RMIT UNIVERSITY

Data Communication and Net-Centric Computing (COSC1111)

AI-assisted Supporter

School of Computing Technologies

1. Introduction and Project Overview

ARTIM is an AI-powered assistant designed to help students navigate Canvas LMS (Learning Management System) effectively. Developed as part of RMIT's DCNC 2025 Community startup project "AI-assisted supporter," this project demonstrates the practical implementation of advanced Retrieval-Augmented Generation (RAG) techniques in an educational context.

The project was inspired by RMIT's GenAI tool "Val" and its Course Assist persona, aiming to create an intelligent assistant that could provide accurate, contextual support for students using Canvas LMS. The development process followed an evaluation-driven approach, emphasizing empirical testing and iterative improvements to ensure reliable performance.

1.1 Project Objectives

The primary aims of the project were:

- Create an intelligent assistant to help students navigate Canvas LMS effectively
- Implement and evaluate various RAG techniques in a real-world application

1.2 Technical Stack

- **Backend**: Python-based implementation using LangChain/LangGraph and Claude on Amazon Bedrock
- **Frontend**: Streamlit interface for user interaction
- **Vector Store**: ChromaDB for knowledge base storage
- **Embedding Model**: BGE-M3 (1024-dimensional embeddings via Ollama)
- **LLM Models**: Claude-3.5-Sonnet
- **Evaluation Framework**: DeepEval for synthetic data generation and testing

The project structure is organized into distinct phases:

1. Data indexing and knowledge base setup
2. Synthetic data generation for testing
3. Retrieval evaluation and optimization
4. MVP development with boto3/AWS Bedrock
5. Generation evaluation and improvement
6. Final production-ready implementation with LangGraph and Streamlit

2. Functional Overview of the Final Product (ARTIM)

2.1 System Architecture

ARTIM is implemented as an agentic RAG chatbot leveraging LangGraph for the backend orchestration and Streamlit for the user interface. The system employs a sophisticated multi-tool architecture that enables autonomous decision-making and dynamic information retrieval.

2.2 Core Tools and Components

1. Query Rewriting Tool (rewrite_query)

The query rewriting tool optimizes user queries for better retrieval performance. It breaks down complex, multi-part questions into focused search queries and refines ambiguous inputs. This preprocessing step significantly improves the precision of document retrieval by creating cleaner, more targeted search vectors.

2. Document Retrieval Tool (fetch_canvas_guides)

This tool retrieves relevant information from the ChromaDB vector store based on optimized queries. It uses BGE-M3 embeddings (1024-dimensional) and performs similarity search to identify the most relevant document chunks from the indexed Canvas documentation knowledge base.

3. Contextual Compression Tool (compress_information)

Implementing LLM-based contextual compression, this tool extracts the most relevant information from retrieved documents. It filters out extraneous content and focuses on information directly pertinent to answering the user's query, reducing token consumption and improving response relevance.

2.3 Key Features

- **Multi-turn Conversation Support**: The system maintains conversation context across multiple exchanges, enabling natural, coherent dialogues with users.
- **Multiple Individual Chat Sessions**: Users can maintain separate conversation threads, allowing them to organize different topics or questions independently.
- **Last Message Editing Capability**: Users can edit their most recent message, and the system will regenerate responses based on the modified input without losing prior conversation context.
- **Long-term Memory Retention**: The system persists conversation history, enabling users to return to previous sessions and continue where they left off.
- **Dynamic and User-friendly Interface**: The Streamlit frontend provides an intuitive, responsive interface with real-time updates and clear visual feedback.
- **Real-time Response Streaming**: Responses are streamed to the user as they are generated, providing immediate feedback and reducing perceived latency.
- **Indexed Knowledge Base with Custom Pre-processing**: The knowledge base is built from curated Canvas documentation, including student guides, troubleshooting guides, and basic tutorials. Custom pre-processing ensures high-quality, semantically rich document chunks.

- **Tool-call Trace for Enhanced User Experience**: During response generation, users can see which tools the system is invoking, providing transparency into the retrieval and reasoning process.
- **Multi-Lingual Support**: Ability to cater for a wide range of audience.

2.4 Workflow

The agentic workflow allows the LLM to autonomously make multiple tool calls until it produces a satisfactory output.

The typical flow involves:

1. User submits a query through the Streamlit interface
2. The system optionally rewrites the query for optimal retrieval
3. The retrieval tool fetches candidate documents from the vector store
4. The compression tool filters retrieved content for relevance
5. The LLM generates a response based on the compressed, relevant information
6. The response is streamed to the user in real-time
7. The conversation state is updated and persisted

This agentic approach enables the system to adapt dynamically to different query types and complexity levels, making autonomous decisions about which tools to use and when.

3. Results and Metrics

3.1 Retrieval Evaluation

The retrieval pipeline was evaluated using the Contextual Relevancy metric, which employs an LLM-as-a-judge approach to measure how relevant the retrieved information is for a given input query.

Baseline Performance

The naive retriever (basic vector search over the Chroma knowledge base) achieved a 5.26% pass rate on the test set, indicating significant room for improvement.

Improved Performance

After implementing query-rewriting and contextual compression techniques, the retrieval system achieved a 36.84% pass rate. This represents a 7x relative improvement over the baseline, demonstrating the substantial impact of these optimization techniques.

Key Improvements:

- Query-rewriting: Breaking down multi-part user messages into focused search queries improved retrieval precision
- Contextual compression: Filtering retrieved contexts to minimal relevant information reduced noise and improved relevance

Trade-offs:

Both query-rewriting and contextual compression add extra processing steps, increasing end-to-end retrieval latency. The trade-off between accuracy/cost and latency is an important consideration for production deployment.

3.2 Generation Evaluation

The generation quality was evaluated using the Answer Relevancy metric from DeepEval, which uses an LLM-as-a-judge to assess how relevant the generated output is compared to the user's input query.

Baseline Performance

The naive generation approach (simple retrieval + generation without optimization) achieved a 23% pass rate across 100 test cases (23/100).

Improved Performance

After implementing advanced RAG techniques including query rewriting, document filtering with contextual compression, and tool-augmented responses in an agentic workflow, the system achieved a 37% pass rate across 100 test cases (37/100).

This represents a 61% relative improvement over the baseline (14 additional test cases passing), demonstrating the effectiveness of the agentic approach and optimization techniques.

3.3 Evaluation Methodology

- **Synthetic Dataset Generation**: Test cases were generated using DeepEval's synthetic data generation capabilities, creating realistic user queries based on the Canvas documentation knowledge base.
- **Automated Evaluation**: Both retrieval and generation were evaluated using automated LLM-based metrics, enabling systematic and scalable testing across large test sets.
- **Quality Validation**: Generated synthetic questions underwent quality checks to ensure they were representative of actual user queries and appropriate difficulty levels.

3.4 Performance Insights

While the improvements are significant, the absolute pass rates indicate there is still substantial room for optimization.

- The RAG system involves numerous hyperparameters that significantly impact performance (embedding model configuration, retrieval parameters, context window sizes, LLM temperature, etc.)
- More comprehensive fine-tuning and rigorous testing would be beneficial before production deployment
- Extended parameter optimization across different scenarios could yield further improvements
- Larger-scale relevancy testing and performance benchmarking under various load conditions are needed

4. Key Learnings and Challenges

4.1 Deep Dive into Open-Source Libraries

The project provided an opportunity to deeply explore various open-source libraries including LangChain, LangGraph, and DeepEval. A significant learning occurred mid-project when LangChain released v1.0, bringing substantial changes. This necessitated reading the source code of LangChain and its integrations, which proved pivotal in the learning process. While initially daunting, this approach resulted in a more comprehensive understanding of the internal workings than documentation alone would have provided.

4.2 Low-Level APIs vs High-Level Abstractions

During development, the project explored interactions with Anthropic models using both boto3/Bedrock (low-level) implementations and high-level abstractions provided by LangChain.

Key Insights:

- LangChain makes it easier to get started with GenAI and Agentic AI applications
- Low-level implementations provide greater control and flexibility
- At the time of initial MVP development, LangChain's AWS Bedrock integration had not reached feature parity, so the MVP was implemented using boto3/Bedrock
- LangChain/LangGraph was favored in the final product due to the rich features and complete package it offers

The experience demonstrated the value of understanding both abstraction levels, enabling informed decisions about when to use each approach.

4.3 Working with Anthropic's Claude

Having worked predominantly with OpenAI chat models through LangChain, this project provided an opportunity to work extensively with Anthropic's Claude, specifically claude-3-haiku and claude-3.5-sonnet, which bring their own interesting features and capabilities. The systematic exploration in the playground section, progressing from basic API interactions to advanced features like streaming, tool usage, and RAG implementation, provided a thorough understanding of Claude's capabilities.

4.4 Local Models and Embeddings

The project involved working with Ollama and local Hugging Face models, providing experience with:

- Local model deployment and management
- Embedding model selection and configuration
- Trade-offs between local and cloud-based solutions

4.5 Evaluation-Driven Development

The use of DeepEval for systematic evaluation proved to be a cornerstone of the project. The ability to generate synthetic datasets based on the knowledge base and test implementations iteratively enabled data-driven decision-making about which techniques to implement and optimize.

4.6 Embedding Quality and Visualization

A notable insight came from embedding inspection using Nomic Atlas to visualize clusters and spot-check nearest neighbors. This lightweight check revealed how documents were positioned in vector space and whether relevant documents were placed close together, indicating embedding quality. The visualization demonstrated the importance of the indexing pipeline's design, which intentionally includes metadata enrichment and provenance hints to encourage semantic clustering of related content.

4.7 Technical Challenges

Dependency Management: Due to the LangChain v1.0 release mid-project and specific version requirements for local models (particularly torch requiring older numpy versions), dependency conflicts emerged. The solution was to create an incremental installation notebook (installing_requirements.ipynb) that installs dependencies in a tested order, avoiding conflicts.

Feature Parity Issues: The MVP was implemented using boto3/Bedrock because LangChain's AWS Bedrock integration had not reached feature parity at that time. This required maintaining two parallel implementations temporarily.

Hyperparameter Complexity: The RAG system involves numerous hyperparameters affecting performance (embedding model configuration, retrieval parameters, context window sizes, LLM temperature settings, query rewriting parameters, compression settings). Navigating this complexity and finding optimal configurations required systematic experimentation and evaluation.

4.8 Design Decisions

Query-Rewriting: While experiments showed that the LLM often internally filters and reformulates noisy user messages, custom query-rewriting was implemented for consistent reformulation of queries. Manual inspection of chat logs validated this decision.

Contextual Compression: The trade-off between improved relevancy/reduced token consumption and increased latency was carefully considered. The decision to implement compression was validated by the significant improvement in evaluation metrics.

Storage Implementation: The decision to use ChromaDB for the vector store, along with specific chunking strategies and metadata enrichment approaches in the indexing pipeline, was driven by both practical considerations and the goal of improving semantic clustering in embedding space.

System Prompt: Final system prompt was curated after numerous iterations of system prompts, each varying with the different levels of role adherence and characteristics. The final system prompt is clear in its role while still be transparent. This was done so that user can probe into the chatbot’s reasoning and view tool results. This can easily be modified as needed. Tuning the system posed a hard challenge, even minor changes resulted in a drastically different behaviour. Sometimes the model forgets some instructions but follows upon pointing out it’s mistake.

5. Conclusion and Next Steps

5.1 Project Achievements

ARTIM successfully demonstrates the practical implementation of advanced RAG techniques in an educational context. The project achieved its primary objectives:

- Created an intelligent, functional assistant for Canvas LMS navigation
- Implemented and systematically evaluated various RAG techniques
- Developed a near production-ready system with a user-friendly interface
- Established an evaluation-driven development process that yielded measurable improvements

5.2 Current Limitations

Evaluation Scope: Due to time and resource constraints only minimal evaluation was conducted. More comprehensive fine-tuning and larger-scale testing would strengthen confidence in production deployment.

Hyperparameter Optimization: The RAG system involves numerous hyperparameters that significantly impact performance. Systematic optimization across different scenarios remains an area for improvement.

Performance Benchmarking: Extended testing under various load conditions and systematic evaluation of tool-calling strategies would provide better insights into production readiness.

LangSmith Utilization: The project currently uses LangSmith only for basic debugging. More extensive utilization for monitoring, tracing, and performance analysis could yield valuable insights.

5.3 Next Steps and Future Development

5.3.1 Immediate Improvements

Enhanced Evaluation:

- Conduct larger-scale relevancy testing with more diverse test cases
- Perform systematic hyperparameter optimization
- Implement comprehensive performance benchmarking under various load conditions
- Expand use of LangSmith for detailed debugging and performance analysis

System Optimization:

- Further fine-tuning of retrieval parameters
- Optimization of query rewriting and compression techniques
- Latency optimization to reduce the overhead of multi-step processing
- Investigation of alternative embedding models or hybrid retrieval approaches

5.3.2 Feature Enhancements

User Feedback Collection:

Currently not implemented due to technical challenges, but would enable:

- Response quality tracking and monitoring
- System improvement based on user satisfaction data
- Hyperparameter optimization informed by real-world usage patterns

Text Input with Screenshot Support:

While current AI models lack reliable capability to understand screenshots and UI components, this remains a valuable future enhancement as vision-language models improve. This would enable users to share Canvas interface issues directly for troubleshooting.

Model Context Protocol Integration:

A future capability to access student's Canvas through API keys would enable:

- Direct interaction with the Canvas system
- Personalized responses based on the student's actual Canvas environment
- Actionable assistance (e.g., helping with assignment submissions, course navigation)

5.3.3 Production Readiness

Before full production deployment, the following steps are recommended:

- Extended Testing: Conduct comprehensive testing with real users to identify edge cases and usability issues
- Performance Monitoring: Implement robust monitoring and logging infrastructure to track system performance in production
- Scalability Assessment: Evaluate system behavior under high load and implement appropriate scaling strategies
- Security Review: Conduct thorough security audit, particularly for API integrations and user data handling
- Documentation: Create comprehensive user documentation and administrator guides
- Fallback Mechanisms: Implement graceful degradation strategies for cases where retrieval or generation fails