RAG Application Using Knowledge Graph and Vector Search

Prototype Project Report

HackerEarth

MECA Dynamics

Ethan Villalovoz, Molly Iverson, Adam Shtrikman, Chandler Juego

TABLE OF CONTENTS

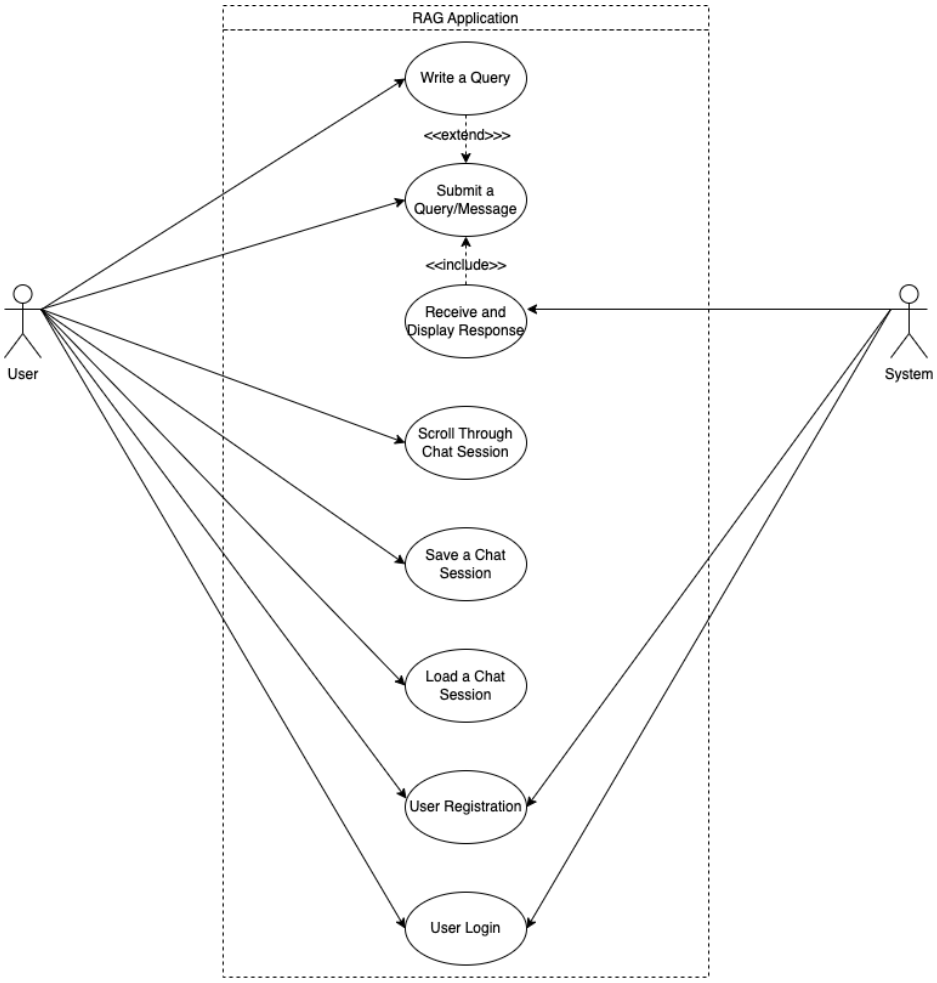
I. Introduction

This project focuses on developing a Retrieval-Augmented Generation (RAG) application that integrates knowledge graphs and vector search to enhance the accuracy and context of responses. RAG systems combine the strengths of large language models (LLMs) with information retrieval to generate contextually relevant responses. However, current systems relying solely on vector search struggle with unstructured data, which limits their effectiveness.

We are incorporating knowledge graphs to address this and provide structured information alongside vector embeddings. The knowledge graphs enhance the system's understanding of relationships between concepts, thus improving the retrieval and generation of more precise, fact-based responses. The project aims to develop a web-based RAG application that demonstrates the effectiveness of combining knowledge graphs and vector search in handling complex, multi-step queries using a dataset of 10,000 Wikipedia articles.

II. System Requirements Specification

II.1. Use Cases



Write a Query

Pre-Condition	- The user has opened the web application. - The query input field is visible and active.
Post-Condition	- The user has successfully entered a query in the input field.
Basic Path	- The user clicks on the query input field. - The user types a query or message in the input field. - The system accepts and displays the query in the input field.
Alternative Path	- If the query input field is inactive or unavailable, the system prompts the user to refresh the page or displays a message saying that the input field is unavailable. - If the user input exceeds a character limit, then the system displays an error message next to the input field asking the user to revise the query.
Related Requirements	- Chat Window

Submit a Query/Message

Pre-Condition	- The user has typed a query into the input field. - The system is connected to the backend which is connected to the knowledge graph and vector search database.
Post-Condition	- The system successfully receives the query, initiating retrieval from the RAG model. - The query appears in the chat interface as a sent message.
Basic Path	- The user clicks on the “Submit” button or presses “Enter” to submit the query. - The system records the query and forwards it to the RAG model for processing. - The query appears in the chat window as a sent message from the user.
Alternative Path	- If the user submits an empty query, then the system displays a message indicating that input is required. - If the system encounters an issue with the query submission (e.g., network error), then an error message is displayed in the chat window. - If the user submits a query that violates content moderation rules, then a warning message is displayed in the chat window and informs the user that the query is invalid.
Related Requirements	- Chat Window

Receive and Display Response

Pre-Condition	- The user has submitted a valid query, and the system has processed it using the RAG model.
Post-Condition	- The system returns a response based on the query and displays it in the chat window.
Basic Path	- The system processes the query using the RAG model. - The system retrieves relevant information. - The system generates a response using the RAG model and formats the information in a readable message. - The system displays the response in the chat window above the user input field.
Alternative Path	- If the user submits an empty query, then the system displays a message indicating that input is required. - If the system encounters an issue with the query submission (e.g., network error), then an error message is displayed in the chat window.
Related Requirements	- Chat Window - Query Response Retrieval

Scroll Through the Current Chat Session History

Pre-Condition	- The user has had an ongoing chat session with multiple messages exchanged between the user and the system.
Post-Condition	- The system successfully receives the query, initiating retrieval from the RAG model.
Basic Path	- The user scrolls up in the chat window. - The system dynamically loads previous queries and responses from the current session as the user scrolls. - The user can review and scroll back to any message in the session.
Alternative Path	- If the system encounters an issue loading older messages, a loading spinner is shown until the history is fully loaded. - If the chat session is too long, the system may implement infinite scrolling to manage performance.
Related Requirements	- Chat Window - Saving and Loading

Save a Chat Session

Pre-Condition	- The user has completed a chat session and wants to save the conversation for future reference.
Post-Condition	- The system saves the chat session and stores it under the user's profile for later access.
Basic Path	- The user clicks the "Save Session" button at the end of the chat. - The system prompts the user to name or label the chat session. - The system saves the chat session with the provided name and stores it in the user's session history. - A confirmation message is displayed to the user indicating the session was successfully saved.
Alternative Path	- If the system encounters an error during the saving process, the user is notified and asked to try saving again later. - If the user does not provide a session name, the system automatically saves the session with a default timestamp-based name.
Related Requirements	- Saving and Loading

Load a Chat Session

Pre-Condition	- The user has previously saved chat sessions and wants to load a past conversation.
Post-Condition	- The user is able to view and interact with a previously saved chat session.
Basic Path	- The user navigates to the "Saved Sessions" section in the application. - The system displays a list of saved chat sessions, each labeled with the session name and timestamp. - The user selects a session to load. - The system loads the selected chat session in the chat window. - The user can scroll through and review the saved conversation.
Alternative Path	- If the selected session fails to load, the system shows a pop-up error message and offers the user a chance to retry. - If the user has no saved sessions, the system displays a message "No saved sessions" in the saved sessions section of the UI.
Related Requirements	- Saving and Loading

User Registration

Pre-Condition	- The user is on the homepage of the RAG web application and is not logged in. The registration form is available to new users.
Post-Condition	- The user successfully creates an account, and their details are stored in the system. The user is now logged in and is able to access logged-in features such as saving and loading chat histories.
Basic Path	- The user clicks on the 'Sign Up' or 'Register' button. - The system displays a registration form requesting information such as username, email, and password. - The user fills in the required details and submits the form. - The system validates the provided information (e.g., checking if the email is already in use). - The system creates a new user account, saves the user details in the database, and logs the user into the application. - A confirmation message or welcome page is displayed, and the user is redirected to the application's main page.
Alternative Path	- If the user submits incomplete or invalid information (e.g., invalid email format, weak password, etc.), the system highlights the errors and prompts the user to correct them. - If the username or email is already registered, the system displays an error and asks the user to choose a different email or username. - If the system encounters a database or server error during registration, a generic error message is shown, and the user is advised to try again later.
Related Requirements	- User Registration and Log In

User Login

Pre-Condition	- The user is on the homepage or login page of the RAG web application and already has an account registered.
Post-Condition	- The user successfully logs into the application and is redirected to the main page where they can begin to use the application.
Basic Path	- The user clicks on the "Login" button or navigates to the login page. - The system displays a login form requesting the user's email/username and password. - The user enters their email/username and password and submits the form.

	- The system validates the credentials by checking the information against the stored user database. - If the credentials are valid, the system logs the user in and redirects them to the main page. - A success message is shown briefly, confirming the login.
Alternative Path	- If the user leaves any field blank, the system prompts the user to fill in the missing fields. - If the entered email/username or password is incorrect, the system displays an error message (e.g., "Invalid username or password") and prompts the user to try again. - If the user has forgotten their password, the system provides a "Forgot Password" link that directs them to a password recovery process. - If there is a server/database error during login, a generic error message is shown, and the user is asked to try again later.
Related Requirements	- User Registration

II.2. Functional Requirements

II.2.1. RAG Model Components

RAG Model Architecture

Description	The system will implement RAG architecture, including a receiver and generator. The receiver will search the knowledge base for the most relevant data based on user input, and the generator (LLM) will take the search results and generate an accurate response.
Source	Project scope document provided by the client.
Priority	Priority Level 0: Essential functionality.

Vector Search Implementation

Description	The system will implement vector search to retrieve relevant information based on user queries. This includes generating embeddings, indexing them using vector search libraries (e.g., FAISS or Annoy), and finding similar results based on user input.
Source	Project scope document provided by the client.
Priority	Priority Level 0: Essential functionality.

Knowledge Graph Integration

Description	The system will integrate knowledge graphs (e.g., DBpedia or YAGO) into the RAG pipeline to better retrieve relevant and contextual information for the query.
Source	Project scope document provided by the client.
Priority	Priority Level 0: Essential functionality.

II.2.2. Core App Functionality

Query Response Retrieval

Description	The system will process user queries, fetch relevant information from the knowledge graph and vector search index, and generate responses using an LLM. If the system does not know the answer to a query, it should tell the user that it doesn't have enough information to respond accurately; it should not lie.
Source	Project scope document provided by the client.
Priority	Priority Level 0: Essential functionality.

User Registration

Description	The system will allow users to create an account by providing a username, password, and email address. This involves a user-friendly interface for registration, input validation, and secure storage of user data.
Source	Internal requirements elicitation among members of the team.
Priority	Priority Level 1: Desirable functionality.

Saving and Loading

Description	The system will automatically save the last 20 chat sessions and display them as clickable titles on the side of the chat window. Users should be able to load previous sessions for reference.
Source	Internal requirements elicitation among members of the team.
Priority	Priority Level 1: Desirable functionality.

II.2.3. User Interface

Chat Window

Description	The system will display a chat window where users can type messages or queries. Once they press enter or click submit, the system will generate and display a response.
Source	Internal requirements elicitation among members of the team.
Priority	Priority Level 0: Essential functionality.

Error Handling

Description	The system will handle errors like invalid queries, network issues, or app failures by displaying appropriate error messages in the chat window.
Source	Internal requirements elicitation among members of the team.
Priority	Priority Level 1: Desirable functionality.

Content Moderation

Description	The system will prevent responses to harmful or inappropriate queries by showing a warning message. It will moderate content that includes hate speech, threats, violence, or graphic material. However, it will recognize when the query is research-based (e.g., questions about sensitive historical events like the Holocaust) and respond appropriately.
Source	Internal requirements elicitation among members of the team.
Priority	Priority Level 1: Desirable functionality.

II.3. Non-Functional Requirements

Scalability:

The system shall be scalable to handle large datasets, ensuring it can efficiently manage and query over 10,000 Wikipedia articles without degradation in performance.

Response Time:

The system shall respond to user queries within 2 seconds for typical requests, ensuring a smooth user experience.

Data Storage:

The system shall maintain sufficient storage for the knowledge graph and vector embeddings, ensuring persistent and reliable data retrieval.

Platform Compatibility:

The RAG application shall be accessible through a web browser, making it platform-independent and usable on various operating systems.

Maintainability:

The system shall be designed with maintainability and modular code that allows for future updates and extensions without significant refactoring.

Reliability:

The system shall maintain a 99% uptime, ensuring high availability for users, particularly during the testing and demonstration phases.

Security:

The system shall ensure the security of data and queries, particularly in handling sensitive information, by implementing secure protocols for data transmission and storage.

User Interface Usability:

The web interface shall be intuitive and easy to use, enabling users to input queries and view results without requiring extensive technical knowledge.

III. System Evolution

A critical consideration in the evolution of the system is software refinement. The system will experience multiple stages of development, testing, and feedback. During the feedback phase, the team will gain insight into technical shortfalls or missing functionality. Risk points in our design include dependencies on third-party libraries and services, which may introduce compatibility problems or become deprecated. Integration with large data sets might introduce performance bottlenecks, requiring alternative strategies. Output quality may be incorrect if the underlying data set contains inaccuracies or becomes outdated. The system can operate on a modular data set, and the team will pay close attention to the relationship between the data set and output quality during each iteration and feedback phase. The team anticipates potential

challenges and changes to occur, and design decisions will be made to avoid technical restrictions in coming developments. The focus is on modularity and detailed documentation, which will improve the ability to modify the system as needed, as well as ensure it remains adaptable to technological advances, evolving user requirements, and stakeholder feedback.

IV. Glossary

Knowledge Graph: A knowledge base that uses a graph-structured data model or topology to represent and operate on data.

Large Language Model (LLM): A machine learning model that can perform natural language processing tasks. They are trained on vast amounts of data in order to detect patterns effectively.

Natural Language Processing (NLP): A branch of artificial intelligence that uses machine learning to help computers interpret and generate human language.

Retrieval-Augmented Generation (RAG): A framework that combines the capabilities of large language models with information retrieval systems to improve the accuracy and relevance of AI-generated text.

Vector Search: A method for finding similar items to data points in large collections by using vector representations of items. These representations, or vector embeddings, can be used to quickly locate items in large data sets and allow for searches by meaning, rather than just keywords.

V. References

[1] P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in Curran Associates Inc., Vancouver, BC, Canada, 2020, pp. 9459–9474. doi: https://dl.acm.org/doi/abs/10.5555/3495724.3496517.

[2] K. Guu et al., “Traversing Knowledge Graphs in Vector Space”, *Stanford NLP Group – Standford University*, 2015. Available: https://nlp.stanford.edu/pubs/kg_traversal.pdf

[3] A. Kollegger, “Knowledge Graphs for RAG,” DeepLearning.AI. [Online]. Available: https://www.deeplearning.ai/short-courses/knowledge-graphs-rag/

[4] L. Voss, “JavaScript RAG Web Apps with LlamaIndex,” DeepLearning.AI. [Online]. Available: https://www.deeplearning.ai/short-courses/javascript-rag-web-apps-with-llamaindex/

[5] J. Liu and A. Datta, “Building and Evaluating Advanced RAG Applications,” DeepLearning.AI. [Online]. Available: https://www.deeplearning.ai/short-courses/building-evaluating-advanced-rag/

[6] “Generative AI with Large Language Models,” Coursera. [Online]. Available: https://www.coursera.org/learn/generative-ai-with-llms