Ben Carpenter

James Stevens

Professor Hao Loi

Intro to Software Engineering CSC 212-90

22 April 2025

Generative AI and Learning: Using Retrieval-Augmented Generation (RAG) for C++ Tutoring

**Abstract:**

By utilizing advancements from the field of AI, we aim to develop a tutoring app that answers the questions of intro-level C++ programming students, while creating a framework to expand into other academic areas. A Retrieval-Augmented Generation (RAG) approach will provide targeted feedback, dynamically generating context-specific responses by interfacing with a locally hosted AI.

Our goal is to minimize the risk of oversaturated or hallucinated responses. By processing incoming questions from users and communicating with the AI model, a RAG method ensures that the answers generated by the backend are both concise and relevant to the search. The front end will allow students to easily input their questions and view immediate feedback, ultimately enhancing the interactive learning experience. The functionality will be custom-built, while the graphic design from a previous personal project will be used. Combined, these systems will create a user-friendly application that provides students with clear and direct responses to their specific queries.

Overall, the project aims to create a scalable and efficient tutoring tool. Not only would this tool improve students’ understanding of C++ programming concepts, but it could be integrated into other educational platforms. Unlike the VS Code extension Continue or other similar tools, our project is designed as a two-part web application for easy integration into current LMSs (Learning Management Systems). The desired outcome is an engaging, adaptive, and accessible learning environment that can be extended to additional subjects and environments in the future.

**Paper:**

The objective of our project is the design and implementation of an AI-based tutor app that addresses introductory-level C++ programming questions. By using a Retrieval-Augmented Generation (RAG) approach to providing targeted feedback, our goal is to develop a basic, but effective AI framework that facilitates learning. Instead of relying on a pre-stored database of answers, the system will dynamically generate context-specific responses by interfacing with a local Llama3 AI model through Ollama’s local REST API.

The backend, implemented in Python, will serve as the engine of the application. It will process incoming questions from users and communicate with the Llama 3.2 model via the REST API, ensuring that the answers generated are both concise and relevant. This design minimizes the risk of oversaturated or generic responses, providing students with clear and direct explanations tailored to their queries.

We chose to use Retrieval-Augmented Generation because it solves many of the outstanding issues that exist with generative AI. One of generative AI’s largest and most well-known flaws is that of hallucination, where the AI fabricates information with no factual basis. In a recent study, the authors found that out of 5000 ChatGPT responses, 19.5% contained hallucinations. (Li et al) Retrieval-Augmented Generation has been shown to improve code generation and summarization ***Citation Needed***. By implementing Retrieval-Augmented Generation, we intend for this app to draw from a stricter data pool, thus limiting the AI’s answers from straying into hallucination and solving a key potential problem with the use of AI as a tutor.

(*beginning of introduction section)*

The value of our project is in its ability to provide an introductory computer science student with a course-specific learning tool. The app is not intended to replace textbooks or teachers, but to help support currently established methods of education. By focusing on specific material as the basis for our tutor app’s responses, we can add support to the development of problem-solving skills for these students, enabling a stronger knowledge of the material. The app also provides students with active, accurate support outside a tutor’s or professor’s available hours.

Furthermore, we chose C++ as the focus of our tutor app because it is a foundational programming language to the software engineering process. It is important for a beginner to understand the more complex syntax of C++, before they can effectively learn topics such as pointers, memory management, data structures, and algorithms. The building blocks of these data types are laid in an introductory course. The learning curve for C++ is steeper than higher-level languages such as Python. These advanced concepts are more tedious to grasp early on. If a student has access to an accurate, effective tutor app, they will be able to learn the material much more effectively.

Our tutor app also solves another problem, bridging an important gap for many students: the financial one. Personal tutors can be very expensive. Most students can’t afford to pay a human tutor $50-$200 per hour for guidance. For students who are enrolled in schools with high class populations, it may be hard to get tutoring help from other students or faculty. Our tutor app would be a great supplement to traditional teaching utilities such as textbooks and class lectures, making extended support possible as they establish their basic skills.

Retrieval-Augmented Generation is the cornerstone of our project. It is a generative AI process that combines the power of Large Language Models with a curated supply of information. Retrieval-Augmented Generation supplies the AI with contextual information from a data store that is related to a user query. It then uses this context to inform and guide the generative process, giving the AI a stronger knowledge of the topic at hand. (Lu, Yiu, 2.3) (citing references format?)

Online education already exists, but traditional online resources may not be effective for some students and may not offer the best examples early on in their coding education. Some newer students can struggle finding pertinent information without a clear overview of the problem. By using Retrieval-Augmented Generation, the program will dynamically adjust to each unique question, providing students with a personalized response to each question and empowering them with answers that help build a broader understanding.

Retrieval-Augmented Generation has the unique capability to draw on current class materials. By sourcing information from traditional educational resources, such as a textbook, this approach remains consistent informationally, while delivering it in a more engaging, personal way for each student. Using relevant coding examples can often be a more efficient method in reaching inexperienced students, allowing them to better digest the textbook information.

This app can also help build confidence by approaching the information in an unthreatening way. Eliminating the fear of being judged by a tutor or faculty member, students are free to explore answers to their questions in an effective and comfortable environment. By freely pursuing basic questions, students can build their understanding and confidence to ask more precise questions of an instructor. This serves to lower barriers for new students and facilitate quicker, more stress-free progress and eventual mastery of the basics of C++ programming.

*(beginning of related works section)*

Computer Science is currently facing educational obstacles. The reliance on software in all aspects of modern society is growing, causing increased interest in software careers. Schools are seeing record undergraduate enrollment in Computer Science. At the same time, they are facing a both a lack of qualified faculty and varied curriculum challenges. The combined force of these factors is stressing Computer Science education. As Ma, Martins, and Lopes pointed out, “Providing individualized support to many students in introductory courses, especially regarding mastery of complex material, has been challenging.” A strategic use of AI could further the educational reach of the faculty that now exist, reducing the need for an instructor’s direct involvement in simpler questions.

Online materials traditionally used for learning introductory computer science have their limitations. For example, Python Tutor helps students visualize runtime data structure changes during program execution. Visual Algo helps students visualize algorithms through animation. These are good tools and help provide insight into programming, data structures and, algorithms, but they are not always helpful with introductory topics.

Without a working knowledge of computer concepts, many beginning computer science students need to be able to learn and review the intro. Ma, Martins, and Lopes looked at a pool of 455 students at the University of California – Irvine. They deployed five RAG Man tutor apps to assist the students with their supplemental homework assignments. The tutor apps were designed to give guidance, not solutions. In this way, the students developed experience by participating in a more practical process, ultimately finding their own answers.

The research suggested that, “AI tutors can positively impact student success and provide important help, especially to students who would be struggling in challenging courses.” (Ma) The increase in the number of students continuing through the degree path when using the RAGMan tutors was considered statistically significant. Furthermore, the student feedback demonstrated a positive user experience, ensuring a broader use of these tools.

Creating a virtual personal assistant for computer science students is very promising based on the results of such research. Our tutor app provides pressure free, efficient, personalized tutor experience for introductory students that is able to draw on trusted course materials. If we continue to prioritize the feedback and interactions of the students, we can further enhance these learning tools, making them more effective and user friendly.

It would also be a cost-efficient tool for learning. By using Ollama, a locally hosted AI, running the Llama 3.2 model, we were able to reduce the costs often associated with generative AI. Most large language models cost per token, adding operating costs with each use. However, with Ollama, you can run a variety of models locally. After the initial cost of setup, this limits continuing costs of operating to just maintenance and electricity.

This also reduces the environmental impact of AI. A recent article by ???... shows the difference in the impact of cloud AI versus that of a locally run iteration…

BEGIN YOUR REWRITING / ADDING HERE (to the end of the paper)

*(beginning of solutions/implementation section)*

Our app is made with two main modules, the front-end and the back-end. The front-end handles the displaying and storing of messages and user input, and the back-end handles the context information and response generation. We built it like this so that if other people want to integrate it into their existing platforms or software, all they need to do is use the documentation to talk to the back-end.

For our front-end we opted for a user-interface built with Hypertext Markup Language, Cascading Style Sheets and JavaScript. This allows us to build a robust interface that can be used on a website, or hosted by the local user. We built it with several core features, Messaging the AI for help, showing you chat history, remembering you inputs from last session, a chapter selection, and running you code so you can test in-browser.

For the messages we built a display box that follows similar design language to most cell-phone texting apps, as to have it feel familiar to most people. It has a box beneath the display are where you can type a question, a send message, and a clear history message. This is the main way you interact with the tutor application.

This chat are not only takes the input, but its core functions inclue rembering the information and passing that history back to the back-end with every new question. This allows the tutor to have a chain of thought where it can remper you first question and respond to normal human language refrencing previously discussed information.

At the top it also has a drop down that allows you to select the chapter of the textbook you are working in. This not only allows the user to specify where in the class they are, but it aids the generation process by limiting the information that is needed to search through.

To the left of the message area, we have a code box where you can input your C++ code, and it will color code it much like a typical Integrated Development Environment. This allows for a familiar experience for people that are used to a normal Integrated Development Environment. It was achieved by using a utility called highlight.js to parse the information from the text input into a colorized and stylized format, then displaying it behind that are in real time. The code you edit is actually invisible text in the text input on top of the display box. We als made it so a tab places four spaces similar to Visual Studio Code.

Below this are we have a user input as the code cannot be run live in the browser (that requires web assembly, more on that later). In this box the user can put whatever they would normally type into the terminal window while running their code. One they do all this they can click the run button to run the code, and the output is shown in a stylized terminal below.

Once the user question and other collected information (user’s code and chat history) is submitted, it is sent to the back-end for processing. The back-end was originally designed in C++, but due to issues with the libraries we were trying to use, it had to be move to Python.

The back-end has three main components, one that loads all the context information into the database, one that processes questions, and one that runs the user’s code.

When the back-end is initially started the data, text documents stored in folder according to chapter, is gathered and set to Ollama running Llama 3.2 for processing into embeds (can probably grab a reference and explain what embeds are here). In our case, we used the ZyBooks text, chapters one through six for our context documents. Once it has the embeds, the back-end stores them, along with their corresponding context documents and titles, in a ChromaDB database. We chose ChromaDB due to its high functionality and ease of integration into python. The next step for the back-end is to start the Flask server (webserver library for python) and start listening for questions form the front-end. Once all of this is done the back-end is ready to answer questions.

When it receives a question from the front-end, its first task is to separate out the information that was sent into the basic components. It does with the native JSON parsing tools in python. The it takes the user’s question and get the embed of it in order to search the ChromaDB. Once ChromaDB returns a document, the generation process can begin.

The generation passes several bits of information to Ollama running Llama 3.2 to prep it for generation, we do this using the LangChain library, to make formatting the requests to Ollama easier. First it passes the following prompt as a system setting “You are a Tutor for CSC108 - Intro to C++. You are answering questions about C++ coding. Use the following pieces of context to answer the question at the end. If there are No relevant documents found, ask for clarification instead of answering. If you don't know the answer, just say that you don't know, don't try to make up an answer. If it is a vague question, ask for more information. Whenever possible use the Socratic method.” This is followed by passing the context document as a system setting, and then the chat history sent by the front end. Finally, it passes the user’s question, and then begins the generation. Once the generation is done, Ollama send the response back to our back-end, which then adds the document name to the end as a source (chapter and section) so the student knows where to look further on the topic in their textbook. This process allws for accurate and sourced answers, to better help the student. Finally, the completed message is sent back to the front-end, which then displays it to the user.

The final part of the back-end hands processing the user’s code and returning the results. This is achieved from once again parsing the JSON information sent by the front end, and then sending a request to the JDoodle Application Programming Interface containg the user’;s code and desired program input. Once our back-end receives a response, it sends that information to the front-end to be displayed to the user.

ChromaDB efficiently identifies and retrieves the most relevant chunks to the query. So, it’s this combination that creates the accuracy and precise context needed to help the student. This remedies the typical downfall of generative AI where it just generates an answer from metadata without concise context. Again, the retrieval process builds the foundation for the optimal educational experience for the student. It creates a beginning-user-friendly environment.

Thus, avoiding hallucinations. Also, system prompts allow it to be guided against responding with hallucinations if the model doesn’t know the answer or doesn’t have enough information. This is an important part of the process as well.

*(beginning of the section about the modularity of the backend)*

*I have not checked past this, I just got sent it and have no time today to look at it.*

Also, one of our goals (*needs to be added to abstract?)* was to create a tutor app that could facilitate other courses and subjects for future work. Our goal was to have an interchangeable backend that can be integrated easily with other subject matter, if that subject matter data is in .txt format. Basically, we can use the RAG tutor for any subject or course and still maintain consistent, efficient experience for the students.

The main components of our backend operate independently of the C++ material so that we can use it for other subjects quickly and efficiently. Our RAG pipeline uses LangChain for text processing, langchain\_ollama for generating semantic embeddings, and ChromaDB for vector-based storage. The combination of these tools creates a formidable RAG pipeline that is totally flexible in the subjects it can cover and be used as a tutor. Instead of embedding specific rules for C++ education, the backend will use any .txt material that a tutor or instructor provides the system. The texts are divided into chunks, each chunk is transformed into an embedding vector that represents its semantic content. These vectors are then stored in ChromaDB, providing efficient retrieval when a student submits a question.

Furthermore, any user can convert course textbooks, lecture notes, or supplementary materials into standardized .txt files. These files should be segmented by chapters or topics/ideas to facilitate contextual chunking. This subdivision is critical because it preserves the proper context and ensures that the retrieved content directly relates to the student’s/user’s query.

Next, using LangChain, the text files are taken in and split into semantically coherent chunks. Langchain\_ollama then generates embeddings from these chunks, which are stored along with metadata into ChromaDB. This process requires no adjustments to the backend code, which remains the same regardless of the subject matter.

The next phase includes when a user submits a question, the backend converts the question into an embedding using the same model, ensuring compatibility with the indexed content. A semantic similarity search is executed in ChromaDB, and the most relevant text chunks are retrieved. These retrieved chunks are then augmented with the original question to construct a concise prompt for the LLM. Lastly the LLM generates a response that is concise and relevant to the user’s question.

The backend gets its functionality through a REST API, which makes it simple to be integrated into learning systems such as Canvas or Blackboard. Whether the LS uses a web interface or some other interface, the API endpoints take care of queries and responses without any further changes required from the LS. The modularity ensures that teachers, professors, and tutors can employ the backend easily regardless of the learning system.

**Furthermore, this modularity offers significant advantages in terms of scalability. Upgrades to the embedding model or improvements in the vector database can be implemented centrally, benefiting all courses that use the system. This ability to serve multiple subjects with the same core engine reduces redundancy and enhances the overall reliability of the tutoring system. (check this paragraph)**

Finally, the flexibility of our backend allows it to be used with other course materials if they are in a structured .txt format. Simple integration is made possible by the REST API and the other interchangeable components. Additions to the tutor app could include more advanced feedback systems for the students, which seems to be a focus of some of the RAG tutor research that I cited earlier in this paper. The more we can measure the performance of the application, then the more we can improve upon its functionality. One of the main ways to do that is to efficiently get detailed feedback from the students.

The cost of premium AI services of outpaces the speed benefits. We found Llama to be an excellent when weighing speed, flexibility, and cost.

**MY OUTLINE**

Approach

Why choices (RAG, LLM, OLlama, Llama)

The advantage of our app is that it is set up to be able to use any model avalible for Ollama. THis means that you can use a bunch of different models, and with the Results-Agumented Generation, increase their accuracy. This allows for you to reach near comercial qualtiy, but at a very reduced cost.

Another study by Wang and Ramon **(Quantitative Evaluation of Using Large Language Models and Retrieval-Augmented Generation in Computer Science Education)** quantified the performance of different AI models and how effective they were by a cost-effective analysis for instructors. "Implementing RAG enhances the ability of LLMs to answer context-specific questions accurately. This improvement is particularly noticeable in models with integrated course materials and pre-answered question databases and allows open-source models to close some of the gap with GPT-4."

Moreover, they found that advanced model Large Language Models did outperform open-source models in Q&A tasks. However, the performance gap wasn’t significant enough to justify the cost-benefit of using locally hosted open-source models. The needs of the instructor/students should be considered in this regard.

Though we took a unique path, a significant inspiration was Ragman

UC-I Study (Ma…)