VIETNAM NATIONAL UNIVERSITY HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Dinh Thai Duong

ENHANCING UMENTOR BY USING KNOWLEDGE GRAPH AND RAG

HIGH QUALITY GRADUATION THESIS
Major: Information Systems

Hanoi - 2025

VIETNAM NATIONAL UNIVERSITY HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Dinh Thai Duong

ENHANCING UMENTOR BY USING KNOWLEDGE GRAPH AND RAG

Major: Information Systems

Supervisor: Nguyen Ngoc Hoa

Hanoi - 2025

Acknowledgements

I would like to express my sincere gratitude to those who have supported and helped me throughout the duration of this thesis.

First of all, I would like to express my deep gratitude to Associate Professor, Dr. Nguyen Ngoc Hoa, who spent his valuable time and knowledge to guide, advise and provide continuous encouragement during the research and writing process. His insight and dedication helped me develop and perfect my research capacity, thereby completing this thesis.

I would also like to express my gratitude to the University of Technology - Vietnam National University, Hanoi, which has provided important opportunities, documents and information to help us complete our thesis. Along with that, the professional and spiritual support from the lecturers at the University of Technology - Vietnam National University, Hanoi has played an important role in helping me have the right direction to go to today's thesis.

Besides, I would also like to thank all of my classmates in class K66-T-CLC who have accompanied me throughout my studies and training at university.

Hanoi, May 2025

Dinh Thai Duong

COMMITMENT

I commit that my graduation thesis ”ENHANCING UMENTOR BY USING KNOWLEDGE GRAPH AND RAG” is my own research work under the guidance of Associate Professor, Dr. Nguyen Ngoc Hoa and has never been submitted as a thesis report at the University of Technology - Vietnam National University, Hanoi or any other university. During the process of writing this thesis, I did refer to answers from artificial intelligence platforms to make an outline. Most of the support from chatbots was in organizing ideas and editing expressions in sections 1 and 2. What I write is not copied from any human documents, nor used by others without specific citations.

I hereby declare that the content I present in my thesis was developed by myself and did not copy the source code of others. If there is any violation, I will take full responsibility according to the regulations of the University of Technology - Vietnam National University, Hanoi.

Author

Dinh Thai Duong

ABSTRACT

In today's university environment, the phenomenon of students having difficulties with problems related to school, classes, and individual subjects is still happening and its rate is not small. Because each student's situation is unique, a personalized solution such as using an AI-powered university mentor (uMentor) is a promising and potential direction. However, conventional chatbot implementations that rely solely on Retrieval-Augmented Generation (RAG) often struggle to capture the complex relationships and complex structures inherent in university-specific knowledge areas. This limitation can lead to fragmented answers or a lack of in-depth understanding of context. This thesis addresses this challenge by proposing an enhanced uMentor system architecture that integrates Knowledge Graph (KGs) with the RAG framework. KG introduces structured, relational data to complement the unstructured information collected by RAG. This synergistic approach aims to significantly improve uMentor's ability to understand context, synthesize information, and provide more accurate, coherent, and connected responses, thereby enhancing the overall quality and effectiveness of student support services.

Keywords: *University, Knowledge Graph, RAG, Student Support*

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Definition	Description
RAG	Retrieval-Augmented Generation	An AI framework that enhances language models by retrieving relevant information from external sources before generating a response.
FAISS	Facebook AI Similarity Search	A library for efficient similarity search and clustering of dense vectors, used in applications like recommendation systems and image search.
ScaNN	Scalable Nearest Neighbors	A Google library for efficient vector similarity search, particularly optimized for maximum inner-product search on large datasets.
BART	Bidirectional and Auto-Regressive Transformer	A sequence-to-sequence model that uses a denoising autoencoder approach, combining features of BERT and GPT for various NLP tasks.
LLM	Large Language Model	Very large deep learning models trained on vast text data, capable of understanding, generating, and processing human-like text.
TF-IDF	Term Frequency-Inverse Document Frequency	A numerical statistic reflecting how important a word is to a document in a collection or corpus.
BM25	Best Matching 25	A ranking function used by search engines to estimate the relevance of documents to a given search query based on term frequency and document length.
KGs	Knowledge Graph	Structured representations of entities and their relationships, organizing information to provide context and enhance AI understanding.

Introduction

Motivation

The problem of students graduating late or not on time is no longer new in university classrooms. In recent years, this rate in universities is often higher than the average figure. Each student has specific circumstances and reasons, and schools often have difficulty helping each student individually. Taking advantage of the strengths of Artificial Intelligence (AI), a promising solution has been developed called an AI-powered University Mentor, or ”uMentor” [1]. This mentor will receive school data and will represent them to meet and solve student problems in the classroom. The strongest point of this system is the ability to personalize to each student and always be able to help them get important information quickly.

The limitations of existing AI mentoring systems require us to look for smarter approaches and new systems that are capable of understanding and reasoning about the structured nature of university knowledge. The inference and context-awareness capabilities of uMentors are important not only for improving the answer output but also for optimizing institutional resources by reducing the burden on human mentors for routine tasks. This research is motivated by the potential to significantly improve the capabilities of educational chatbots by integrating Knowledge Graphs (KGs), structured knowledge representations, with the retrieval power of RAGs, thus creating a more efficient and reliable AI mentoring tool.

Research Challenges

Despite the potential benefits of chatbots, current implementations, particularly those based primarily on Retrieval Augmentation Generation (RAG) techniques, face many limitations in the information landscape of a university. Traditional RAG models can be very effective at finding relevant text snippets from large document collections but often struggle to synthesize information, gain insight into implicit relationships between different pieces of knowledge (e.g., course prerequisites, faculty professional affiliations, sequential administrative steps), or grasp the semantic nuances of complex queries. This can result in responses that are factually correct at the surface level but lack depth, context, or coherence, potentially causing confusion or requiring the user to ask multiple follow-up questions. Therefore, the performance of uMentors using only the Vector Storage RAG model is not so comprehensive and reliable.

Research Objectives and Contents

This thesis aims to propose an enhanced architecture for the uMentor system by integrating Knowledge Graphs with the Retrieval-Augmented Generation framework in a synergistic manner. The core idea is to leverage the ability of KGs to explicitly model entities (e.g., courses, professors, departments) and their relationships, forming a structured knowledge graph. This structured understanding complements the power of the RAG component in retrieving relevant information from unstructured text sources (such as university websites, handbooks, and policy documents). By combining these approaches, the proposed system aims to generate responses that are not only relevant, but also contextually rich and accurate, and demonstrate a deep understanding of the university's knowledge landscape.

The main contents of this research are:

- Design a new system architecture that effectively integrates Knowledge Graph with the RAG pipeline for the university advisor chatbot (uMentor).
- Develop methods to construct and optimize university-specific knowledge graphs.
- Implement an enhanced prototype of the uMentor system based on the proposed architecture.
- Evaluate the performance of the enhanced RAG system KG compared to other RAG system.

Thesis Scope

This research focuses on improving chatbot responses to queries related to study path advising and administrative procedures in a specific university context (or simulated context based on public data). Therefore, the construction of the Knowledge Graph will be limited to entities and relationships relevant to these domains. The empirical evaluation will use a dataset related to the disciplines and use established natural language processing evaluation techniques. Limitations include the potential scalability of the KG construction process and the inherent complexity of evaluating a natural language generation system.

Thesis Outline

This thesis consists of 5 chapters:

- Chapter 1 provides a summary of the relevant literature on chatbots, Retrieval-Augmented Generation, Knowledge Graphs, and their applications in education.
- Chapter 2 details the proposed methodology, including system architecture, KG design and construction, RAG component setup, and integration strategy.
- Chapter 3 details the implementation of the prototype system, the experimental set-up for evaluation, and discusses the evaluation results.

- Finally, chapter Conclusion concludes the thesis, summarizing the main findings, contributions, and potential future research directions.

Chapter 1

Background and Related Works

This chapter provides a detailed review of the existing literature on developing a university-advising chatbot. We will dive into the basic concepts and advancements in chatbots, especially their applications in the field of education. In addition, we will also mention the core technologies that support the proposed solution: Retrieval-Augmented Generation (RAG), Knowledge Graphs (KGs), and the agent-based architecture coordinating their interaction.

Finally, this chapter explores previous works on the integration of these technologies and identifies the specific research gap that this thesis aims to address. Our goal is to establish the theoretical context, highlight the limitations of existing approaches, and demonstrate the necessity and novelty of the proposed KGs-enhanced RAG framework for the uMentor system.

1.1 Chatbot in Education

Conversational agents, or chatbots, have shown a significant evolution in recent years, from simple rule-based systems to sophisticated AI-enhanced entities capable of understanding and creating human-like dialogue. Their applications have penetrated many fields, and education is a particularly promising area. In higher education institutions, chatbots can be deployed for a variety of purposes, including:

- **Administrative support:** Answer frequently asked questions (FAQs) on admissions, course registration, tuition deadlines, and campus services, thus reducing administrative staff workload.
- **Academic support:** Providing information about course materials, information on courses, and programs within the school. There can also be reminders about assignments or exams.
- **Tutor support:** Provides practice exercises, explanations of concepts, and personalized learning paths for each specific major.

Examples of educational chatbots are Georgia Tech's "Jill Watson" [2] and Deakin University's "Genie" [3] who demonstrate the potential of chatbots to enhance student support. However, many challenges remain, including ensuring the accuracy of the response, maintaining context in long conversations, handling obscure queries, and providing truly personalized solutions. The uMentor

concept addresses these needs head on, aiming for a more comprehensive and intelligent support system than conventional FAQ bots.

1.2 Retrieved-Augmented Generation

Retrieval-augmented generation (RAG) has become a prominent technique for building large language models (LLMs) that are informed about external information and evidence to generate responses [4]. Specifically, the architecture of RAG, unlike traditional LLMs that rely solely on parametric knowledge learned during pretraining, can have knowledge of relevant information from external data stores with the core idea of combining non-parametric memory (retrieval, usually based on dense vector search such as FAISS or ScaNN) with parametric memory (generator, usually a sequence-to-sequence model such as BART or T5 or a decoder-only model such as GPT variants). This approach mitigates issues such as factual inaccuracy, outdated knowledge, and limited transparency often associated with pure parametric models. The standard RAG process consists of two main phases:

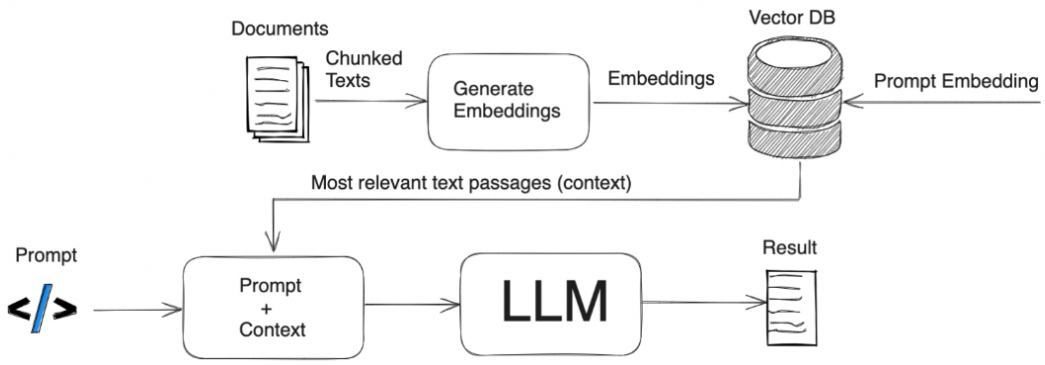


Figure 1.1: Traditional RAG flow

1. **Retrieval:** Given an input query, the retriever searches a large corpus (e.g. university documents, websites, handbooks) for relevant text passages or documents. In this phrase, traditional RAG employs various retrieval techniques to identify relevant context, such as sparse retrieval methods such as TF-IDF and BM25 or dense retrieval methods using learned embeddings, such as DPR and MIPS. Hybrid approaches that combine sparse and dense retrieval are also explored.

2. **Generation:** The generator takes the original query and the retrieved passages as input context and generates the final response. This phrase often uses a general purpose LLM or a chat LLM to take into account the contexts extracted in the previous step to generate answers. Context integration also has many strategies, from a standard approach, such as simple concatenation, to a more sophisticated method that takes advantage of attention mechanisms or cross-attention layers [5] to allow the LLM to selectively focus on relevant parts of the retrieved information.

RAG offers significant advantages over pure generative models, primarily by reducing the probability of hallucination (generating factually incorrect information) and allowing the model to

access updated information without retraining. It relies on concrete evidence for the generated text, thereby increasing reliability. Despite its strengths, a standard RAG model still faces limitations, especially relevant to the semantic data context in universities:

- **Reliance on retrieval quality:** The final quality of the output is highly dependent on the relevance and precision of the retrieved documents. If the information retrieved lacks relevance or conflicts with the question, it can produce a substandard result.
- **Aggregation challenges:** While RAG retrieves relevant snippets, the generator may have difficulty synthesizing information coherently from multiple retrieved snippets, especially when complex reasoning or understanding of relationships between documents is required.
- **Lack of deep understanding of the relations:** RAGs operate primarily on textual similarity and may not capture underlying semantic relationships between entities (e.g. understanding that a particular course is a prerequisite for another course or that a particular professor belongs to a particular research group). This hinders the ability to answer complex, multi-step questions efficiently.

These limitations motivate the exploration of integrating more structured sources of knowledge.

1.3 Knowledge Graphs

Knowledge graphs (KGs) are structured representations of knowledge that model entities (real-world objects, facts, abstract concepts) and the relationships between them [6]. They typically represent knowledge assets of triples, forming a graph structure in which nodes represent entities, and edges represent relationships. Key concepts include:

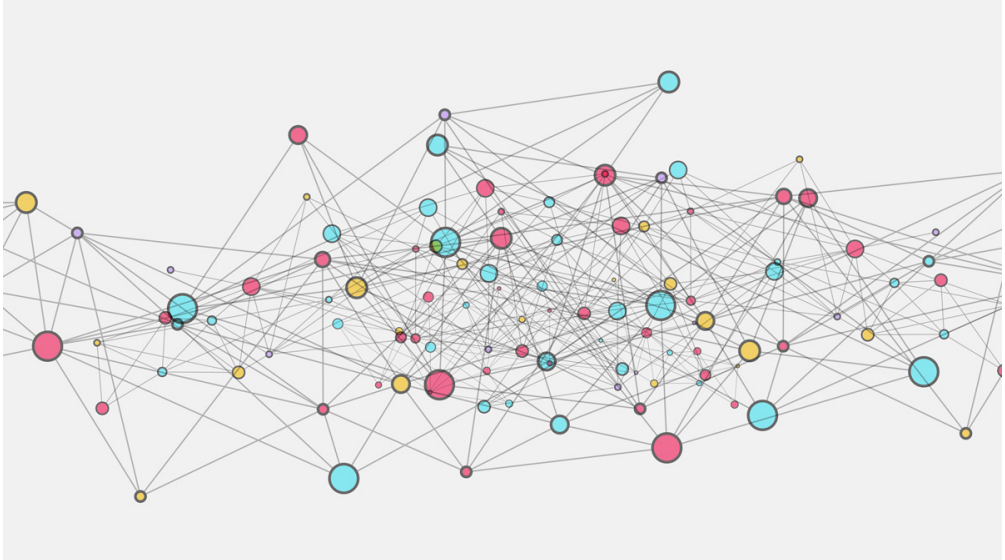


Figure 1.2: Knowledge Graph

- **Entities:** Nodes in the graph (e.g., 'University X', 'Computer Science Department', 'CS101 Course', 'Professor Smith').

- **Relationships:** Edges that connect entities (e.g., 'CS101' is offered by the 'Department of Computer Science', 'Professor Smith' teaches 'CS101', and 'CS101' has prerequisite 'CS099').
- **Schema/Ontology:** Defines the types of entities and relationships allowed in the KG (e.g., defining classes like 'Course', and 'Professor' and attributes like 'teaches', and 'has prerequisite'). Common standards include the RDF (Resource Description Framework) and OWL (Web Ontology Language).

KG construction can be done manually, semiautomatically (e.g., using predefined rules or patterns), or automatically by extracting information from structured sources (databases) or unstructured text.

KGs can be found in a wide range of applications, including semantic search (Google Knowledge Graph), recommender systems (Amazon, Netflix), data integration, and various other critical systems. In the educational domain, KGs can effectively model curricula, course prerequisites, faculty expertise, institutional resources, and administrative structures. This capability allows them to capture complex relational knowledge often overlooked by traditional RAG models.

1.4 Agent-based Orchestration in RAG Systems

As RAG systems become increasingly complex and need to interact with diverse knowledge sources, the concept of an "Agent" based on the Large Language Model (LLM) has emerged as a powerful orchestration architecture. An Agent, in this context, is not just a simple chatbot but an entity that is capable of reasoning, making decisions, and using provided "tools" to complete a task. In advanced RAG architectures, especially when combining unstructured data (such as text in vector storage) and structured data (such as Knowledge Graphs), the Agent plays a central role in deciding how and in what order to retrieve information. Instead of a fixed process, the Agent can be equipped with multiple retrieval tools. This Agent-based approach allows the system to handle queries more flexibly. The agent, with the reasoning capabilities provided by LLM as a foundation, can analyze the input query, select the most appropriate tool, or even execute a sequence of tool calls if necessary. For example, the agent can try to retrieve from the vector storage first and evaluate the results, and if it is not satisfied, it will decide to call the KG query engine to get more precise information about the entity relationships. This dynamic coordination is especially useful in university environments, where questions can range from simple FAQs to complex advice requests that require the aggregation of many different sources and data types.

Using agents with such specialized tools promises to address some of the challenges of traditional RAG, such as selecting the appropriate knowledge source and synthesizing information from heterogeneous sources more efficiently.

1.5 Integrating Knowledge Graphs and Language Models

Recognizing the complementary strengths of KG (structured knowledge, reasoning) and LLM (natural language understanding and generation), significant research has focused on integrating these two models and various integration strategies have emerged:

- **KG-informed Input:** Using information retrieved from a KG to enrich input to an LLM, providing a structured context.
- **KG-based Retrieval:** Use KG queries or traversals to collect relevant facts or subgraphs that can then be used by the language model, potentially replacing or improving text-based retrieval in RAG [7].
- **KG for Generation Control/Verification:** Using KG facts to guide the generation process, ensuring factual consistency, or verifying the factual accuracy of subsequently generated text.

Specifically relevant to this thesis is the concept of **KG-enhanced RAG**. Many researchers have used KG to:

- **Improve Retrieval:** Instruct the retriever to focus on documents related to the entities or relationships identified in the query using the KG lookup [8].
- **Augment Generator Context:** Feed retrieved KG triples/subgraphs directly into the generator along with the fragments of the retrieved text, providing an explicit relational context.
- **Facilitate Synthesis:** Use KG structures to help the generator synthesize information from multiple sources by providing a framework for how different pieces of information relate to each other.

These methods aim to mitigate the limitations of RAG by basing the generation process not only on unstructured text, but also on structured, validated entities and relationships, resulting in more accurate, consistent, and context-aware responses.

1.6 Research Gap and Contribution

The literature review shows significant progress in chatbots, RAGs, KGs, and their integration. However, there is still a specific gap in the adoption and optimization of an integrated system of KGs and RAGs designed specifically for university-advising chatbots to handle complex and comprehensive queries across academic and administrative fields. Although general KG-RAG integration methods already exist, adapting and evaluating them to uMentor's specific requirements, which requires understanding complex university rules, relationships, and procedures, has not been fully explored. Furthermore, optimizing a coordination strategy, possibly through an agent framework, to dynamically choose between unstructured text retrieval and structured KG querying based on the nature of the query in a university context remains an open field. Current education chatbots often rely on simpler FAQ matching, rule-based systems, or standard RAGs, which do not

take full advantage of the potential of structured knowledge for deeper understanding. This thesis aims to fill this gap by:

1. Designing a concrete architecture that tightly integrates a university-specific KG with an RAG pipeline for uMentor.
2. Investigating methods to effectively build a KG from diverse university data sources.
3. Evaluate the tangible benefits (accuracy, coherence, contextual relevance) of this integrated approach over a baseline RAG system in a university context.

Our main contribution is to demonstrate a practical and effective method for enhancing RAG-based educational chatbots with structured knowledge, resulting in a more capable and reliable uMentor system.

1.7 Chapter Summary

This chapter reviews the background material necessary to understand the context and components of this thesis. This chapter covers the role of chatbots in education, the mechanisms and limitations of Retrieval-Augmented Generation, the structure and utility of Knowledge Graphs, current approaches to integrating KG with language models, and the emerging role of agent-based architectures in orchestrating complex retrieval tasks. The review identified a specific research gap related to the optimal integration and coordination of KG and RAG for advanced college advice chatbots. The following chapters will detail the methodology proposed to address this gap and evaluate its effectiveness.

Chapter 2

Proposed Method

This chapter details the methodological framework used in this thesis to design, implement, and evaluate the proposed Knowledge Graph (KGs enhanced Retrieval-Augmented Generation (RAG) system for the University Mentor (uMentor) chatbot. Based on the theoretical foundation and research gaps discussed in Chapter 1, this chapter outlines the specific steps taken to achieve the research objectives outlined in Chapter Introduction. This chapter focuses on the system architecture, the design and construction process of a university-specific Knowledge Graph using advanced LLM techniques, the implementation details of the hybrid information retrieval module, and the core strategy for integrating structured and unstructured information through the ReAct-based generation framework.

2.1 System Architecture

The architecture of the uMentor system is designed to effectively leverage both unstructured text data and structured knowledge derived from those data, enabling sophisticated thinking and reasoning when generating responses. As depicted in Figure 2.1, the system consists of two main phases: an offline data processing phase to extract and index knowledge and an online query processing phase to answer user questions. Offline data processing phase: This phase prepares the knowledge contexts required for online operations.

1. **Document collection and segmentation:** Input documents are collected from various university sources, including websites and internal school circulation documents (PDF, DOCX, TXT, MD, and JSON formats are supported). The raw text is then segmented into manageable chunks using, ensuring that the segmentation is done based on the number of tokens for good compatibility with downstream LLMs and API calls to avoid rate limits and over-budget API calls. Details is presented in Figure 2.2.

2. **Parallel knowledge extraction:** The document chunks are then fed into two parallel pipelines

    - **Vector storage pipeline:** The chunks are embedded into high-dimensional vectors using a Hugging Face pre-trained sentence transformation model (sentence-transformers/all-MiniLM-L6-v2). These embeddings are stored and indexed in a FAISS (Facebook AI Similarity Search) vector store (IndexFlatL2 for Euclidean distance searching), managed

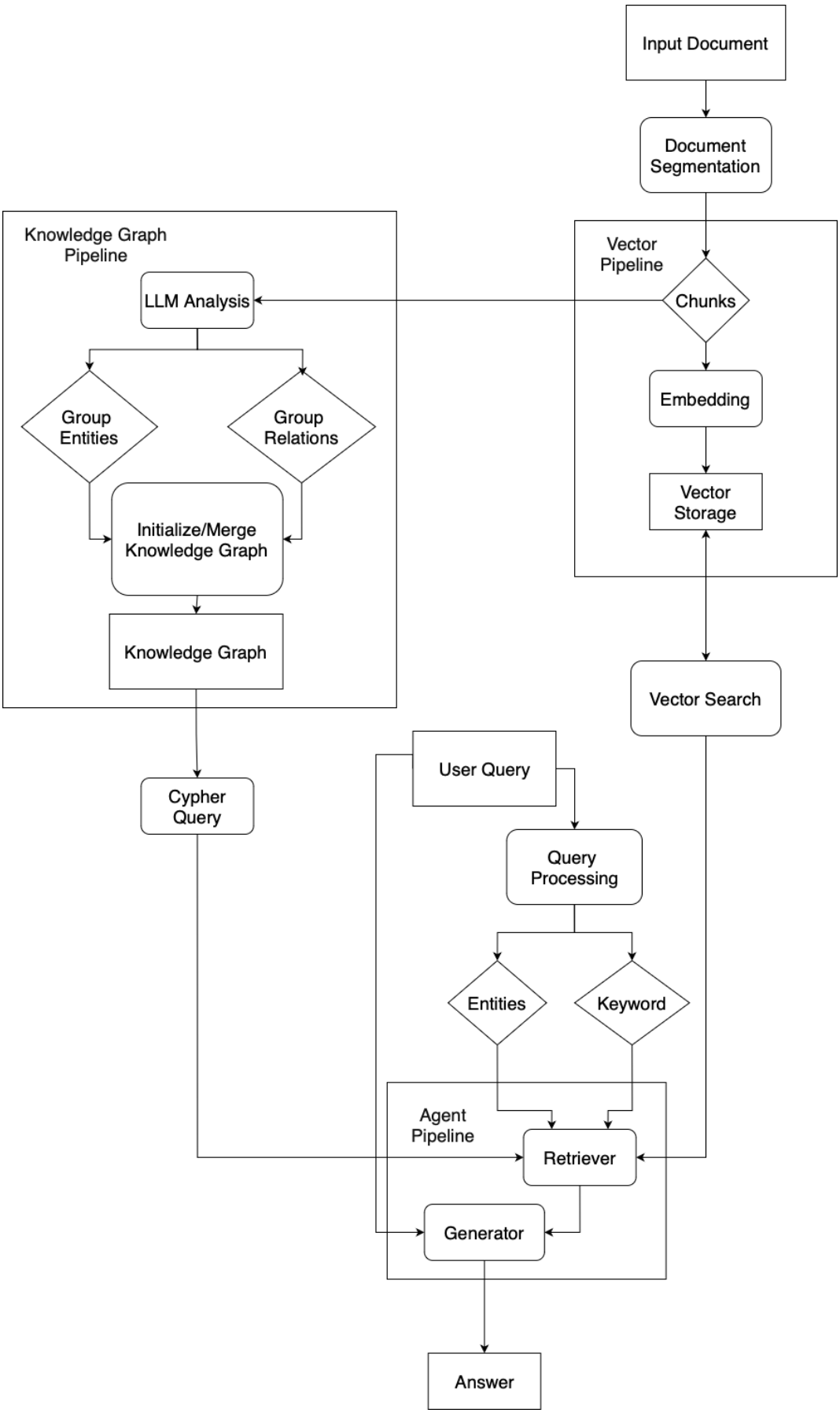


Figure 2.1: uMentor Dataflow

```python
from langchain_community.document_loaders import DirectoryLoader, TextLoader,
PyPDFLoader, Docx2txtLoader, UnstructuredMarkdownLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

KNOWLEDGE_DIR = "./uData/UET"

def load_and_chunk_documents(directory):
   loaders = [
       DirectoryLoader(directory, glob="**/*.pdf", loader_cls=PyPDFLoader),
       DirectoryLoader(directory, glob="**/*.docx", loader_cls=Docx2txtLoader),
       DirectoryLoader(directory, glob="**/*.txt", loader_cls=TextLoader),
       DirectoryLoader(directory, glob="**/*.md",
loader_cls=UnstructuredMarkdownLoader),
   ]
   docs = []
   for loader in loaders:
       docs.extend(loader.load())

   text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
       chunk_size=1028, chunk_overlap=56
   )
   chunks = text_splitter.split_documents(docs)
   return chunks

documents = load_and_chunk_documents(KNOWLEDGE_DIR)
```

Figure 2.2: Segmantation Phrase

by LangChain's FAISS wrapper with InMemoryDocstore, and cached locally for efficient retrieval.

```python
import faiss
from langchain_community.docstore.in_memory import InMemoryDocstore
from langchain_community.vectorstores import FAISS
from uuid import uuid4
from langchain_core.documents import Document

VECTOR_STORAGE_PATH = "VectorStorage"

index = faiss.IndexFlatL2(len(embedding.embed_query("hello world")))

vector_store = FAISS(
    embedding_function=embedding,
    index=index,
    docstore=InMemoryDocstore(),
    index_to_docstore_id={},
)
uuids = [str(uuid4()) for _ in range(len(documents))]
vector_store.add_documents(documents=documents, ids=uuids)
vector_store.save_local(VECTOR_STORAGE_PATH)
```

Figure 2.3: Vector storage pipeline

- **Knowledge Graph Pipeline:** Text blocks are fed into LangChain's LLMGraphTransformer, which uses a robust external LLM (e.g. GPT-4o configured with a low temperature for consistency) guided by carefully designed prompts (detailed in Section 2.2.3). This process extracts entities and relationships directly from Vietnamese text. The extracted graph documents are then used to construct the Neo4j graph database.

3. **KG Refinement - Node Merging:** After the initial clustering, the Neo4j graph undergoes a critical refinement step to merge duplicate entities. A custom iterative process using Cypher query (detailed in Section 2.2.5) identifies candidate pairs based on the degree of similarity

```python
from langchain_neo4j import Neo4jGraph, Neo4jVector
from langchain_experimental.graph_transformers import LLMGraphTransformer

llm_transformer = LLMGraphTransformer(
    llm=llm,
    prompt=extract_prompt
)

graph_documents = llm_transformer.convert_to_graph_documents(documents)
graph = Neo4jGraph(url=url, username=username, password=password,
refresh_schema=False)
graph.add_graph_documents(
    graph_documents,
    baseEntityLabel=True,
    include_source=True
)
```

Figure 2.4: Knowledge Graph Pipeline

of the Jaccard Index between their neighbors (computed using the APOC plugin) exceeding a

```python
SIMILARITY_THRESHOLD = 0.56
EXCLUDED_LABELS = {'_Bloom_Perspective_', '_Bloom_Scene_'}

MERGE_NODES_QUERY = """
MATCH (nodeToKeep), (nodeToRemove)
WHERE id(nodeToKeep) = $idToKeep AND id(nodeToRemove) = $idToRemove
CALL apoc.refactor.mergeNodes([nodeToKeep, nodeToRemove], {properties:'overwrite'}) YIELD node
RETURN id(node) as mergedNodeId
"""

def find_all_merge_pairs(driver, labels_to_process, threshold, database_name):
    all_pairs = []
    with driver.session(database=database_name) as session:
        for label in labels_to_process:
            if '{' in label or '}' in label or '`' in label:
                continue
            try:
                dynamic_query = f"""
                MATCH (n1:`{label}`), (n2:`{label}`)
                WHERE elementId(n1) < elementId(n2) // Dùng elementId()
                WITH n1, n2,
                     [(n1)--(neighbor) | id(neighbor)] AS neighbors1_ids,
                     [(n2)--(neighbor) | id(neighbor)] AS neighbors2_ids
                WITH n1, n2, neighbors1_ids, neighbors2_ids,
                     apoc.coll.intersection(neighbors1_ids, neighbors2_ids) AS intersection_ids,
                     apoc.coll.union(neighbors1_ids, neighbors2_ids) AS union_ids
                WITH n1, n2, neighbors1_ids, neighbors2_ids, intersection_ids, union_ids,
                     size(intersection_ids) AS intersectionSize,
                     size(union_ids) AS unionSize
                WHERE unionSize > 0 AND toFloat(intersectionSize) / unionSize >= $threshold
                WITH n1, n2, size(neighbors1_ids) as degree1, size(neighbors2_ids) as degree2
                WITH n1, n2,
                    CASE WHEN degree1 >= degree2 THEN n1 ELSE n2 END AS nodeToKeep,
                    CASE WHEN degree1 >= degree2 THEN n2 ELSE n1 END AS nodeToRemove
                RETURN id(nodeToKeep) AS idToKeep, id(nodeToRemove) AS idToRemove
                """

                result = session.run(dynamic_query, threshold=threshold)
                pairs = [record.data() for record in result]
```

Figure 2.5: Merging Phrase

threshold (0.56). Duplicate nodes are then merged using apoc.refactor.mergeNodes.

This offline phase generates two primary knowledge sources: the FAISS Vector Storage for semantic text retrieval and a refined Neo4j Knowledge Graph for structured data and relational retrieval. **Online Query Processing Phase:** This phase handles user interactions and generates responses.

1. **User Interaction & Query Processing:** Users interact with the system through a user interface (implemented using Streamlit), as shown in Figure 2.6. User natural language queries are received and go through a processing step to extract: (a) keywords for semantic search and (b) named entities. Entity extraction uses dedicated LLM calls (e.g. GPT-4o-mini via a custom OpenAIChatInterface wrapper that ensures structured output that complies with the Pydantic schema).

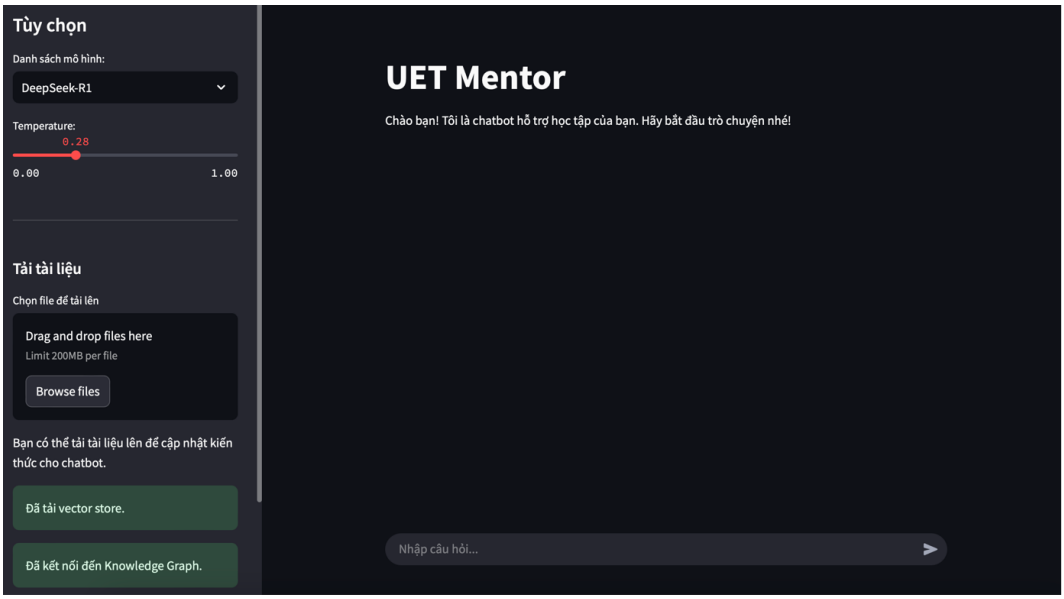


Figure 2.6: User Interface

2. **ReAct-driven Information Gathering and Reasoning:** Instead of a traditional fixed, simple RAG pipeline, the core of the online processing relies on an LLM agent operating within the Reasoning and Acting (ReAct) framework (implemented via a custom AugmentedLLM layer). This agent coordinates the retrieval and generation process:

   - The agent receives the user query and initiates an internal reasoning loop (<thinking>...</thinking>).
   - It determines the information needed and decides to use a specifically defined tool.
   - The architecture has two main Retrieval process: the FAISS Vector Storage query keyword (Section 2.3.1) and the Neo4j Knowledge Graph query extracted entities (Section 2.3.2).
   - The tool returns the aggregated unstructured text and the structured KG information (serialized as text) to the agent.
   - The agent incorporates this retrieved context into its reasoning process to supplement and refine its understanding and then decides whether to invoke further tools

```
<thinking> Spark Streaming is an extension of Apache Spark that provides scalable,
high-throughput, fault-tolerant stream processing of live data streams. It enables
real-time data processing through several key components and concepts:
1. **Micro-batch Processing:** Spark Streaming processes data in small batches
(micro-batches) instead of processing each record individually. This allows for
efficient processing while still enabling near real-time analysis.
2. **DStream (Discretized Stream):** DStreams are the core abstraction in Spark
Streaming. They represent a stream of data that is divided into micro-batches. Each
DStream is created from a variety of sources, such as Kafka, Flume, or TCP sockets,
and can be processed using Spark's rich set of APIs.
3. **Integration with Spark Ecosystem:** Spark Streaming seamlessly integrates with
the core Spark engine, allowing users to leverage existing Spark capabilities such
as machine learning libraries (MLlib), graph processing (GraphX), and SQL (Spark
SQL). This integration enables complex data processing workflows and analytics.
4. **Fault Tolerance:** Spark Streaming provides fault tolerance through data
replication and lineage information. If a node fails, Spark can recover lost data
by reprocessing it from the source or using the lineage information to recompute
lost computations.
5. **Windowed Operations:** Spark Streaming allows for windowed operations, where
data can be processed over a defined interval, enabling operations such as sliding
windows and tumbling windows. This is useful for aggregating data over specific
time periods.
6. **Backpressure Handling:** Spark Streaming can manage backpressure, which occurs
when the data ingestion rate exceeds the processing rate. It can automatically
adjust the rate of data processing to avoid overwhelming the system.
7. **Support for Multiple Sources and Sinks:** Spark Streaming can ingest data from
various sources and write the processed output to multiple sinks, such as
databases, file systems, or messaging systems.
With these features, Spark Streaming enables organizations to handle real-time
analytics, monitoring, and event-driven applications effectively. However, it is
essential to consider the trade-offs between batch processing and true stream
processing, as Spark Streaming operates on micro-batches rather than continuous
data streams.</thinking>

FINAL ANSWER: Spark Streaming enables real-time data processing through micro-batch
processing, the use of DStreams (Discretized Streams), seamless integration with
the Spark ecosystem, fault tolerance, windowed operations, backpressure handling,
and support for multiple data sources and sinks. It provides an effective framework
for real-time analytics and event-driven applications.
```

Figure 2.7: ReAct Answer

3. **Final Response Generation:** Once the ReAct agent is confident that it has gathered enough information and completed its inference steps, it synthesizes a final natural language response, based on the retrieved context and its internal thought process, and presents it to the user (as illustrated in Figure 2.7).

2.2 Designing and Building Knowledge Graphs

The quality and structure of the knowledge graph are paramount to the system's ability to answer complex relational queries.

2.2.1 Schema/Semantic Design

Based on the target domain (academic advising and university administrative procedures), we designed a schema that defines the core entity types and relationship types that are closely related

to this domain. The main entity types include Course, Professor, Department, Program, Policy, AcademicTerm, Building, ServicePoint, etc. The main relationship types include hasPrerequisite, isTaughtBy, belongsToDepartment, leadsToDegree, appliesTo, locatedIn, contactPerson, etc. Standard vocabularies like RDF Schema (RDFS) or Web Ontology Language (OWL) principles were considered to ensure consistency and potential interoperability, although a custom schema tailored to the specific organizational context was mainly developed.

2.2.2 Data Collection and Preprocessing

Data is collected from various sources of the university, including official websites, course catalogs, handbooks, and directories (supporting PDF, DOCX, TXT, MD, and JSON formats). The main preprocessing step involves segmenting the text into parts based on the number of tokens (part size 2056 tokens, 185 overlapping tokens) using RecursiveCharacterTextSplitter.from_tiktoken_encoder to be compatible with LLM and embedding models. Splitting text segments based on the number of tokens has two major benefits: 1) Easy to control the maximum context length of the LLM model being used and 2) Ensuring the maximum number of API calls per unit of time and avoiding Rate Limit errors. The reason for choosing the two numbers 2056 and 185 is also based on personal experience and to satisfy the two conditions just mentioned. As we can see from the example below, a small number of tokens (chunk_size) can lead to a large number of documents (chunks) that needed to be processed and and this number (1675) will definitely give us an error of exceeding the number of APIs we are allowed to call in a period of time.

```
    text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
        chunk_size=1024, chunk_overlap=100
    )
    chunks = text_splitter.split_documents(docs)
    return chunks

documents = load_and_chunk_documents(KNOWLEDGE_DIR)
print(len(documents))
✓ 27.1s

1675
```

Figure 2.8: Chunking Phrase

2.2.3 Graph Transformer and Prompt Technique

Leveraging the capabilities of modern LLM, structured knowledge extraction is performed using LangChain's LLMGraphTransformer function combined with a powerful underlying LLM (e.g. GPT-4o). This function essentially writes a default and automatic prompt to send to the LLM and asks it to search for entities and relationships.

However, we decided to rewrite the prompt ourselves to better suit uMentor. Specifically, our system prompt instructs LLM to: act as a top-level information extractor to construct KGs from Vietnamese text; and capture entities (Book, Chaper, Section, Subsection, Concept, Example, Exercise, Lecture, Course, Class, Major, Reference, Document, Online Document, etc.) and

```
PROMPTS["BOOK"] = """# Knowledge Graph Instructions
## 1. Overview
You are a top-tier algorithm designed for extracting information in structured formats
to build a knowledge graph focused on books data.
- **Goal**: Extract entities and relationships accurately based on the provided text
snippets. Capture as much relevant information as possible without inventing data.
- **Nodes**: Book, Chapter, Section, Subsection, Concept, Example, Exercise.
- **Relationships**: CONTAINS_CHAPTER, HAS_SECTION, MENTIONS_CONCEPT,
PROVIDES_EXAMPLE, HAS_EXERCISE, LOCATED_ON_PAGE.

## 2. Node Types and Properties (Schema)
* **`Book`**:
    * `title` (string)
    * `author` (string)
    * `edition` (string, optional)
    * `publisher` (string, optional)$
    * `year` (string, optional)
* **`Chapter`**:
    * `chapter_number` (int)
    * `title` (string)
* **`Section`**:
    * `section_number` (int)
    * `title` (string)
* **`Subsection`**:
    * `subsection_number` (int)
    * `title` (string)
* **`Concept`**:
    * `name` (int)
    * `description` (string)
* **`Example`**:
    * `description` (string)
* **`Exercise`**:
    * `description` (string)
    * `difficulty` (string, optinal)

## 3. Relationship Types
* `CONTAINS_CHAPTER`: (`Book`) - [:GIẢNG_DẠY] -> (`Chapter`)
* `HAS_SECTION`: (`Chapter`) - [:HAS_SECTION] -> (`Section`)
* `HAS_SUBSECTION`: (`Section`) - [:HAS_SUBSECTION] -> (`Subsection`)
* `MENTIONS_CONCEPT`: (`Chapter`)/Section - [:MENTIONS_CONCEPT] -> (`Concept`)
* `PROVIDES_EXAMPLE`: (`Concept`) - [:PROVIDES_EXAMPLE] -> (`Example`)
* `HAS_EXERCISE`: (`Chapter`)/Section - [:HAS_EXERCISE] -> (`Exercise`)

## 4. Extraction Guidelines
- The user input usually contain 1 book, so you need to keep the name of the book
concise, do not create many entities for 1 books.
- Extract entities and relationships strictly based on the schema and do not omit any
entities/relations that satisfied.
```

Figure 2.9: Custome Prompt

relationships by convention; use basic, consistent node labels; use human-readable identifiers for nodes; perform reference resolution (using the most complete identifier); and strictly adhere to the specified output format. This prompt-driven approach uses the comprehension and generation capabilities of LLM for complex extraction tasks directly from blocks of text. Figure 2.9 is a custom prompt used specifically to process teaching input.

2.2.4 Knowledge Graph Database

The extracted graph documents, representing entities and their relationships, were collected and stored persistently in the Neo4j graph database, version 5.26.1. Neo4j was chosen for its native graph processing capabilities, Cypher query writing, integration with many Python libraries, and local usability (Figure 2.10).

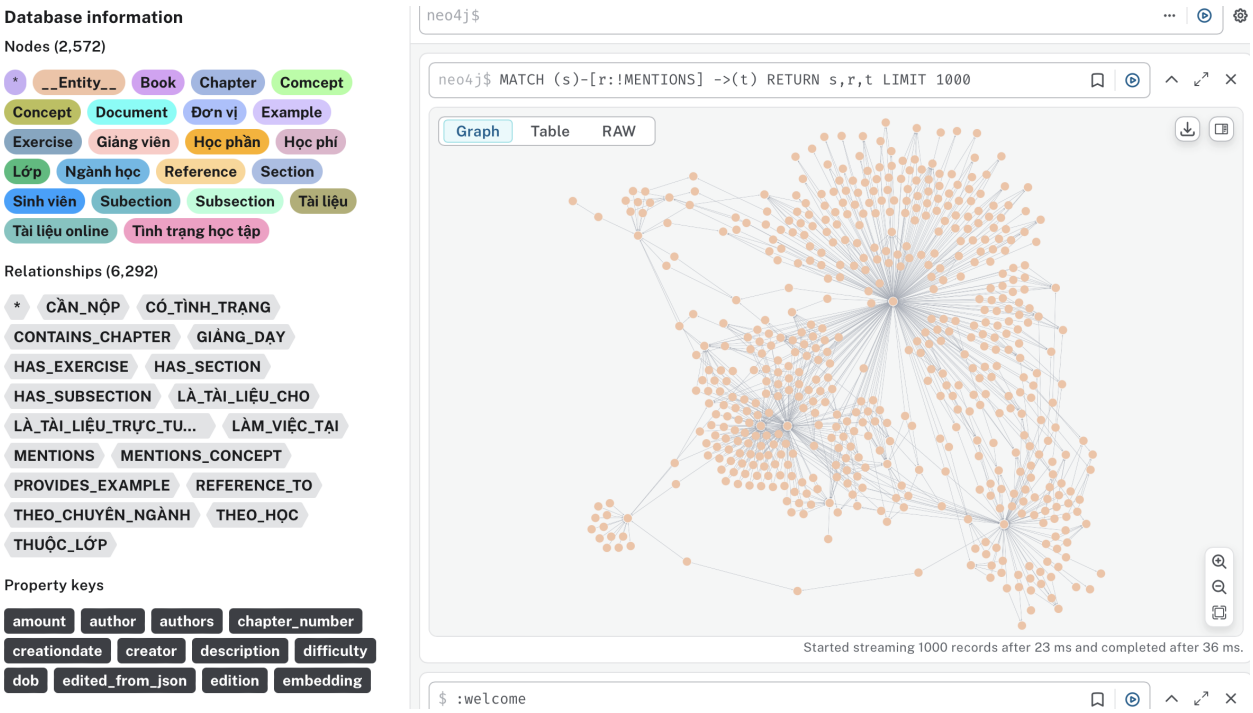


Figure 2.10: uMentor Knowlege Graph

2.2.5 Entity Resolution and Refinement Using Jaccard Index

Entity Resolution or Coreference Resolution is an essential task that needs to be resolved in order to optimize and reduce the noise of the Knowledge Graph. This is a complex problem and there is usually no perfect solution with just a single function. Some common approaches to this problem are:

1. Pre-processing: Using a specialized Coreference Resolution tool/model when extracting entities and their relations. This tool will read the original text and replace different mentions (like ”Potter” or ”he”) with a canonical name (e.g. ”Harry Potter”).

2. Post-processing - Graph-based (Heuristic): After building the initial graph, we will have to write Cypher queries or use a graph processing library (like NetworkX in Python combined with the Neo4j driver) to:

   - Find pairs of ”candidate” nodes that might be the same entity (e.g. similar names, same Type).
   - For each pair, get a list of their neighbors and their relations.

Table 2.1: Pro and Cons of Coreference Resolution Approaches

Approaches	Advantages	Disadvantages
Pre-processing	Helps the input LLM receive "cleaner" text. Reducing the possibility of creating duplicate nodes from the beginning.	Need to find good co-reference tools for some languages and domains. Libraries like spaCy or specialized API services can be useful but still require additional training for the model. Depends on the chunking process before.
Post-processing (Graph-based)	Take advantage of existing graph structure.	Requires careful heuristic and threshold design. Can be computationally expensive. Cypher queries can be complex.
Post-processing (LLM)	Takes advantage of LLM's semantic reasoning capabilities.	Expensive LLM API overhead. Prompts need to be carefully designed for this comparison task. LLM's results are not always perfect.

- Calculate the similarity based on the number of common neighbors and the type of common relations.
- If the similarity exceeds a threshold, decide to merge the two nodes. Merging typically involves: selecting a node to keep, transferring all relations from the deleted node to the kept node, and then deleting the redundant node.

3. Post-processing - Using LLM: Similar to (b) to find candidate pairs. Then, instead of using hard heuristics, we will create a new prompt for LLM. This prompt will provide information about the two candidate nodes (name, type) and some context (their important relationships and neighbors). You ask LLM: "Based on this information, do '[Node A]' and '[Node B]' refer to the same real-world entity?". Based on LLM's answer (Yes/No) to decide to merge. An example using this approach is LightRAG [9].

Although we have LLM perform a reprocessing step when checking for coreference resolution during entity and relation extraction (explained in Section 2.2.3), our graph is still not completely free enough from graph "noise" and singleton entities. Therefore, to improve data quality and handle the inevitable duplicates from the extraction process, we will use an additional merging step after initializing the knowledge graph using Jaccard statistics.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Figure 2.11: Jaccard Formulation

Using this formula, the Jaccard index will be calculated by dividing the "Number of Com-

mon Neighbors" by the "Total Number of Their Unique Neighbors". We will then initialize a custom iterative process to dentify candidate pairs of nodes (within the same label) for merging based on the similarity of their graph neighborhoods.The Jaccard index is ccalculated between the sets of neighborhoods for each pair using Neo4j APOC library functions (apoc.coll.intersection, apoc.coll.union). If Jaccard similarity exceeds a predefined threshold (SIMILARITY_THRESHOLD = 0.56), the pair of nodes is considered a duplicate. The node with a higher degree (more connections) is usually retained, and the remaining node is merged into it using apoc.refactor.mergeNodes. This iterative process continues until no more pairs reach the merge threshold in a given number of iterations, or the maximum iteration limit is reached, ensuring graph consistency. The explanation for the rate 0.56 is that since the Jaccard index ranging from 0 to 0.35 may not be enough to effectively distinguish between the two types of alert in that particular case [10], a Jaccard index value around 0.5 usually gives slightly better results.

```
2025-04-13 23:45:07,947 - INFO - --- Quá trình hợp nhất hoàn tất ---
2025-04-13 23:45:07,947 - INFO - Tổng cộng đã hợp nhất 128 cặp node.
2025-04-13 23:45:07,947 - INFO - Đã đóng kết nối Neo4j.
2025-04-13 23:45:07,947 - INFO - Tổng thời gian thực thi script: 168.32 giây.
```

Figure 2.12: Merging Output

2.3 Information Retrieval and Generation Module

This module handles the retrieval of online information from both knowledge sources and the generation of the final answer, coordinated by the ReAct agent.

2.3.1 Unstructured Information Retrieval

FAISS (Facebook AI Similarity Search) is an open source library developed by Facebook AI Research (FAIR). FAISS is specifically designed to perform similarity search and clustering of dense vectors efficiently at scale. With the ability to handle a large volume of vector embedding, FAISS provides a variety of indexes and optimal data compression methods, allowing a balance between query speed, accuracy, and memory usage.

The FAISS vector repository, which contains embeddings (all-MiniLM-L6-v2) of token-based text blocks, is queried using the embeddings from the user query. The Retrieval uses the Euclidean distance to calculate the similarity. Euclidean distance (IndexFlatL2) is a way to calculate the distance between 2 vectors in Euclidean space. It's the straight fly distance between these 2 vectors.

For two vectors in N-dimensional space (The dimensionality of the vector is decided by the embedding model and in our case this number is 384), the L2 distance is calculated by the square root of the sum of squares of the differences between corresponding components of the two vectors. So, the smaller the score returned by the function, the smaller the distance between the two vectors in space or the more similar the semantics they carry.

Our system use a custom filtering logic to retrieved chunks based on their similarity scores: blocks with scores below a threshold (0.55) are prioritized, selecting the top 5 blocks if there are

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}.$$

Figure 2.13: L2 Formulation

multiple blocks; otherwise, the top 6 blocks with the highest overall scores are selected. This is intended to balance relevance and diversity in the context of retrieved text. The selected block text is formatted and passed to the aggregator.

```
if chunks_below_threshold:
    if len(chunks_below_threshold) > 5:
        chunks_below_threshold.sort(key=lambda item: item[5])  # Sort up

        final_chunks = [doc for doc, score in chunks_below_threshold[:5]]
    else:
        final_chunks = [doc for doc, score in chunks_below_threshold]
else:\
    results_with_scores.sort(key=lambda item: item[6])
    final_chunks = [doc for doc, score in results_with_scores[:6]]
```

Figure 2.14: Unstructured Retrieval Conditions

At this step, any embedding model can be used (”text-embedding-3-large” from OpenAI, ”nomic-embed-text” from Ollama,...). However, we decided to use ”all-MiniLM-L6-v2” from HuggingFace because it is one of the best models of this provider, as well as an opensurce that can run completely locally (can be trained if needed). Table 2.2 showns that ”all-MiniLM-L6-v2” is 5 times faster than ”all-mpnet-base-v2” but still offers good quality with very low model size.

2.3.2 Structured Information Retrieval

Neo4j is one of a popular graph databases, featuring a native graph architecture, where relationships are stored directly and peer-to-peer with nodes. It allows us to perform graph traversal and query that link to data quickly and efficiently, outperforming traditional relational databases when dealing with highly connected data. Data querying in Neo4j is done through the Cypher language, a declarative graph query language. In our RAG system, Neo4j serves as the primary repository for the Knowledge Graph.

Invoking via the ReAct engine, the Structured Information Retrieval retrieves information from the Neo4j KG based on the entities identified in the query (extracted using GPT-4o-mini via OpenAIChatInterface). It combines:

- Neo4j full-text search on entity IDs (using fuzzy matching over the generated query string) to locate starting nodes
- Cypher pattern matching queries to traverse the graph and retrieve relevant connected nodes and relationships (excluding specific metadata relationships like MENTIONS), formatted as textual descriptions. kept node, and then deleting the redundant node.

Table 2.2: Compare Embedding Models

Model Name	Performance Sentence Embeddings (14 Datasets)	Performance Sentence Embeddings (6 Datasets)	Avg. Performance	Speed	Model Size (MB)
all-mpnet-base-v2	69.57	57.02	63.3	2800	420
multi-qa-mpnet-base-dot-v1	66.76	57.6	62.18	2800	420
all-distilroberta-v1	68.73	50.94	59.84	4000	290
all-MiniLM-L12-v2	68.70	50.82	59.76	7500	120
multi-qa-distilbert-cos-v1	65.98	52.83	59.41	4000	250
all-MiniLM-L6-v2	68.06	49.54	58.80	14200	80
multi-qa-MiniLM-L6-cos-v1	64.33	51.83	58.08	14200	80
paraphrase-multilingual-mpnet-base-v2	65.83	41.68	53.75	2500	970
paraphrase-albert-small-v2	64.46	40.04	52.25	5000	43
paraphrase-multilingual-MiniLM-L12-v2	64.25	39.19	51.72	7500	420
paraphrase-MiniLM-L3-v2	62.29	39.19	50.74	19000	61
distiluse-base-multilingual-cased-v1	61.30	29.87	45.59	4000	480
distiluse-base-multilingual-cased-v2	60.18	27.35	43.77	4000	480

Entities extracted from the user query will start to retrieve from Neo4j KG. This process involves two steps:

- Node Identification: Neo4j's built-in full-text search index (on the id attribute of __Entity__ nodes) is queried using an opaque version of the entity name (extracted using GPT-4o-mini via OpenAIChatInterface) to accurately locate the corresponding nodes in the graph.
- Neighborhood Expansion: Once the entity nodes are identified, Cypher queries are executed to retrieve their relevant neighborhood information. This typically involves retrieving directly connected nodes and relationships (excluding metadata relationships such as MENTIONS),

```
graph.query(
    "CREATE FULLTEXT INDEX entity IF NOT EXISTS FOR (e:__Entity__) ON EACH [e.id]")

def remove_lucene_chars(text: str) -> str:
    return re.sub(r'[+\-&|!(){}[\]^"~*?:\\]', ' ', text)

def generate_full_text_query(input: str) -> str:
    full_text_query = ""
    words = [el for el in remove_lucene_chars(input).split() if el]
    for word in words[:-1]:
        full_text_query += f" {word}~2 AND"
    full_text_query += f" {words[-1]}~2"
    return full_text_query.strip()
```

Figure 2.15: Structured Retrieval Funtions

limited to a reasonable number (e.g., 50) per entity to keep the context concise.

```
response = graph.query(
    """
    CALL db.index.fulltext.queryNodes('entity', $query, {limit:2})
    YIELD node, score
    CALL {
      WITH node
      MATCH (start)-[r:!MENTIONS]->(end)
      WHERE start = node OR end = node
      RETURN start, r, end, start.id + ' - ' + type(r) + ' -> ' + end.id AS relationship
    }
    RETURN relationship, start, end LIMIT 50
    """,
    {"query": full_text_query},
)

nodes = {}
relationships = []
for el in response:
    relationship = el['relationship']
    start = el['start']
    end = el['end']
    relationships.append(relationship)
    if start['id'] not in nodes:
        nodes[start['id']] = start
    if end['id'] not in nodes:
        nodes[end['id']] = end

entity_result = f"Entity: {entity}\nNodes:\n"
for node_id, node in nodes.items():
    properties = {k: v for k, v in node.items() if k != 'id'}
    entity_result += f"ID: {node_id}, Properties: {properties}\n"
entity_result += "\nRelationships:\n"
entity_result += "\n".join(relationships) + "\n"
result += entity_result
```

Figure 2.16: Neo4j Cypher Queries

- Collect all properties fields: For each node that we got in the previous step, we will also filter and get all the properties values that come with it.

2.3.3 ReAct-enhanced Generator

The final response generation is handled by the custom AugmentedLLM class, configured with a generator model (e.g. GPT-4o-mini, temperature = 0.85) within the Reasoning and Acting (ReAct) framework (use_react=True). This class implements the ReAct logic, along with system prompts instructing, managing the interaction with the LLM, handling the lifecycle of tool calls (parsing requests, executing tools via registered handlers, formatting results), maintaining the conversation history including reasoning steps (<thinking>...</thinking>), and processing the response stream. It allows the LLM to intelligently utilize both the retrieval tool to gather evidence and construct well-reasoned answers.

```
react_prompt = """
You are a highly capable and thoughtful assistant that employs a ReAct (Reasoning
and Acting) strategy. For every query, follow this iterative process:

1. **Initial Reasoning:** Start by writing a detailed chain-of-thought enclosed
within <thinking> and </thinking> tags. This should include your initial reasoning,
hypotheses, and any uncertainties.
2. **Tool Invocation:** If you determine that further information is needed, or
that a specific computation or external lookup is required.
   Then, wait for the result of that tool call.
3. **Iterative Refinement:** Once you receive the tool's output, update your chain-
of-thought inside a new <thinking>...</thinking> block that integrates the tool's
result. Continue to iterate—refining your chain-of-thought and making additional
tool calls if necessary. Do not finalize your answer until you have thoroughly
considered all angles and feel that you have thought about the best response.
4. **Final Answer:** After multiple iterations of reasoning and tool use, and once
you are confident that you have fully addressed the query, provide your final
answer clearly labeled as FINAL ANSWER.
5. **Clarity and Transparency:** Include all your reasoning steps (the
<thinking>...</thinking> blocks with any tool invocations and their results) along
with the final answer so that the user can follow your complete thought process.

Use this structured approach to handle complex queries, ensuring that each stage of
your internal reasoning is transparent by using the <thinking> tags. Bellow if your
specific role in this case:
"""
```

Figure 2.17: ReAct Prompt

2.4 KG-RAG Integration Strategy

The coordination between the KG and RAG components is achieved by providing separate retrieval capabilities as callable 'tools' to the Agent (AugmentedLLM) operating within the ReAct framework. This strategy (Agent-driven Sequential Tool Usage with ReAct) emphasizes the Agent making sequential and conditional decisions about when to use each knowledge source:

1. **Query Understanding:** The initial user query is processed to identify both semantic keywords and specific named entities (using an LLM call like GPT-4o-mini).

2. **Tool Definition:** Two separate tools are registered with the AugmentedLLM Agent:

- unstructured_retriever: This tool description tells the Agent that it performs a semantic search on the FAISS vector store to retrieve relevant unstructured text fragments based on an input query or keyword.
- structured_retriever: This tool description tells the Agent that it queries the Neo4j Knowledge Graph to find structured facts and relationships related to the identified entities.

3. **Agent-driven Sequential Retrieval (ReAct Cycle):** During its reasoning cycle (<thinking>... </thinking>), the Agent will:
   - Initial Action Decision: Determine the information needed based on the user's query and decide what tool to use to retrieve the necessary information.
   - Re-prompting the query: The agent will rewrite the syntax of the query so that it can use the tool or the tool returns useful results to generate the answer.
4. **Tool Execution and Iterative Reasoning:** The agent executes the call to the first tool:
   - Context Evaluation: The agent evaluates the context it has just received. It asks itself: Is this information sufficient to answer the question? Is it directly relevant? Is there a need for information about specific relationships that are not explicitly shown in the text?
   - Decision for second Tool: Only if the Agent determines that the unstructured context is insufficient, inaccurate, or the nature of the question requires explicit structured knowledge (e.g., "What is the prerequisite for CS101?", "Which professor is in department X?"), does it decide that it needs to call structured_retriever.
   - Second Tool Execution (Conditional): If the decision in step 5b is yes, the Agent constructs a request for new tool and executes the call.
5. **Final Synthesis and Generation:** The Agent synthesizes information from one or both tools (depending on the decision in step 5) during its final reasoning. Based on this evaluated and synthesized context, the Agent generates the final answer for the user.

This sequential and conditional integration approach allows the system to dynamically decide when and how to retrieve structured knowledge, only when it is needed after it has attempted to exploit unstructured knowledge. This not only increases efficiency (avoiding unnecessary KGs queries) but also ensures that structured knowledge is used in a targeted way to address the limitations of relying solely on text, resulting in more robust and contextually relevant answers than static RAG or parallel fusion approaches from the outset.

2.5 Chapter Summary

This chapter presents the comprehensive methodology underpinning the KG-enhanced uMentor system. It delineated the dual-pipeline system architecture, encompassing offline knowledge processing (LLM-based KG extraction, Jaccard merging, FAISS indexing) and online query handling. Critically, it detailed the integration strategy, where hybrid retrieval (combining semantic search

and graph queries) is provided as a tool to a ReAct-based LLM agent (AugmentedLLM). This allows the agent to collect and dynamically reason about structured and unstructured information. Finally, we describes the specific techniques and technologies used, including token-based segmentation, prompt-driven knowledge extraction, neighborhood-based Jaccard similarity to refine the KG, and the components of the retrieval and generation module.

Chapter 3

Experiments and Evaluation

Following the methodological framework detailed in Chapter 3, this chapter outlines the specific tools, libraries, configurations, and procedures used to build the system components, including the data processing pipeline, knowledge graph construction, vector storage generation, hybrid retrieval mechanism, and ReAct-based generative agent. Furthermore, this chapter specifies the setup for comparative experiments designed to evaluate the system performance against the baseline system (Naive RAG) and the SelfRAG framework, detailing the dataset preparation, configuration of the tested systems, and evaluation procedure using the RAGChecker framework.

3.1 System Implementation

The uMentor system was implemented in Python (version 3.12.8), integrating various open-source libraries and custom components.

3.1.1 Development Environment

1. **Core Libraries:** LangChain (v0.3 - newest) served as the primary framework for orchestrating document loading, text splitting (RecursiveCharacterTextSplitter), LLM interactions, vector store management (FAISS wrapper), and graph operations (LLMGraphTransformer, Neo4jGraph).

2. **LLMs & Embeddings:** The OpenAI Python library was used for interfacing with GPT models (specifically gpt-4o model for KG extraction and ReAct generation and gpt-4o-mini for query entity extraction). Hugging Face's Sentence-Transformers library provided the embedding model (sentence-transformers/all-MiniLM-L6-v2).

3. **Databases & Search:** FAISS library was used for efficient vector similarity search. A Neo4j database (version 5.26.1) served as the graph data store, with the Neo4j Python driver for connection. The APOC (Awesome Procedures On Cypher) library plugin was installed in Neo4j to enableJaccard similarity calculation and node merging.

4. **LLM Access (Alternative):** Ollama (version 0.6.6) was installed and configured locally to host and serve LLM, specifically a 70 billion parameter model used for the SelfRAG to train

and compare.

5. **Evaluation:** The RAGChecker library (v0.1.9) was employed for evaluating the RAGs pipeline performance.
6. **User Interface:** Streamlit (v1.43.2) was used to develop the user interface for interaction and demonstration.
7. **Hardware:**
    - Processor: Intel Core i7 processor or equivalent
    - RAM: 64 GB or more
    - Hard Drive: 29.1 GB free space

3.1.2 Data Processing Pipeline Implementation

- **Loading & Chunking:** The data loading mechanism utilized LangChain's DirectoryLoader to handle various input formats (PDF, DOCX, TXT, MD) from the specified knowledge directory (KNOWLEDGE_DIR). Text segmentation was implemented using RecursiveCharacterTextSplitter.from_tiktoken_encoder configured precisely with chunk_size = 2056 and chunk_overlap = 185.
- **Vector Store Creation:** The FAISS vector store was created by embedding the text chunks using the all-MiniLM-L6-v2 model. A faiss.IndexFlatL2 index was initialized, and embeddings were added along with document metadata. The index was persisted locally using FAISS.save_local.

3.1.3 Knowledge Graph Implementation

- **Neo4j Setup:** A Neo4j database (v5.26.1) was configured and using the APOC plugin.
- **LLM-based Extraction:** The LLMGraphTransformer component was configured with the GPT-4o model (low temperature so the model can sticks to the prompt and not try to be too creative) and the detailed system prompt described in Section 3.3.3. This component processed the text chunks to generate graph documents (entities and relationships).
- **Graph Population**: The generated graph documents were ingested into the Neo4j database using the Neo4jGraph.add_graph_documents method.
- **Jaccard Node Merging:** The Python code was executed to connect to the Neo4j instance, queried node pairs based on labels, calculated neighborhood Jaccard similarity with APOC functions, and iteratively merged nodes exceeding the SIMILARITY_THRESHOLD using apoc.refactor.mergeNodes.

3.1.4 Retrieval Module Implementation

- **Query Entity Extraction:** The OpenAIChatInterface custom class was implemented as a wrapper around the OpenAI API to specifically handle entity extraction from user queries using the GPT-4o-mini model and Pydantic schemas for structured output.
- **Unstructured Retriever:** The unstructured_retriever function was implemented to query the loaded FAISS index. It performed a similarity search and applied the custom filtering logic (high threshold with top-k selection based on the context length) to return formatted context strings.
- **Structured Retriever:** The structured_retriever function to call the query entity extractor. For each entity, it utilized the generate_full_text_query helper function to create fuzzy search strings and queried the Neo4j full-text index (db.index.fulltext.queryNodes). Afterward, it executed Cypher queries to fetch and format the neighborhood graph structure as text.

3.1.5 ReAct Agent Implementation

- **Agent Initialization:** The custom AugmentedLLM class was implemented to encapsulate the ReAct logic. An instance was created using the GPT-4o-mini model (temperature=0.85) and the use_react=True flag, injecting the combined system prompt (including the ReAct instructions).
- **Tool Integration:** The core hybrid retrieval logic (conceptually combining unstructured and structured retrieval results) was implemented as a callable function. This function was then registered as a tool (named query_FAISS_VS_and_Neo4j_KG) with the AugmentedLLM instance using its add_tool method, providing the necessary description and input schema.
- **ReAct Loop Execution:** The generate and process_stream methods within AugmentedLLM handled the interaction cycle: receiving user input, invoking the LLM, parsing the LLM's output for reasoning steps (¡thinking¿) and tool calls, executing the registered retrieval tool via execute_tool when requested, formatting the tool's output, adding it back to the message history, and allowing the LLM to continue reasoning until a final answer was generated.

3.1.6 User Interface Implementation

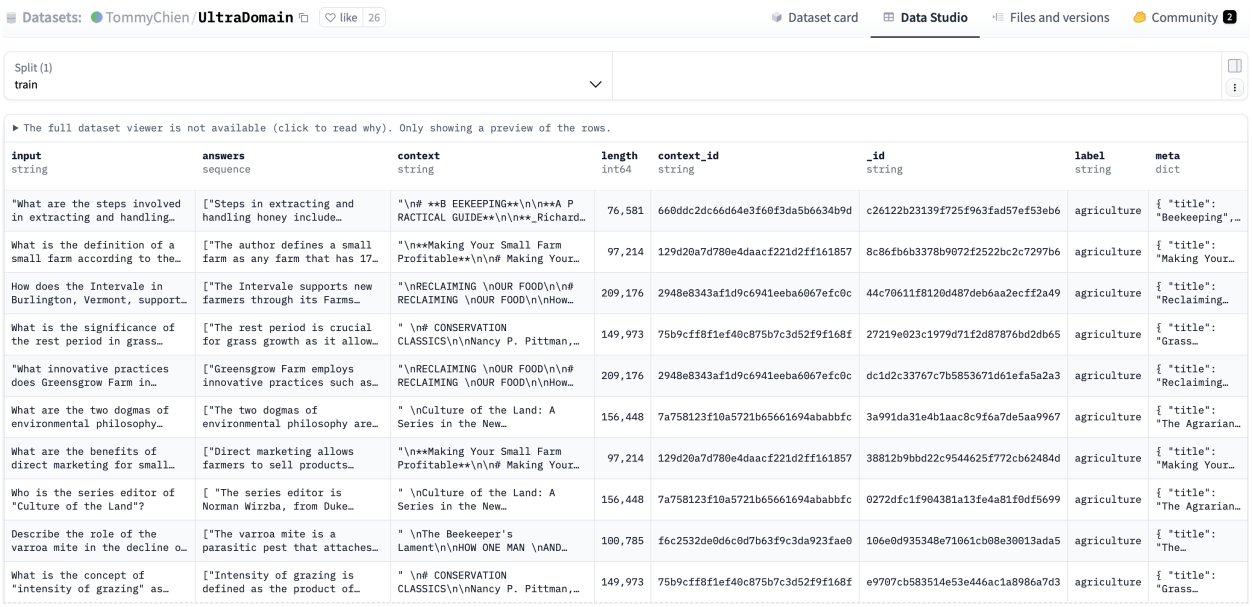
The Streamlit application provided an interface for:

- User authentication/access (implicitly via environment variables for APIs).
- Selecting the LLM model and adjusting the temperature.
- Uploading new documents to update the knowledge base.
- Triggering the offline data processing pipeline (chunking, embedding, KG creation/merging) via a button click.
- Loading existing FAISS/Neo4j assets if available.

- Engage in a conversational chat with the uMentor backend, displaying the dialogue history and the final generated responses.

3.2 Evaluation Dataset

The evaluation leverages a subset of the publicly available English-language dataset ”Tommy-Chien/UltraDomain” [11] hosted on the Hugging Face Dataset Hub. This dataset is taken from recent news from China and then trained by the GPT-4 model and generated standard question-answer pairs. The advantage of this dataset is that it’s new knowledge to the model and contains various domain-specific files and we can consider each field as a major in university. For this evaluation, four main fields were selected to become a standard for comparison, focus mostly on academic and technical knowledge.


Datasets: TommyChien / UltraDomain like 26 Dataset card Data Studio Files and versions Community 2

Split (1)
train

▸ The full dataset viewer is not available (click to read why). Only showing a preview of the rows.

input string	answers sequence	context string	length int64	context_id string	_id string	label string	meta dict
"What are the steps involved in extracting and handling…	["Steps in extracting and handling honey include…	"\n# **B EEKEEPING**\n\n**A P RACTICAL GUIDE**\n\n**_Richard…	76,581	660ddc2dc66d64e3f60f3da5b6634b9d	c26122b23139f725f963fad57ef53eb6	agriculture	{ "title": "Beekeeping",…
What is the definition of a small farm according to the…	["The author defines a small farm as any farm that has 17…	"\n**Making Your Small Farm Profitable**\n\n# Making Your…	97,214	129d20a7d780e4daacf221d2ff161857	8c86fb6b3378b9072f2522bc2c7297b6	agriculture	{ "title": "Making Your…
How does the Intervale in Burlington, Vermont, support…	["The Intervale supports new farmers through its Farms…	"\nRECLAIMING \nOUR FOOD\n\n# RECLAIMING \nOUR FOOD\n\nHow…	209,176	2948e8343af1d9c6941eeba6067efc0c	44c70611f8120d487deb6aa2ecff2a49	agriculture	{ "title": "Reclaiming…
What is the significance of the rest period in grass…	["The rest period is crucial for grass growth as it allow…	" \n# CONSERVATION CLASSICS\n\nNancy P. Pittman,…	149,973	75b9cff8f1ef40c875b7c3d52f9f168f	27219e023c1979d71f2d87876bd2db65	agriculture	{ "title": "Grass…
"What innovative practices does Greensgrow Farm in…	["Greensgrow Farm employs innovative practices such as…	"\nRECLAIMING \nOUR FOOD\n\n# RECLAIMING \nOUR FOOD\n\nHow…	209,176	2948e8343af1d9c6941eeba6067efc0c	dc1d2c33767c7b5853671d61efa5a2a3	agriculture	{ "title": "Reclaiming…
What are the two dogmas of environmental philosophy…	["The two dogmas of environmental philosophy are…	" \nCulture of the Land: A Series in the New…	156,448	7a758123f10a5721b65661694ababbfc	3a991da31e4b1aac8c9f6a7de5aa9967	agriculture	{ "title": "The Agrarian…
What are the benefits of direct marketing for small…	["Direct marketing allows farmers to sell products…	"\n**Making Your Small Farm Profitable**\n\n# Making Your…	97,214	129d20a7d780e4daacf221d2ff161857	38812b9bbd22c9544625f772cb62484d	agriculture	{ "title": "Making Your…
Who is the series editor of "Culture of the Land"?	["The series editor is Norman Wirzba, from Duke…	" \nCulture of the Land: A Series in the New…	156,448	7a758123f10a5721b65661694ababbfc	0272dfc1f904381a13fe4a81f0df5699	agriculture	{ "title": "The Agrarian…
Describe the role of the varroa mite in the decline o…	["The varroa mite is a parasitic pest that attaches…	" \nThe Beekeeper's Lament\n\nHOW ONE MAN \nAND…	100,785	f6c2532de0d6c0d7b63f9c3da923fae0	106e0d935348e71061cb08e30013ada5	agriculture	{ "title": "The…
What is the concept of "intensity of grazing" as…	["Intensity of grazing is defined as the product of…	" \n# CONSERVATION CLASSICS\n\nNancy P. Pittman,…	149,973	75b9cff8f1ef40c875b7c3d52f9f168f	e9707cb583514e53e446ac1a8986a7d3	agriculture	{ "title": "Grass…

Figure 3.1: Dataset Preview

1. **Mathematics:**
    - Description: This domain focuses on mathematical problems, about integration, logical rules and paradoxes and their applications in areas such as rating systems, business models
    - Number of question-ground truth answers: 160 pairs
    - Number of knowledge sources: 20 news or papers
    - Storage: 113 MB
2. **CS (Computer Science):**
    - Description: This domain focuses on computer science and encompasses key areas of data science and software engineering. It particularly highlights machine learning and big data processing, featuring content on recommendation systems, classification algorithms, and real-time analytics using Spark.

- Number of question-ground truth answers: 100 pairs
- Number of knowledge sources: 10 news or papers
- Storage: 95.3 MB

3. **Technology:**
   - Description: This domain centers on a lot of general technical knowledge such as HTML programming, basic to advanced programming or routing in EAGLE.
   - Number of question-ground truth answers: 240 pairs
   - Number of knowledge sources: 28 news or papers
   - Storage: 160 MB
4. **Mix:**
   - Description: This domain presents a rich variety of literary, biographical, and philosophical texts, spanning a broad spectrum of disciplines, including cultural, historical, and philosophical studies.
   - Number of question-ground truth answers: 130 pairs
   - Number of knowledge sources: 61 news or papers
   - Storage: 5.74 MB

Each file within this dataset provides structured evaluation instances containing:

- **Questions:** A set of user queries representative of the domain.
- **Reference Answers:** Ground truth or ideal answers corresponding to the questions.
- **Context Passages:** Relevant text passages intended to serve as the knowledge base (document corpus) from which RAG systems should retrieve information to answer questions. Thus, although Mix has the lowest capacity, it consumes the most resources to run the test because the large number of knowledge data to learn for the RAG system.

Overall, this structure provides the necessary input for running the RAG systems (questions and context passages as the knowledge base) and the ground truth required for calculating various evaluation metrics (reference answers). The evaluation will therefore be conducted using this English dataset to leverage its comprehensive structure while acknowledging the system's potential application in other language contexts like Vietnamese.

3.3 Evaluation Metrics

The quantitative evaluation of the proposed uMentor system and the baseline systems relies on the RAGChecker framework [12], a tool designed for assessing various aspects of Retrieval-Augmented Generation (RAG) pipelines. RAGChecker evaluates system performance by breaking down the generated answers and retrieved contexts into claims and assessing them against the ground

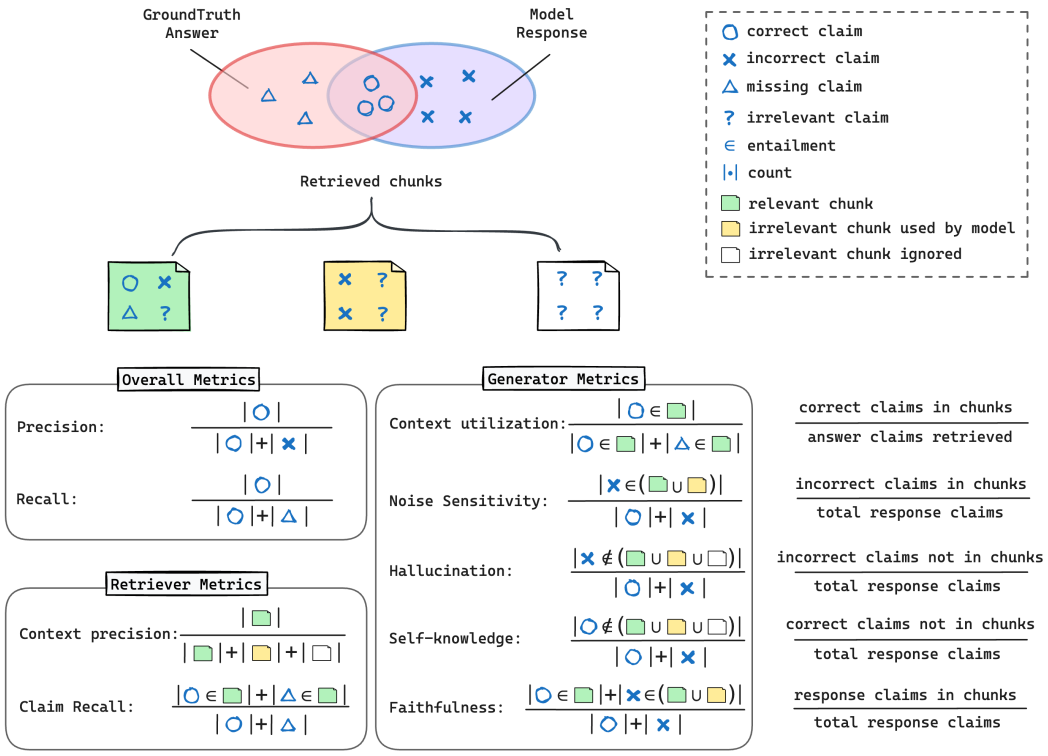


Figure 3.2: Metrics Calculation

truth and the source material. These metrics are categorized to evaluate the overall performance of the system as well as the individual performance of the two main components: the retriever and the generator. Here is a detailed explanation of what each metric means:

1. **Overall Metrics:** These metrics reflect the end-to-end performance of the entire RAG system in answering a question based on a user query.
   - **Precision:** Measures the proportion of correct, relevant, and authoritative information generated in the system's final answer. A high Precision value indicates that the system is less likely to generate false or irrelevant information.
   - **Recall:** Measures the completeness of the generated answer, i.e. the proportion of necessary and important information (based on the actual information required to answer the question) that the system has covered. A high Recall value indicates that the system provides a comprehensive answer.
   - **F1:** Is the harmonic mean of Precision and Recall. The F1 measure provides a balanced overview of overall performance, which is particularly useful when assessing both the accuracy and completeness of an answer.
2. **Retriever Metrics:** These metrics focus on assessing the effectiveness of the retrieval component in finding relevant text or information from the data source (including the Knowledge Graph) to provide context to the generator.
   - **Claim Recall:** Measures the ability of the retriever to find context that supports the claims needed to answer the user's question. The proportion of important claims that are

covered by the retrieved context.

- **Context Precision:** Measures the proportion of context fragments returned by the retriever that are actually relevant and useful for answering the question. A high Context Precision indicates that the retriever is effective in filtering out noise and only retrieving valuable information.

3. **Generator Metrics:** These metrics evaluate the performance of the text generator, focusing on its ability to use the context provided and generate accurate, truthful answers.

   - **Context Utilization:** Measures the extent to which the generative model actually uses the context information provided by the retriever to generate the answer. A high value indicates that the model makes good use of the input context.
   - **Noise Sensitivity in Relevant:** Measures the extent to which the generative model's performance is affected when the retrieved context includes both relevant and noisy information. A high value indicates that the model is susceptible to 'misdirection' or performance degradation when exposed to noise in useful context.
   - **Noise Sensitivity in Irrelevant:** Measures the extent to which the generative model is affected when the context provided is completely irrelevant to the question. A low value indicates that the model is successful in recognizing and ignoring irrelevant information, minimizing the risk of generating misleading content.
   - **Hallucination:** Measures the proportion of information in the generated answer that is not directly based on the context provided. This is an important measure of how much the model 'fabricates' information. A low Hallucination value is an important goal.
   - **Self-Knowledge:** Measures the extent to which the generative model relies on its own internal knowledge rather than prioritizing the context provided by the retriever. In RAG architectures, it is often desirable for the model to rely heavily on context to ensure the up-to-dateness and transparency of the information.
   - **Faithfulness:** Measures the extent to which the generated answer is faithful to the information contained in the context provided by the retriever. This measure assesses the informational consistency between the context source and the generated answer.

To evaluate a comprehensive RAG with the above measurements, the following are the higher the better: Precision, Recall, F1, Claim Recall, Context Precision, Context Utilization, and Faithfulness. The remaining measurements such as: Noise Sensitivity in Relevant, Hallucination,.. the lower the better for the RAG system.

In general, these metrics provide a comprehensive view of the performance of the RAG system, including the efficiency of context retrieval, the ability of the generator to generate correct answers, the degree of reliance on the Knowledge Base versus pre-learned knowledge, and the ability to handle noise.

3.4 Experimental Results

Here I have a small dataset taken from VNU University of Engineering and Technology. This dataset includes information related to subjects, learning materials, course materials, list of lecturers and majors in the school. Below is a Q&A session with the chatbot for these test scenarios:

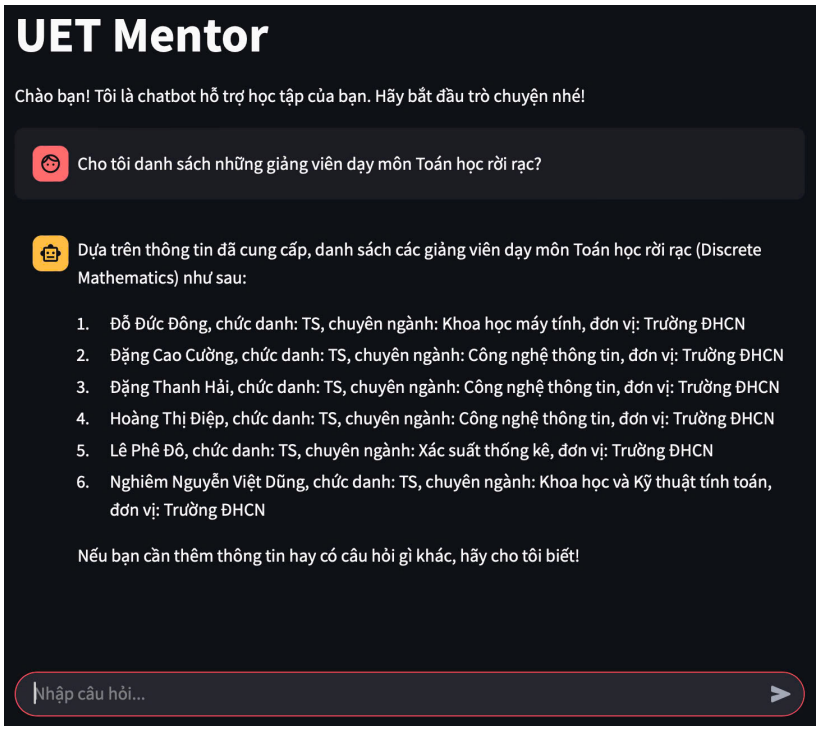


Figure 3.3: Questions and answers about the Lectures

1. Retrieve information about the list of lecturers at UET university (Figure 3.3) along with its ReAct mechanism (Figure 3.4). uMentor will first use the unstructured data query engine from FAISS Vector Store, then it will execute structured queries from Neo4j Database. When using ReAct, it will analyze the user's question (translation, keyword search) and then check the retrieved information from contextual sources to make inferences. And by default, the chatbot's reasoning (<thinking>... </thinking>) phrase will be written in English.

2. uMentor provides learning materials to support new students (Figure 3.5). Using Neo4j's FULLTEXT INDEX, uMentor can perform approximate retrieval on Knowledge Graph data. As in the example, "Programming" can be understood as "Introduction to Programming" or "Advanced Programming". uMentor will access both of those nodes, and finally combine with the context from the user's query (new students) to select documents for the subject "Introduction to Programming".

3. The ability to summarize the learning text book by section of uMentor (Figure 3.6). Based on pre-set prompts, uMentor can summarize the main knowledge areas, illustrative examples and exercises contained in each book chapter.

4. uMentor provides practice exercises to help students practice directly from their lecture' textbooks (Figure 3.8)

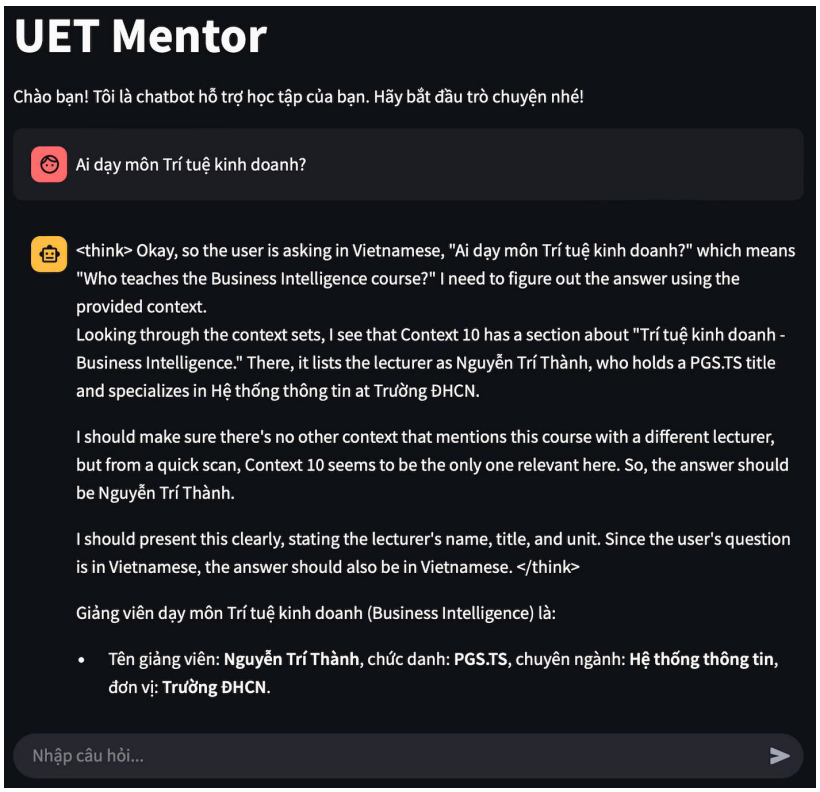


Figure 3.4: uMentor's reasoning mechanism

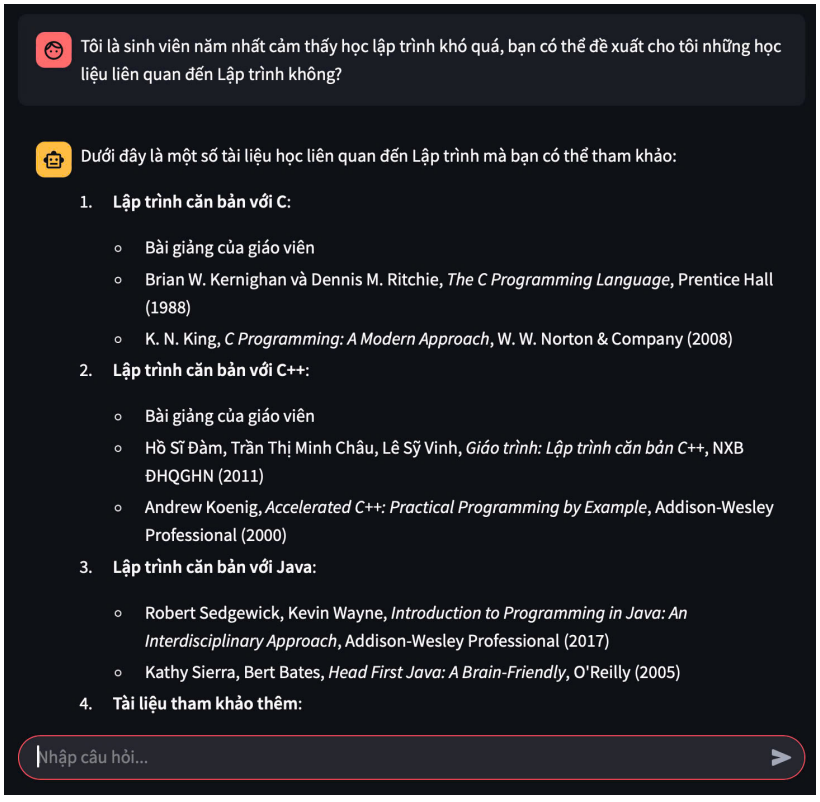


Figure 3.5: Questions and answers about learning materials

To more comprehensively evaluate the capabilities of uMentor, we will need to use a larger dataset with specific metrics as mentioned in the previous section. Here is how to implement the uMentor evaluation flow and systems

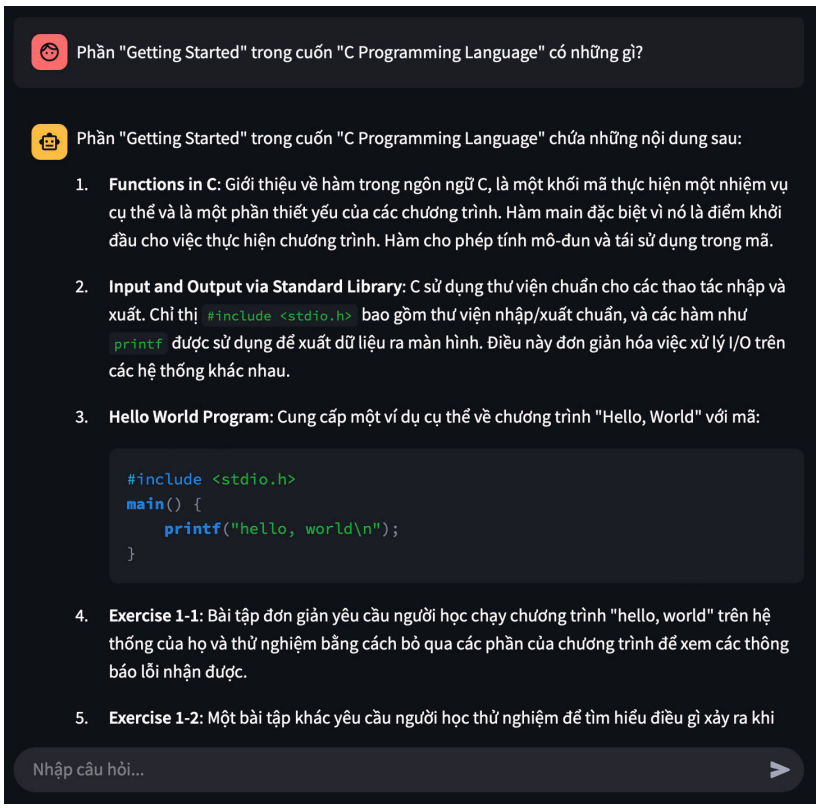


Figure 3.6: uMentor summarizes and synthesizes information from book chapters

```
Entity: Getting Started
Nodes:
ID: A Tutorial Introduction, Properties: {'chapter_number': '1', 'title': 'A Tutorial Introduction'}
ID: Getting Started, Properties: {'title': 'Getting Started', 'section_number': '1.1'}
ID: Functions in C, Properties: {'description': 'A function in C is a block of code that performs a specific task and is a fundamental building block of programs.
The main function is special because it serves as the entry point where program execution begins. Functions enable modularity and reusability in code.'}
ID: Input and Output via Standard Library, Properties: {'description': 'C uses the standard library for input and output operations. The #include <stdio.h> directive
includes the standard input/output library, and functions like printf are used to output data to the console. This abstraction simplifies I/O handling across
different systems.'}
ID: Hello World Program, Properties: {'description': '#include <stdio.h> main() {    printf("hello, world\n");}'}
ID: Exercise 1-1, Properties: {'difficulty': 'Easy', 'description': "Run the ``hello, world'' program on your system. Experiment with leaving out parts of
the program, to see what error messages you get."}
ID: Exercise 1-2, Properties: {'difficulty': 'Easy', 'description': "Experiment to find out what happens when prints's argument string contains \\c, where c is some
character not listed above. "}

Relationships:
A Tutorial Introduction - HAS_SECTION -> Getting Started
Getting Started - MENTIONS_CONCEPT -> Functions in C
Getting Started - MENTIONS_CONCEPT -> Input and Output via Standard Library
Getting Started - PROVIDES_EXAMPLE -> Hello World Program
Getting Started - HAS_EXERCISE -> Exercise 1-1
Getting Started - HAS_EXERCISE -> Exercise 1-2
Entity: C Programming Language
Nodes:
ID: C Programming Language, Properties: {'year': '', 'author': 'Brian W. Kernighan, Dennis M. Ritchie', 'edition': '2nd', 'publisher': '', 'title': 'C Programming
Language'}
ID: A Tutorial Introduction, Properties: {'chapter_number': '1', 'title': 'A Tutorial Introduction'}
ID: Types, Operators And Expressions, Properties: {'chapter_number': '2', 'title': 'Types, Operators and Expressions'}
ID: Control Flow, Properties: {'chapter_number': '3', 'title': 'Control Flow'}

Relationships:
C Programming Language - CONTAINS_CHAPTER -> A Tutorial Introduction
C Programming Language - CONTAINS_CHAPTER -> Types, Operators And Expressions
C Programming Language - CONTAINS_CHAPTER -> Control Flow
```

Figure 3.7: uMentor retrieve structured data from the Knowledge Graph to answer questions

1. Data Preprocessing and Embedding

   - Get the ”content” field from the ”meta” column from the dataset. This will be the main knowledge base for uMentor. (Figure 3.9)
   - Chunk the documents into smaller text chunks (Review section 2.2.2 for hyperparameter selection).

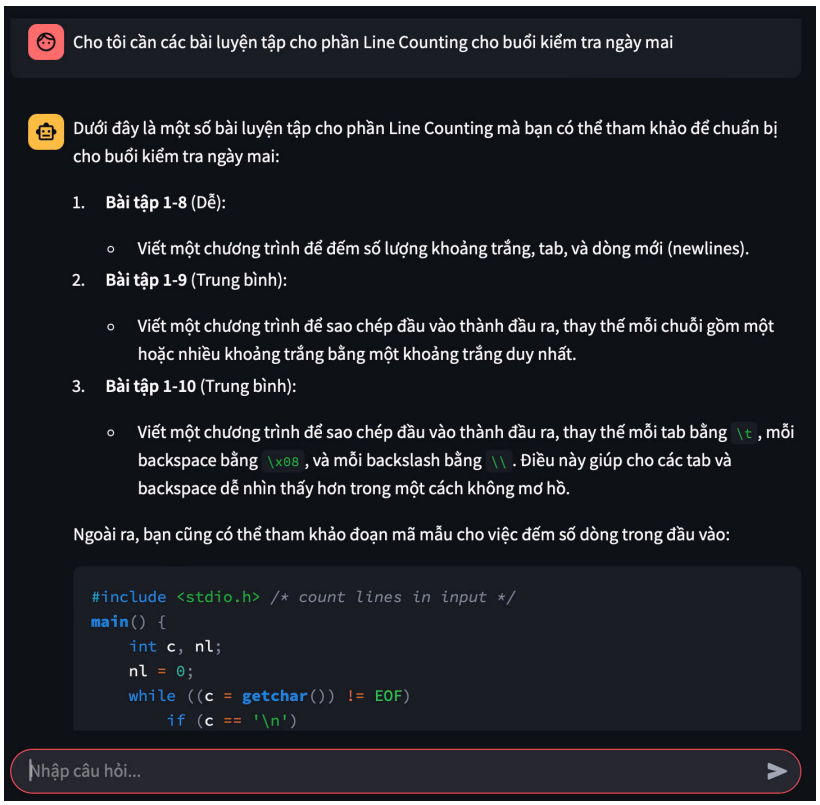


Figure 3.8: uMentor provides practice exercises

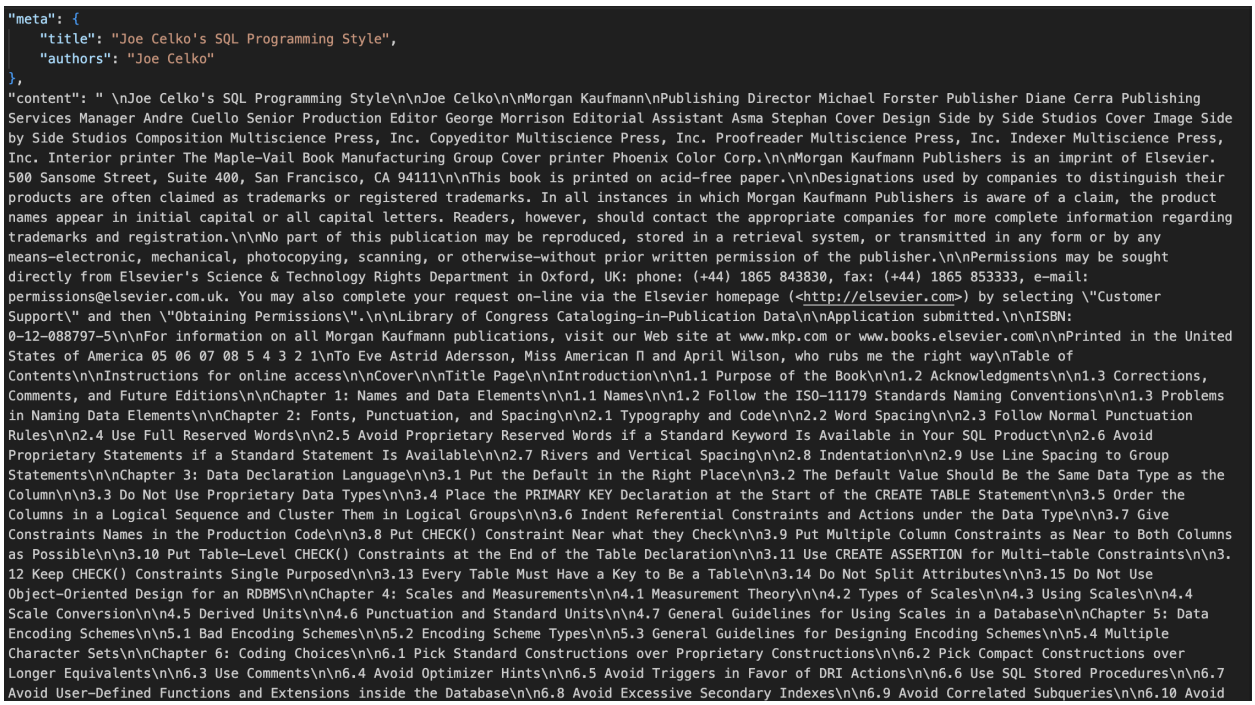


Figure 3.9: Content from dataset

- Use the selected embedding model to generate embeddings for each text chunk.
- Store these vectors along with the corresponding original text into the FAISS Vector Database. This storage vector will be used for all compared systems.

2. Building Knowledge Graph

```
merge_output.txt
2025-04-13 23:42:39,751 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:39,752 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:39,753 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:39,754 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:39,754 - INFO - Received notification from DBMS server: {severity: INFORMATION} {code: Neo.ClientNotification.Statement.CartesianProduct} {category: PERFORMANCE} {title:
2025-04-13 23:42:39,755 - INFO -  -> Tìm thấy 5 cặp ứng viên (vượt ngưỡng 0.56) cho Label 'Organization'.
2025-04-13 23:42:39,755 - INFO - Đang tìm ứng viên hợp nhất cho Label: Concept
2025-04-13 23:42:48,776 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:48,777 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:48,777 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:48,777 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:48,777 - INFO - Received notification from DBMS server: {severity: INFORMATION} {code: Neo.ClientNotification.Statement.CartesianProduct} {category: PERFORMANCE} {title:
2025-04-13 23:42:48,778 - INFO -  -> Tìm thấy 28 cặp ứng viên (vượt ngưỡng 0.56) cho Label 'Concept'.
2025-04-13 23:42:48,779 - INFO - Đang tìm ứng viên hợp nhất cho Label: Product
2025-04-13 23:42:49,520 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:49,522 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:49,522 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:49,523 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:49,524 - INFO - Received notification from DBMS server: {severity: INFORMATION} {code: Neo.ClientNotification.Statement.CartesianProduct} {category: PERFORMANCE} {title:
2025-04-13 23:42:49,525 - INFO -  -> Tìm thấy 5 cặp ứng viên (vượt ngưỡng 0.56) cho Label 'Product'.
2025-04-13 23:42:49,526 - INFO - Đang tìm ứng viên hợp nhất cho Label: Location
2025-04-13 23:42:49,682 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:49,682 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:49,683 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:49,683 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:49,683 - INFO - Received notification from DBMS server: {severity: INFORMATION} {code: Neo.ClientNotification.Statement.CartesianProduct} {category: PERFORMANCE} {title:
2025-04-13 23:42:49,684 - INFO -  -> Tìm thấy 1 cặp ứng viên (vượt ngưỡng 0.56) cho Label 'Location'.
2025-04-13 23:42:49,684 - INFO - Đang tìm ứng viên hợp nhất cho Label: Person
2025-04-13 23:42:50,308 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:50,309 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:50,310 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:50,310 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:50,311 - INFO - Received notification from DBMS server: {severity: INFORMATION} {code: Neo.ClientNotification.Statement.CartesianProduct} {category: PERFORMANCE} {title:
2025-04-13 23:42:50,311 - INFO -  -> Tìm thấy 6 cặp ứng viên (vượt ngưỡng 0.56) cho Label 'Person'.
2025-04-13 23:42:50,312 - INFO - Đang tìm ứng viên hợp nhất cho Label: Version
2025-04-13 23:42:50,314 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:50,314 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:50,315 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:50,315 - WARNING - Received notification from DBMS server: {severity: WARNING} {code: Neo.ClientNotification.Statement.FeatureDeprecationWarning} {category: DEPRECATION}
2025-04-13 23:42:50,315 - INFO - Received notification from DBMS server: {severity: INFORMATION} {code: Neo.ClientNotification.Statement.CartesianProduct} {category: PERFORMANCE} {title:
```

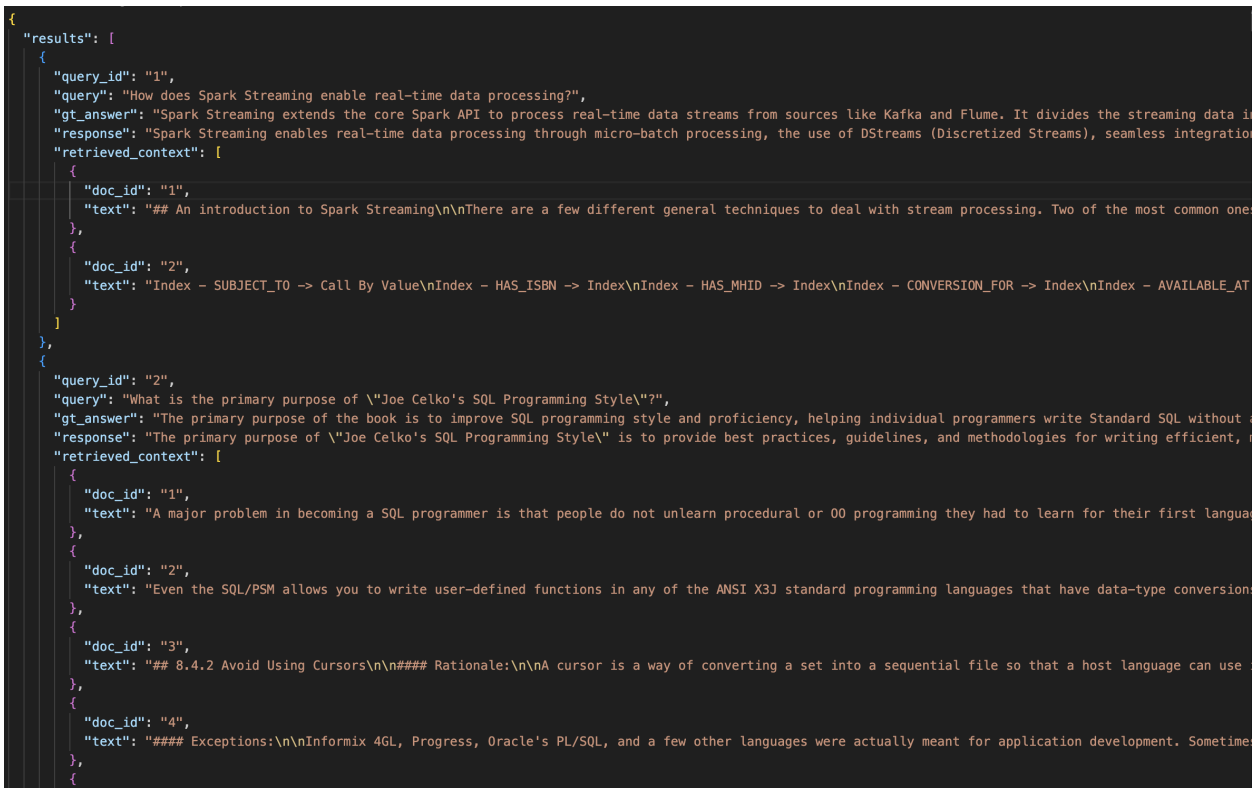
Figure 3.10: Merge Entities

- Use LangChain's entity-relationship extraction function with the built-in prompt (Review section 2.2.3) to initialize the initial KGs.
- Run the Jaccard Index algorithm to find and remove duplicate entities. The refined KGs are ready to be used for query execution. (Figure 3.10)

3. Get answers from chatbot systems
   - Using the prepared Evaluation Dataset.
   - Feed each question, from the "input" column of the Evaluation Dataset, into: NaiveRAG, SelfRAG, and uMentor.
   - Record the results: For each question, save the answer generated by NaiveRAG, SelfRAG, and the original text segments retrieved by the them respectively.
   - For uMentor, we will skip the chatbot's reasoning process and take the Final Answer as the response.
4. Collect Evaluation Data
   - Store all execution results in order: query id, query, ground truth answer, response, and retrieved context).

3.5 Evaluation

To evaluate the performance and efficiency of the proposed KG-enhanced ReAct RAG system, we have developed a detailed evaluation plan. This plan includes comparing the proposed system with baseline systems and another state-of-the-art RAG framework, using a standardized dataset



```
{
  "results": [
    {
      "query_id": "1",
      "query": "How does Spark Streaming enable real-time data processing?",
      "gt_answer": "Spark Streaming extends the core Spark API to process real-time data streams from sources like Kafka and Flume. It divides the streaming data i
      "response": "Spark Streaming enables real-time data processing through micro-batch processing, the use of DStreams (Discretized Streams), seamless integratio
      "retrieved_context": [
        {
          "doc_id": "1",
          "text": "## An introduction to Spark Streaming\n\nThere are a few different general techniques to deal with stream processing. Two of the most common one
        },
        {
          "doc_id": "2",
          "text": "Index - SUBJECT_TO -> Call By Value\nIndex - HAS_ISBN -> Index\nIndex - HAS_MHID -> Index\nIndex - CONVERSION_FOR -> Index\nIndex - AVAILABLE_AT
        }
      ]
    },
    {
      "query_id": "2",
      "query": "What is the primary purpose of \"Joe Celko's SQL Programming Style\"?",
      "gt_answer": "The primary purpose of the book is to improve SQL programming style and proficiency, helping individual programmers write Standard SQL without
      "response": "The primary purpose of \"Joe Celko's SQL Programming Style\" is to provide best practices, guidelines, and methodologies for writing efficient,
      "retrieved_context": [
        {
          "doc_id": "1",
          "text": "A major problem in becoming a SQL programmer is that people do not unlearn procedural or OO programming they had to learn for their first langua
        },
        {
          "doc_id": "2",
          "text": "Even the SQL/PSM allows you to write user-defined functions in any of the ANSI X3J standard programming languages that have data-type conversion
        },
        {
          "doc_id": "3",
          "text": "## 8.4.2 Avoid Using Cursors\n\n#### Rationale:\n\nA cursor is a way of converting a set into a sequential file so that a host language can use
        },
        {
          "doc_id": "4",
          "text": "#### Exceptions:\n\nInformix 4GL, Progress, Oracle's PL/SQL, and a few other languages were actually meant for application development. Sometime
        },
        {
```


Figure 3.11: uMentor answer set

and metrics provided by the Ragas evaluation framework. The main objectives of this experimental part are:

1. To demonstrate the ability of the RAG system to answer questions in the selected knowledge domain from the dataset.
2. Evaluate the effectiveness of the RAG Chatbot in terms of: accuracy, relevance of answers, and ability to use context from the original document.
3. Compare the effectiveness of the RAG Chatbot with other system.

3.5.1 Baseline and Comparison Systems

We will compare our system with two other systems:

1. **Naive RAG (Baseline):** This is the main baseline system for comparison. This baseline uses the same set of documents (from the evaluation dataset), the same embedding model (sentence-transformers/all-MiniLM-L6-v2), the same FAISS vector repository, and the same base LLM for generating answers (e.g., GPT-4o-mini). Importantly, however, it operates without integrating context from the Neo4j Knowledge Graph and without using the ReAct framework for reasoning and tooling. Its generator only takes as input the original query and text fragments retrieved directly from the FAISS vector store. This direct comparison is intended to highlight the specific benefits of integrating KG and the sequential Agent ReAct approach.

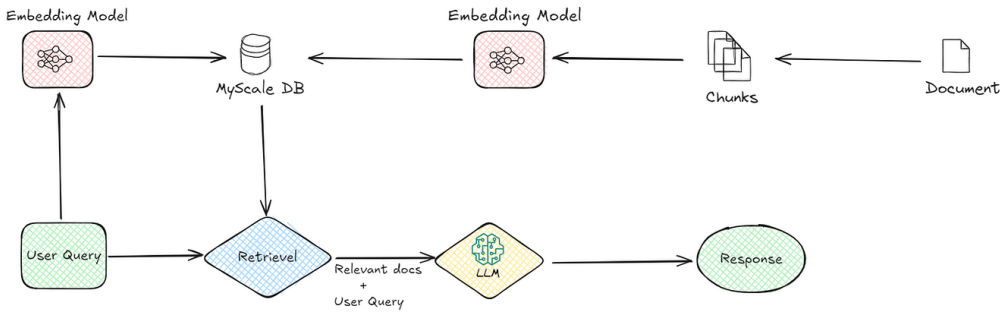


Figure 3.12: NaiveRAG flow

2. **SelfRAG (Comparison Framework):** To position the performance of the recommender system in the context of state-of-the-art RAG techniques, we will also compare it with SelfRAG [13]. SelfRAG is an advanced RAG framework in which the LLM self-evaluates (reflects) the quality of both the retrieved information and the answers it generates before delivering the final result. It uses special "reflection tokens" to control the retrieval and generation process, allowing it to self-correct, verify authenticity, or re-retrieve information if necessary without retraining the LLM.

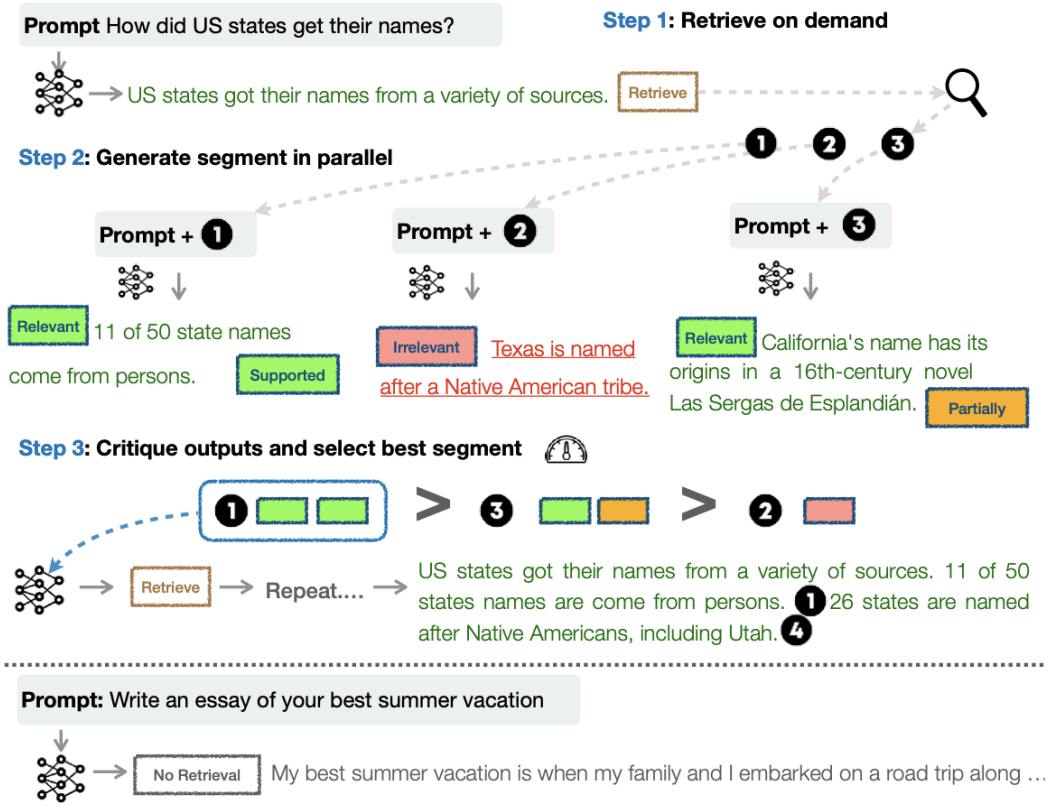


Figure 3.13: Overview of SelfRAG

In this evaluation plan, SelfRAG will be deployed using the same FAISS dataset and vector repository setup (with all-MiniLM-L6-v2 embedding) as the other systems. However, Self-RAG will use a different open-source large language model, specifically a 70 billion parameter

model accessed via Ollama to train .

3.5.2 NaiveRAG vs uMentor

This section presents an overview of the performance comparison of the proposed uMentor system (KG-ReAct-RAG) with the Naive RAG Baseline system. The evaluation uses the RAGChecker framework on a selected subset of the "TommyChien/UltraDomain" dataset, which covers a variety of academic domains. The aggregated results across all metrics are summarized in Table 3.1.

Table 3.1: Comparison of Overall Average RAGChecker Scores between uMentor and Naive RAG across all domains

Metrics	Naive RAG	uMentor
Overall		
Precision	62.8%	73.2%
Recall	63.9%	69.1%
F1	60.2%	71.2%
Retriever Metrics		
Claim Recall	67.8%	72.9%
Context Precision	62.4%	69.6%
Generator Metrics		
Context Utilization	62.7%	70.7%
Noise Sensitivity in Relevant	21.2%	18.2%
Noise Sensitivity in Irrelevant	14.5%	10.4%
Hallucination	21.8%	15.6%
Self-Knowledge	16.1%	8.7%
Faithfulness	60.1%	74.2%

As we can see from Table 3.1, the proposed uMentor system had outperforms the baseline Naive RAG across all evaluated dimensions.

For the Overall Metrics, uMentor shows a substantial improvement, with an F1 score of 71.2% compared to Naive RAG's 60.2%. This shows a better balance between Precision (73.2% vs. 62.8%) and Recall (69.1% vs. 63.9%), indicating that uMentor's answers are not only more accurate (higher precision) but also more relevant (higher recall) to the ground truth claims.

For Retriever Metrics, uMentor's combined retrieval strategy shows clear improvement. Context Precision increased from 67.8% for Naive RAG to 72.9% for uMentor, suggesting that the retrieved context in uMentor is more valuable and relevant. Claim Recall also improved from 67.8% to 72.9%, indicating that uMentor's retrieval is more effective in finding supporting context to answer queries

The most pronounced improvements were observed in Generator Metrics, highlighting the benefits of the KGs and ReAct integration on answer quality. Faithfulness, a measure of consistency with the retrieved context, saw a significant jump from 60.10% to 74.20%. Hallucination dropped sharply from 21.8% to 15.6%, and reliance on Self-Knowledge dropped from 16.1% to just 8.7%. These results clearly indicate that uMentor is much better at basing its responses on the information provided and is less likely to generate unsupported or internally derived facts. Context Utilization

also improved significantly (70.7% vs. 62.7%), indicating that the generator is making better use of the relevant retrieved context. Furthermore, uMentor demonstrated superior noise immunity. Sensitivity to noise in relevant contexts was reduced (21.20% to 18.20%), and importantly, Sensitivity to noise in irrelevant contexts was sharply reduced from 14.5% to 10.4%. This suggests that uMentor is significantly better at ignoring distracting or irrelevant information during generation, likely due to the structured guidance from the KG and the filtering capabilities of the ReAct agent.

Overall, the RAGChecker results compared to the Naive RAG baseline provide compelling quantitative evidence that integrating the Knowledge Graph and the ReAct agent significantly improves the performance of the RAG system across all key dimensions, from relevance and retrieval accuracy to truthfulness, accuracy, and robustness of answers.

3.5.3 SelfRAG vs uMentor

While the comparison with Naive RAG highlights the general advantages of incorporating structured knowledge and reasoning, it is also crucial to compare uMentor against more sophisticated RAG architectures. This section presents the performance comparison between the proposed uMentor system and Self-RAG, an advanced RAG architecture incorporating a self-reflection mechanism, evaluated using the RAGChecker framework. This comparison assesses whether uMentor's approach (KG + ReAct) offers distinct advantages over a sophisticated baseline that uses the language model itself for quality improvement. The results are summarized in Table 3.2.

Table 3.2: Comparison of Overall Average RAGChecker Scores between uMentor and SelfRAG across all domains

Metrics	SelfRAG	uMentor
Overall		
Precision	59.7%	63.9%
Recall	61.5%	63.2%
F1	60.6%	63.6%
Retriever Metrics		
Claim Recall	57.7%	64.2%
Context Precision	53.4%	57.2%
Generator Metrics		
Context Utilization	55.2%	61.1%
Noise Sensitivity in Relevant	28.1%	26.6%
Noise Sensitivity in Irrelevant	10.1%	9.7%
Hallucination	24.3%	18.8%
Self-Knowledge	19.0%	13.1%
Faithfulness	56.9%	68.1%

The RAGChecker evaluation reveals that uMentor demonstrates significant advantages over the Self-RAG baseline across most metrics, indicating that the KG-ReAct integration provides benefits beyond Self-RAG's self-reflection capabilities.

In the Overall Metrics, uMentor achieved a higher F1 score (63.6%) compared to Self-RAG (60.60%), driven by notably higher Precision (63.9% vs. 59.7%). Recall was comparable (63.2%

vs. 61.5%). This suggests uMentor's final answers are generally more accurate and equally, if not slightly more, complete than those from Self-RAG.

3.6 Chapter Summary

The experimental results presented in this chapter provide convincing evidence of the effectiveness of the proposed uMentor system, which enhances Retrieval-Augmented Generation through the synergistic integration of Knowledge Graph and the Reasoning and Acting (ReAct) framework. Quantitative evaluation, conducted using the RAGChecker framework on a subset of the "TommyChien/UltraDomain" dataset, demonstrated that the proposed uMentor system consistently outperforms both the baseline Standard RAG system and the more advanced Self-RAG system across most key metrics evaluated.

Against the Naive RAG baseline, uMentor showed substantial and widespread improvements across all RAGChecker metric categories. Significant gains were observed in overall performance. The system demonstrated improved context relevance and recall, indicating a more effective retrieval mechanism capable of fetching more precise and comprehensive information. Furthermore, uMentor exhibited markedly enhanced generator performance, substantially reduced hallucination and reliance on internal self-knowledge, and better utilization of the provided context.

When compared against the Self-RAG baseline, a more sophisticated architecture incorporating self-reflection, uMentor still demonstrated competitive or superior performance in several critical areas. Not only achieving higher overall scores, its combined retrieval approach also yielded better results in both claim recall and context precision compared to SelfRAG's standard retrieval. In terms of generator performance, uMentor showed distinct advantages in metrics related to grounding and factual reliability. While both systems aimed for robustness, uMentor maintained a slight edge in handling noise. This comparison highlights that the structured knowledge provided by the KG and the explicit reasoning guided by ReAct offer valuable advantages in grounding responses and reducing factual errors, even when compared to an advanced system that uses internal self-correction mechanisms.

In summary, the empirical evidence using the RAGChecker framework strongly supports the hypothesis that leveraging Knowledge Graphs and the ReAct agent significantly enhances the capabilities of RAG systems for complex information retrieval and question answering tasks within the university context.

Conclusion and Future Works

Main Contributions

Based on the initial goals that the project has set, we have conducted research methods . This thesis explores a novel approach to enhance Retrieval-Augmented Generation (RAG) systems by integrating Knowledge Graph (KG), which is specifically designed to process information within the university context. Addressing the limitations of RAG systems that rely primarily on unstructured text retrieval, this study proposes and implements a hybrid retrieval mechanism that leverages the strengths of structured and unstructured data sources. The core contributions of this thesis are outlined as follows:

- **Building a Domain-Specific Knowledge Graph with a Refined Scheme:** We designed and implemented a domain-specific Knowledge Graph based on a dataset from a university dataset. Our KGs can effectively model important entities (e.g., courses, departments, programs, events, students) and their relationships related to university information.
- **Knowledge graph denoising and entity consistency:** To ensure data quality in the knowledge graph, we used a method to improve the quality and compactness of the graph, in terms of entity consistency. This process identifies and resolves inconsistencies or duplications in entity representations, improving the reliability and accuracy of the information stored in the graph.
- **Hybrid retrieval agent development:** The agent dynamically decides whether to retrieve information from a vector repository (for semantic search on unstructured, contextual text) or from a knowledge graph (for extracting structured entities and relationships). This hybrid approach uniquely combines contextual understanding from unstructured data with precise relational knowledge from structured data.

Together, these contributions advance RAG systems by providing a novel architecture that is more efficient than traditional approaches. Our work demonstrates how integrating a clean and well-structured Knowledge Graph, managed by an intelligent retrieval agent, significantly improves the ability of a RAG system to:

- **Handle complex and related queries:** By leveraging KGs, the system can accurately answer questions that require understanding relationships between entities, which is challenging for text-only RAGs.

- **Provide more accurate and realistic responses:** KG retrieval ensures access to verified structured facts, resulting in more reliable and less illusory output from the language model.
- **Improved contextual understanding:** The ability to combine contextual retrieval from the vector repository with specific facts from the agent's KG enables deeper, richer understanding of user queries and generates more relevant responses.
- **Improved information discovery:** The structured nature of the KG facilitates the presentation of relevant information, allowing users to explore relevant concepts in the university.

This thesis contributes to the fields of Knowledge Graphs, Advanced Data Generation for Retrieval, and Information Systems by providing a practical methodology and empirical evidence for building reliable question answering systems mediated by an intelligent agent.

Limitations

Despite the positive results, this study also encountered certain limitations that need to be acknowledged:

- **Scope of the Knowledge Graph:** The Knowledge Graph built in this study may not cover all the knowledge related to all academic disciplines in the university.
- **Rely on data quality:** The performance of the system is heavily reliant on the quality and completeness of the data within the Knowledge Graph and the documents used for the RAG model.
- **Dependence on LLMs:** Because we use APIs from other platforms without models trained from Vietnamese datasets, we will also have difficulty processing these types of data. Depending on LLM model providers is sometimes very unstable and uncertain.
- **Challenges in handling highly complex or ambiguous queries:** Despite improvements, chatbots can still have difficulty handling extremely complex or ambiguous user queries.

Future Works

We intend to integrate the uMentor chatbot with other educational technologies such as Learning Management Systems (LMS) or other learning support tools. Here are some potential avenues for future developments:

- **All-in-one approach:** Schema design can accommodate almost all knowledge domains in the university (e.g., videos, lecture notes). This will help automate the construction and re-updating of KGs.
- **Reducing the cost of building and using Knowledge Graph:** Currently, building a knowledge graph is very expensive (much more than using a vector store). We will continue to research ways to reduce the cost of building a KGs such as compressing it and only decompressing to certain parts of the graph when called.

- **Training a Vietnamese LLM model specifically for university data and building a Knowledge Graph:** Currently, the construction of a Knowledge Graph using API calls with Vietnamese university data is very unstable. To improve this, we will definitely train a multi-purpose LLM model specifically for Vietnamese university texts.

In conclusion, this thesis has demonstrated the significant potential of combining Knowledge Graph and RAG to build a chatbot that supports more effective education for university students. The findings and contributions of this study will serve as research material and support for future studies in developing intelligent learning support systems. Hopefully, these results will inspire further research in this area.

Bibliography

[1] L. T. K. Nguyen, L. D. Pham, and H. N. Nguyen, "umentor: Llm-powered chatbot for harnessing technology books in digital library," ICCCI, 2024.

[2] K. Taneja, P. Maiti, S. Kakar, P. Guruprasad, S. Rao, and A. K. Goel, "Jill watson: A virtual teaching assistant for online education," arXiv:2405.11070 [cs.AI], 2024.

[3] A. Coyne, "Meet genie, deakin uni's virtual assistant for students," tech. rep., iTnews, 2017.

[4] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," NeurIPS, 2020.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," NIPS, 2017.

[6] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres, R. Navigli, A.-C. N. Ngomo, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann, "Knowledge graphs," arXiv:2003.02320 [cs.AI], 2020.

[7] B. Y. Lin, W. Zhou, M. Shen, P. Zhou, C. Bhagavatula, Y. Choi, and X. Ren, "Commongen: A constrained text generation challenge for generative commonsense reasoning," EMNLP, 2020.

[8] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, "From local to global: A graph rag approach to query-focused summarization," arXiv:2404.16130 [cs.CL], 2024.

[9] Z. Guo, L. Xia, Y. Yu, T. Ao, and C. Huang, "Lightrag: Simple and fast retrieval-augmented generation," ICLR, 2024.

[10] E. Gray, "Alert merging with machine learning," tech. rep., Two Six Technologies, 2022.

[11] Y. Lyu, Z. Li, S. Niu, F. Xiong, B. Tang, W. Wang, H. Wu, H. Liu, T. Xu, and E. Chen, "Crud-rag: A comprehensive chinese benchmark for retrieval-augmented generation of large language models," arXiv:2401.17043 [cs.CL], 2024.

[12] D. Ru, L. Qiu, X. Hu, T. Zhang, P. Shi, S. Chang, C. Jiayang, C. Wang, S. Sun, H. Li, Z. Zhang, B. Wang, J. Jiang, T. He, Z. Wang, P. Liu, Y. Zhang, and Z. Zhang, "Ragchecker:

A fine-grained framework for diagnosing retrieval-augmented generation," arXiv:2408.08067 [cs.CL], 2024.

[13] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-rag: Learning to retrieve, generate, and critique through self-reflection," arXiv:2310.11511 [cs.CL], 2023.