Utilization of Large Language Models in the Educational Process

Kyrylo Masaltsev, Marek Vančo
Faculty of Informatics and Information Technologies, Slovak University of Technology, Bratislava, Slovakia
xmasaltsev@stuba.sk, marek_vanco@stuba.sk

Abstract - **This paper addresses the potential of Large Language Models (LLMs) to support students in the educational process by providing personalized assistance, explanations, and examples. LLMs are used by many students, but they often lack up-to-date information or specific knowledge on the subjects they need. This paper looks at how using Retrieval-Augmented Generation (RAG) can help by linking language models to external sources of information so answers can be more accurate and relevant. The proposed solution is implemented using the LangChain framework, which simplifies integration with vector databases, embedding models, and custom prompt workflows. The system is designed as a modular architecture with specialized agents that respond to different types of student queries. This approach demonstrates how LLMs can be enhanced to better learning experience.**

Keywords - **LLM; LangChain; RAG; Agents; Educational**

I. Introduction

Nowadays, the use of LLMs is becoming increasingly common among students at all levels of education. These models are commonly used to assist with tasks ranging from simple questions and assignments to explaining complex concepts. LLMs offer significant potential to enhance the educational process, but their limitations pose important challenges. LLMs may lack domain-specific knowledge, rely on outdated information, or even produce inaccurate responses. Such shortcomings can negatively affect the learning experience, especially when students rely on these models for studying critical or specialized topics.

This paper presents an exploration of integrating LLM with a RAG architecture implemented using the LangChain framework. Similar ideas have been explored in recent research. In [1], the authors describe a university chatbot that uses RAG to answer curriculum-related questions by combining LLMs with institution-specific data. Their system shows how retrieval can help reduce hallucinations in educational settings. Another example is presented in [2], where a tutoring chatbot is built for university students using open-source tools. It retrieves course-specific materials to support learning and emphasizes transparency in the generation process. The system described in this paper follows a similar method, having a modular agent-based design adapted to different types of student queries for a particular subject.

II. Understanding the Role of LLMs in Education

[3] LLMs are designed to understand and generate human language, making them suitable for a wide range of natural language processing tasks. However, their use in education brings with it both opportunities and limitations that need to be considered. A single question asked in slightly different ways may receive completely different responses. This inconsistency is caused by several factors: LLMs rely on probabilistic generation, they lack an understanding of factual correctness, and they are limited to the static data available at the time of training. As a result, their answers can be out of date or even misleading, especially when it comes to subject-specific or curriculum-relevant content.

One approach to improving LLM results is fine-tuning, in which the model is retrained on specialized data. However, fine-tuning is expensive, requires large, curated datasets, and increases the risk of overfitting or drifting away from general reasoning abilities. It is also inflexible — updating even a small portion of knowledge requires retraining the model.

Instead, a more efficient and scalable solution is retrieval augmentation generation (RAG). RAG keeps the original LLM intact but enriches it with external knowledge during inference. By extracting relevant documents from a trusted source (e.g. lecture notes or curated examples), we can guide the model to generate responses based on real, context-specific information. This provides greater accuracy, content relevance, and more control over what the model can say without changing the underlying LLM.

III. Retrieval-Augmented Generation Architecture

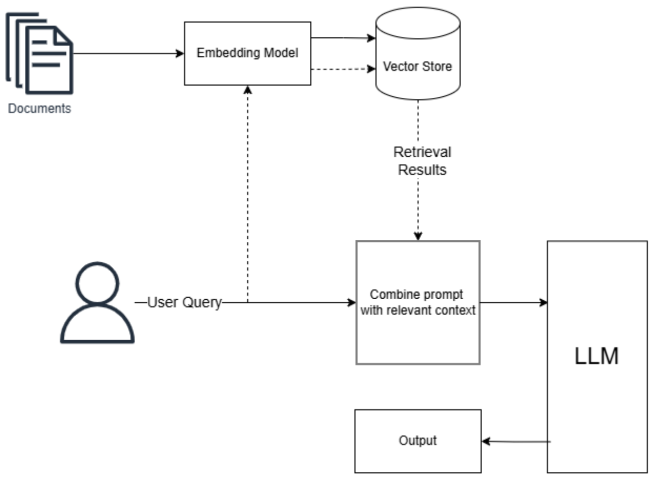

Figure 1. RAG architecture

[4] Retrieval-Augmented Generation (RAG) is a modern approach that extends the capabilities of large language models

by combining them with real-time information retrieval systems. Figure 1 illustrates how RAG works — instead of relying solely on internal knowledge acquired during training, the system searches for relevant documents or data from external sources at the time of the query. These retrieved materials are then fed, along with the original question, to the language model, which uses them to generate a more accurate and informed answer.

This architecture is particularly valuable in educational applications, where up-to-date and context-specific information is essential. Traditional LLMs may provide answers based on general knowledge, but they often lack the precision needed for course-specific tasks or recent subject developments. RAG solves this by connecting the model to curated academic sources, ensuring that students receive responses supported by current and reliable content. Also, it increases the credibility of AI-generated answers and reduces the risk of misinformation.

A. *Embeddings and Text Chunks*

To make RAG systems work, we need a way to connect user questions with the right information stored in documents. This is done using embedding models, which are tools that convert text into numbers — specifically, into vectors [5]. These vectors are mathematical representations of meaning. Texts that have similar meanings will have vectors that are close to each other in this space. This allows compare a user's question with many document chunks and find the most relevant ones.

Since language models can't search documents directly, we first need to split documents into smaller parts, called chunks. These chunks usually have a fixed size — for example, a few hundred words — so they are easier to compare and retrieve. After splitting, each chunk is passed through the embedding model, which converts it into a vector.

In this system, I used OpenAI's "*text-embedding-ada-002*" [6] model, which produces vectors with 1536 dimensions. This means that each chunk is turned into a long list of 1536 numbers that describe its meaning. This model is fast, accurate, and works well for a wide range of topics, which makes it a good choice for educational applications. Once embeddings are created, it can be reused many times without needing to process the text again. Thus, using this technique, LLM gets the opportunity to:

- Match user questions with relevant content, even if the question is phrased differently.
- Be language-independent and can work across many subjects and topics.

However, there are some disadvantages:

- Some very short or very long fragments may be poorly represented.
- The quality of the matching depends on the quality of the fragmentation and the embedding model itself.
- Embedding images and other components distinguishable from text is a complex process and can make it difficult to obtain the desired context.

Despite these issues, embedding plays a key role in making RAG systems efficient. Once we generate vectors, we need a way to store and search them efficiently, and vector databases are used for this.

B. *Vector Database and Reranking*

A vector database is a special type of database designed to store and search high-dimensional vectors. In this system, I used Qdrant [7], an open-source vector database that is fast, scalable, and easy to integrate. Qdrant allows to store millions of vectors and quickly find the most similar ones to a given query. This is done through vector similarity search, which compares the user's question with all the stored document vectors and finds the closest matches.

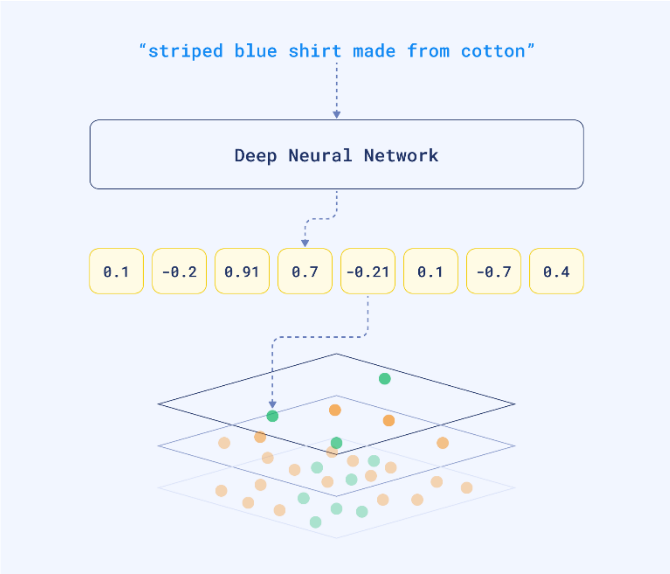

Figure 2. Vector search with Qdrant [6]

The number of matches we return is called Top K. For example, if K=5, the database will return the 5 most similar document chunks. Choosing the right K value depends on the use case:

- Small K (3–5): Better for quick answers where precision matters.
- Large K (10–20): Useful when the question is broad or needs more context.

If K is too small, we might miss some useful information. If it's too large, we might include irrelevant results, which can confuse the language model. In most cases, a value between 5 and 10 works well.

While Qdrant gives results based only on vector similarity. Sometimes, chunks may seem similar in numbers but are not actually the most useful for answering the question. To improve this, I used Cohere Rerank [8]. Cohere Rerank is a model that takes the Top K results from Qdrant and reorders them based on how well they answer the user's question. It scores each result by relevance and then sorts them. This helps make sure that the best chunk is at the top, improving the final answer quality when passed to the LLM.

C. Prompting

Once we have retrieved the most relevant pieces of information from the database, we need to pass it to the language model with user question. This step is called prompt construction, and it plays a key role in getting correct and useful answers. However, it's not enough to just place the context and the question together. The language model needs clear instructions on how to use the context. Without these instructions, it might ignore the provided content or make up answers from its own training data. To avoid this, the prompt must tell the model:

- To use only the provided context when answering.
- To avoid guessing or inventing facts.
- To say *"I don't know"* if the context does not contain enough information.

By including these types of instructions in the prompt, we guide the model to stay grounded in the retrieved data. This is very important in the educational setting, where accuracy and trust are critical.

IV. LangChain and System Architecture

To implement the RAG architecture and integrate vector databases and LLM models more easily, I used the LangChain [9] framework.

A. LangChain

LangChain is a Python-based open-source library designed specifically for building applications that use large language models. It makes easier to connect all the parts of a RAG system — including embedding models, vector stores like Qdrant, and LLMs like GPT. Instead of writing every step from scratch, LangChain provides ready-to-use components.

One of the most useful features of LangChain is the ability to create chains. Chains allow us to build logic on how the answer will be generated, what tools will be used and what context we should use.

B. System Architecture

To make the application more flexible and able to handle different types of student questions, system use architecture based on the concept of agents [10]. Each agent is responsible for a specific task or type of question, and they all work together under one central system.

As shown in Figure 3, at the center of the architecture is the Main Tool Agent. This agent receives the student's input and plays a key role in analyzing it, preparing the prompt, and deciding which specialized agent should be activated. The quality of the prompt in this stage is very important — it must include clear instructions and enough context to help the selected agent understand what kind of task is expected. The Main Tool Agent doesn't answer the question directly. Instead, it acts like a dispatcher that routes the request to one of the five specialized agents, depending on the type of question.

Around this main agent, there are five specialized agents. Each one focuses on a different area of support, such as explaining definitions, solving calculation tasks, or summarizing content. When a student asks a question, the main agent analyzes it and activates the most relevant agent — or combines outputs if needed. This setup makes the system more dynamic and scalable. If needed, new agents can be added later without changing the whole architecture. All communication between agents is handled through LangChain, which helps manage the flow of data and results.

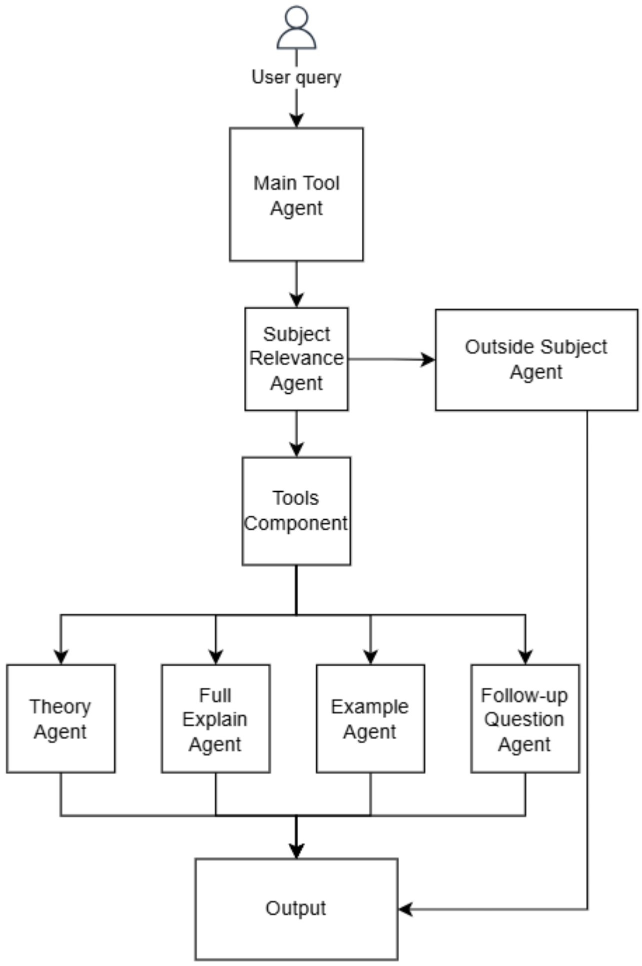

Figure 3. High-level system architecture

C. Subject Relevance Agent

The first step in the processing question is handled by the Subject Relevance Agent. Its main role is to check whether the student's question is related to the subject covered by the system. This helps avoid unnecessary processing and ensures that students only receive answers that are connected to the course content. To do this, the agent uses a special set of documents stored in the vector database. These documents contain a list of all topics included in the subject, with short descriptions and a few keywords. Each topic is stored as a separate chunk, usually representing one page in the original document. This design makes it easy for the system to quickly match a question to a specific topic.

When a student submits a question, the Subject Relevance Agent retrieves the most similar chunks using vector search. It then checks if the similarity score is high enough to consider the question relevant. If it finds a good match, the question continues through the system. If not, it is passed to the Outside Subject Agent. The Outside Subject Agent is a fallback tool that returns a short message indicating that the question is not related to this subject.

D. Thoery Agent

The Theory Agent is designed to handle questions that require more theoretical explanations. This agent focuses on providing clear and accurate answers based on course theory — such as definitions, principles, or explanations of key concepts. To support this, the Theory Agent retrieves context from a separate vector collection that contains only documents from lecture materials. These documents are preprocessed into chunks, each representing a focused topic or section from the lectures. This improves the quality of search results by ensuring that only relevant theoretical content is used when building the answer.

When a question is routed to the Theory Agent, it performs a vector search in the lecture-based collection and retrieves the top-matching chunks. These chunks are then used to build a custom prompt designed specifically for theory-related answers. This targeted prompt helps the model focus on accurate, relevant, and course-specific information rather than generating generic explanations from its training data.

E. Full Explain Agent

The Full Explain Agent is designed for situations where students are just starting to learn about a topic and don't yet know what to ask in detail. This typically happens when a student begins a conversation with a question like: *"Can you explain this topic to me?"*. In such cases, the student doesn't need only a definition or a complex example — they need a balanced overview that includes both the basic theory and a simple example to illustrate it.

This agent works by retrieving context from both lecture materials and example collections. It helps the model generate an answer that is complete but not overwhelming. It's especially useful for students who don't yet have specific questions and are just trying to build a foundation.

F. Example Agent

The Example Agent is designed to do the opposite of the Theory Agent. While the Theory Agent explains concepts in detail, the Example Agent focuses on providing practical examples that show how something is used or solved in practice. This is especially helpful for students who learn better by seeing real use cases or worked-out solutions rather than long explanations. To make this possible, the Example Agent retrieves context from a dedicated collection that contains only examples. Prompt ensures that the model responds in a direct, example-based style, without mixing in unrelated theory or abstract explanations. By keeping theory and examples separate, in the database and in the agents, the system can respond more accurately based on what the student needs.

G. Follow-up Question Agent

The Follow-up Question Agent is responsible for handling all the questions that don't clearly fall into the categories covered by other agents. These are often follow-up questions, general comments, or informal requests that appear during the flow of a conversation. In these cases, the goal is not to retrieve new content or process a specific task, it's to keep the conversation natural and helpful.

Unlike the other agents, this one usually doesn't call any retriever tools. Instead, it works with the summarized conversation history. This gives the agent enough context to understand what the user has already seen and how the response should be adapted. The prompt for this agent is tailored to conversational support. This helps maintain a smooth user experience, especially in longer sessions where the student is building understanding over time. It also ensures that the system doesn't return irrelevant or overly complex answers when a simple conversational reply is enough. The Follow-up Question Agent acts like a smart assistant that supporting the student, guiding the discussion, and keeping the interaction focused and helpful even when a question isn't clearly defined.

Acknowledgment

This paper was supported by DISIC (09I05-03-V2), NEXT (ERASMUS-EDU-2023-CBHE-STRAND-2), EULiST (ERASMUS), CYB-FUT (ERASMUS+), InteRViR (VEGA 1/0605/23).

References

[1] J. Salminen *et al.*, "Using Cipherbot: An Exploratory Analysis of Student Interaction with an LLM-Based Educational Chatbot," in *Proceedings of the Eleventh ACM Conference on Learning @ Scale*, in L@S '24. New York, NY, USA: Association for Computing Machinery, Jul. 2024, pp. 279–283. doi: 10.1145/3657604.3664690.

[2] Z. Chu *et al.*, "LLM Agents for Education: Advances and Applications," Mar. 14, 2025, *arXiv*: arXiv:2503.11733. doi: 10.48550/arXiv.2503.11733.

[3] S. Ganesh and R. Sahlqvist, "Exploring Patterns in LLM Integration - A study on architectural considerations and design patterns in LLM dependent applications," Oct. 2024, Accessed: May 07, 2025. [Online]. Available: https://gupea.ub.gu.se/handle/2077/83680

[4] "Retrieval-Augmented Generation (RAG) and LLM Integration | IEEE Conference Publication | IEEE Xplore." Accessed: May 07, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10845308

[5] A. Mansurova, A. Mansurova, and A. Nugumanova, "QA-RAG: Exploring LLM Reliance on External Knowledge," *Big Data and Cognitive Computing*, vol. 8, no. 9, Art. no. 9, Sep. 2024, doi: 10.3390/bdcc8090115.

[6] "Vector embeddings - OpenAI API." Accessed: May 07, 2025. [Online]. Available: https://platform.openai.com

[7] "Understanding Vector Search in Qdrant - Qdrant." Accessed: May 07, 2025. [Online]. Available: https://qdrant.tech/documentation/overview/vector-search/

[8] "An Overview of Cohere's Rerank Model," Cohere. Accessed: May 07, 2025. [Online]. Available: https://docs.cohere.com/v2/docs/rerank-overview

[9] V. Mavroudis, "LangChain v0.3," Dec. 2024. doi: 10.20944/preprints202411.0566.v1.

[10] G. D. A. E. Aquino *et al.*, "From RAG to Multi-Agent Systems: A Survey of Modern Approaches in LLM Development," Feb. 06, 2025, *Computer Science and Mathematics*. doi: 10.20944/preprints202502.0406.v1.