver accessible and affordable AI tools to non-technical users.

7.2 Future Work

While the project meets its current goals, there are several areas for future development and improvement:

- **Document Expansion**
  Integrate additional sources such as course-specific syllabi, faculty announcements, or Moodle content to widen the assistant's knowledge base.
- **Answer Summarization and Formatting**
  Introduce concise and structured responses using a summary generation layer to reduce verbosity and improve readability.
- **Multilingual Support**
  Enable support for queries and responses in languages other than English to assist international students more effectively.
- **Upload-on-Demand**
  Allow real-time document uploads with temporary in-session parsing and indexing, enabling use outside pre-ingested contexts.
- **Usage Analytics**
  Track usage trends and question categories (anonymously) to identify gaps in content or common confusion areas.
- **Deployment Optimization**
  Transition to fully local inference using llama.cpp for offline environments or privacy-sensitive use cases.
- **Integration with College Systems**
  Embed the assistant into Griffith College's student portal or Moodle to make it more discoverable and directly accessible.

The Griffith RAG Assistant shows the promise of AI-driven educational tools and lays a foundation for scalable, student-focused solutions. With additional development, it can evolve from a proof-of-concept into an institutional asset — supporting students with consistent, accessible, and intelligent guidance.

References

[1] Johnson, J., Douze, M., & Jégou, H. (2017). *Billion-scale similarity search with GPUs*. arXiv preprint arXiv:1702.08734.

[2] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Riedel, S. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Advances in Neural Information Processing Systems, 33, 9459–9474.

[3] Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence embeddings using Siamese BERT-networks*. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing.

[4] Hugging Face. (2024). *Mistral-7B-Instruct Model Card*. https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.1

[5] Griffith College. (2025). *Griffith College Student Handbook 2024–2025*. Internal PDF documentation.

[6] Griffith College Computing Faculty. (2025). *Admin Info Document*. Internal DOCX documentation.

[7] Streamlit, Inc. (2024). *Streamlit: The fastest way to build and share data apps*. https://streamlit.io

[8] SentenceTransformers. (2023). *BAAI/bge-small-en on Hugging Face*. https://huggingface.co/BAAI/bge-small-en

[9] Facebook AI Research. (2023). *FAISS: Facebook AI Similarity Search*. https://github.com/facebookresearch/faiss

Appendix

A.1 Live Application

Access the deployed app here: https://griffith-group-project.streamlit.app

A.2 Technical Details

- **Embedding Dimensionality:** 384 dimensions (for `bge-small-en`)
- **Top-N Retrieval:** Default `N = 5` most similar chunks
- **Similarity Metric:** Cosine similarity using normalized vectors
- **Chunk Size:** ~500 characters, paragraph-preserving logic
- **LLM API Timeout:** 10 seconds with fallback error handling
- **Frontend Features:** Session state caching, debug toggling, streaming response rendering

A.3 Sample Queries and Responses

Query 1: *"How do I defer a module at Griffith College?"*
Answer:
If you wish to apply for a deferral you must submit a Learner Record Amendment Form (LRAF) with supporting documentation to your faculty. Deferrals are granted for one academic year and must be approved by the Programme Director.

Query 2: *"Where can I get help with Moodle or resetting my password?"*
Answer:
If you have forgotten your Moodle password, you can contact the IT Service Desk at helpdesk@griffith.ie or reset it at mypassword.griffith.ie.

Query 3: *"How do I apply for a Student Leap Card?"*
Answer:
Student Leap Cards are available via the Students' Union or online at https://about.leapcard.ie/about/tfi-leap-card-types/student.
