Ben Carpenter and James Stevens

Professor Hao Loi

Intro to Software Engineering CSC 212-90

21 April 2025

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

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, communicate with the Llama3 model via the REST API, and ensure 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 just makes up things it thinks sounds good, but is not at all factual. By implementing RAG, we intend to make it so that this app has a stricter data pool that it can draw from making it so that the AI doesn’t generate things that have no basis, thus solving a key problem with using AI as a tutor.

(*beginning of introduction section)*

We believe that our project is important because it can provide an introductory computer science student with a course-specific learning tool. With course-specific material as the basis for our tutor apps responses, we can help support the development of problem-solving skills for these students. RAG has been shown to improve code generation and summarization, thus helping give the student a stronger knowledge of the material.

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++, so that they can begin to learn effectively topics such as pointers, memory management, and data structures and algorithms. The building blocks of these data types are laid in the introductory course.

The learning curve for C++ is more steep than higher-level languages such as Python. Again, concepts such as pointers and references are more tedious to grasp early on. If a student has access to an accurate, effective tutor app that mitigates hallucinations, they will be able to learn the material much more effectively.

Our tutor app also solves a key problem and bridges 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. There’s no question that our tutor app would be a great supplement to traditional teaching utilities such as textbooks and class lectures.

RAG is retrieval-augmented generation. This is a method of generative AI that is the cornerstone of our project, and for good reasons. RAG combines the power of LLMs with a curated supply of educational information. RAG allows them to retrieve contextual information relating to a user query from a data store and then use this retrieved context to inform and guide the generation process; thus, giving them stronger knowledge of the topic at hand. ***(Lu, Yiu, 2.3) (citing references format?)***

Moreover, this approach will provide students with a personalized response to their questions. The program will dynamically adjust to the questions the students ask, versus traditional online resources that are more rigid in their explanations. Traditional online resources may not be the best way for some students to learn and may not offer the best examples early on in their coding education.

In addition to giving personalized responses and examples, the strength of RAG is the quality of the educational content that it is drawing from. This is a wonderful way to utilize traditional utilities of education, such as a textbook, but in a more engaging, unique way for each student that uses the app. We believe that relevant coding examples help get students started on their path to learning, and when they can access this type of utility, they will be able to spend adequate time to learn the material effectively.

The app can lower barriers to learning for new students and help drive progress and eventually mastery of the important basics of C++ programming. It would also eliminate the fear of being judged by a tutor or faculty, thus providing more personal and comfortable experience for the student. This could foster growth and learning at a greater rate than usual.

*(beginning of related works section)*

Some obstacles computer science education is facing nowadays include high undergraduate enrollment, lack of resources, including lack of qualified faculty, and curriculum challenges. The obstacles are fueled by the growing reliance of software in modern society and the increasing appeal of software careers. **(citation?)** These aren’t the sole obstacles of computer science education, but they are some important ones that can be alleviated with a RAG tutoring program.

Although it may not be able to alleviate high undergraduate enrollment itself, it may have a positive effect on managing it. For some background, at one California university, UC Irvine, has reached its peak in the last three years with over 1,000 incoming freshmen per year. **(Integrating AI Tutors in a Programming Course)** As Ma, Martins, and Lopes pointed out in their research paper, providing individualized support to many students in introductory courses, especially regarding mastery of complex material, has been challenging.

Previously, some traditional online methods of learning introductory computer science material have been things such as Python Tutor, which helps students visualize runtime data structure changes during program execution. A similar tool, Visual Algo helps students visualize algorithms through animation. These are good tools and help provide good insight into programming, data structures and algorithms, they don’t offer support in other topics that are included in an introductory class.

Beginning computer science students need to be able to learn and review the intro material because many of them may not have a working knowledge of computer concepts such as files and memory. Basically, the Ma, Martins, and Lopes research looked at a pool of 455 students at UC-Irvine, and they deployed five RAGMan tutor apps to assist the students with their supplemental homework assignments. The tutor apps were designed to not give solutions but to guide the students in taking the next step towards solving their questions.

Furthermore, the results of the Ma study were positive to say the least. Although there was a larger percentage (44% vs. 42%) of A grades in the 2023 class that didn’t use AI tutors vs. the 2024 class that did, 3% less students received F’s, 5% more received Bs in 2024, which is significant. The very basis of continuing the computer science education tract was ensured, arguably by the RAGMan tutor deployment. About student feedback, it was mostly very positive as well.

There’s no question our app aims to help alleviate the gap between the obstacles of computer science education and ensuring the development of new competent, successful computer science students. 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.

Moreover, they found that advanced model LLMs 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.

In addition to being cost effective, an open-source model using RAG made a great impact as well. They found that implementing RAG enhanced the ability of the LLMs to answer questions and provide concise feedback. They also found it particularly noticeable when the model was integrated with course materials, allowing them to close the gap with the likes of GPT-4, for example.

These studies, and other similar ones emphasize how future research should aim to measure student feedback and student engagement. The paper **(Can Small Language Models with Retrieval-Augmented Generation Replace Large Language Models When Learning Computer Science?)**  shows that SLMs can provide enhanced data privacy and control and provide enriched learning experience for students. It’s imperative that instructors keep their course materials secure and in-line with organizational policies.

Our tutor app would help a student or instructor, institution with limited resources as it would be a cost-efficient tool for learning. Creating essentially a personal assistant for computer science students is very promising based on the results of prior research. This RAG method would be more engaging and productive than current resources, such as Stack Overflow, Python Tutor, etc. Hallucinations would be minimized, which contrasts with basic ChatGPT models where roughly 20% of responses are hallucinated.

Undoubtedly the RAG tutor app provides pressure free, efficient, personalized tutor experience for introductory students. The ability given to the LLM or SLM to draw on trusted course materials is extremely beneficial to the incoming student. If we continue to prioritize the feedback and interactions of the students, we can further enhance these awesome learning tools.

*(beginning of solutions/implementation section)*

It’s clear that our main goals are to facilitate learning and minimize the hallucinations made by generative AI. The latter goal is propelled by our tutor app’s RAG methodology. Again, the retrieval-augmented generation combined with the Llama 3.2 AI model delivers personalized, contextually relevant and concise answers to students’ questions. This approach will minimize hallucinations.

The RAG pipeline begins with the indexing of the collection of data/documents that are being leveraged as the source of relevant material. In our case we used the ZyBooks text, chapters one through six to supplement the generated answers. The whole process of RAG begins when we index the collection with ChromaDB, a vector database that will store the converted documents as embeddings. The embeddings are small, semantic representations for similarity searches.

We chose ChromaDB because it is a seamless match with Python and is known for its efficiency. ChromaDB became the preferred choice. The indexing phase is crucial to facilitate accurate matches to the user queries.

Moving onto the next step, the retrieval stage. When the student makes a query, the query is matched with the most relevant data in the vector database. Next, the augmentation occurs. So, the retrieved information is combined with the user’s query to create an augmented input that will be sent to the Llama AI model. The enhanced context will enable the LLM to generate a more concise response.

Lastly, the generation occurs with an answer that integrates the Llama model’s knowledge and the supplied relevant context. The answers seem more calibrated and centered around the students’ query.

The first step to minimizing hallucinations is the indexing stage. We chose the first six chapters of the ZyBooks online text, which is used in our CSC-108 Intro to Computer Science course here at Quinsigamond Community College. The indexing happens before the runtime of the RAG itself. Indexing builds a strong foundation for the retrieval of concise data.

Next the data is processed into chunks using Langchain and then converted into numerical vector embeddings by importing Langchain\_Ollama. Using Ollama’s built in embedding function provides better performance and stability for the app. Then the embeddings are indexed in ChromaDB, the vector database. ChromaDB offers fast similarity search and semantic retrieval.

The indexing pipeline sets up before runtime and ensures the accurate retrieval of data. The retrieval process is the basis for concise responses. The retrieval uses the indexed information stored in ChromaDB and employs the embeddings to find the most relevant information for each user query.

Basically, when a student submits a query, the retrieval functionality processes the text into a vector embedding, using the same embedding function we used in the indexing stage. This ensures that the query embeddings and indexing embeddings reside in the same space and are similar in context.

The query embedding is then passed to ChromaDB, where the semantic search takes place. 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.

Furthermore, the retrieved content is then inserted into a structured prompt along with the original user query. The prompt would be a guideline or tone for the LLM to follow. This augmented prompt is then sent to the Ollama 3.2 model, and a response is generated based on the contextual data. The result is the student receives an accurate answer that is grounded in the course material. 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)*

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.

(*why we chose ChromaDB vs. FAISS, .txt. vs PDF parsing)*

In designing our backend, we considered different options in terms of balancing performance, cost, privacy, and maintainability to arrive at an optimal, self‑hosted architecture. We first chose FAISS but then compared ChromaDB and FAISS for indexing embeddings. ChromaDB was chosen because it integrates seamlessly with LangChain, supports metadata with vectors. Although FAISS provides superior raw output and GPU acceleration, its lack of ease to implement made it the less favorable choice.

When our system reads a question or a piece of textbook, it needs to turn that text into a set of numbers so it can compare similarity between documents and queries. Choosing how to do this embedding was an important decision for us. We decided to run embeddings locally with Ollama. Keeping things in‑house protects student data and keeps costs steady, even if it meant accepting a small drop in accuracy. So, to make up for that we manually edited our text files and organized them to be clear and well-defined sections. The user can select each section from a dropdown menu, and this ensures that the response will be based on that material.

When it came to loading course materials into our system, we had two options: plain .txt or PDF parsing. We chose the former as it is more reliable and error-free. By not using PDFs we avoided weird characters, columns, or images that can cause issues for LLMs. PDFs often contain images and side bars that confuse extractors, resulting in messy text. Even though it took longer to edit and clean the .txt files, the user always gets clear, concise information because of the .txt files.

WORKS CITED

1. **Seven Failure Points When Engineering a Retrieval Augmented Generation System**

**Authors**: Scott Barnett, Stefanus Kurniawan, Srikanth Thudumu, Zach Brannelly, Mohamed Abdelrazek

1. **Quantitative Evaluation of Using Large Language Models and Retrieval-Augmented Generation in Computer Science Education**

Authors: Kevin Shukang Wang and Ramon Lawrence

1. **Integrating Small Language Models with Retrieval-Augmented Generation in Computing Education: Key Takeaways, Setup, and Practical Insights**

Authors: **Authors**: [Zezhu Yu](https://dl.acm.org/doi/10.1145/3641554.3701844), [Suqing Liu](https://dl.acm.org/doi/10.1145/3641554.3701844), [Paul Denny](https://dl.acm.org/doi/10.1145/3641554.3701844), [Andreas Bergen](https://dl.acm.org/doi/10.1145/3641554.3701844), [Michael Liut](https://dl.acm.org/doi/10.1145/3641554.3701844)

1. **Integrating AI Tutors in a Programming Course** Iris Ma University of California, Irvine Irvine, California, USA huaiyaom@uci.edu Abstract Alberto Krone Martins University of California, Irvine Irvine, California, USA algol@uci.edu Cristina Videira Lopes University of California, Irvine Irvine, California, USA [lopes@uci.edu](mailto:lopes@uci.edu)
2. **Can Small Language Models with Retrieval-Augmented Generation Replace Large Language Models When Learning Computer Science?**

**Authors**: [Suqing Liu](https://dl.acm.org/doi/10.1145/3649217.3653554), [Zezhu Yu](https://dl.acm.org/doi/10.1145/3649217.3653554), [Feiran Huang](https://dl.acm.org/doi/10.1145/3649217.3653554), [Yousef Bulbulia](https://dl.acm.org/doi/10.1145/3649217.3653554), [Andreas Bergen](https://dl.acm.org/doi/10.1145/3649217.3653554), [Michael Liut](https://dl.acm.org/doi/10.1145/3649217.3653554)