Technological Feasibility

October 24th, 2025

C-LASS (Cybersecurity Learning with AI for Static Systems)

Sponsor: Dr. Lan Zhang

Assistant Professor, School of Informatics, Computing, and Cyber Systems, Northern Arizona University

Team Mentor: Scott LaRocca

Team Members: Sean Golez, William Barnett, Kayden Vicenti, Colton Leighton

Table of Contents

1. Introduction

Capture the Flag (CTF) challenges are a widely used and engaging approach for learning cybersecurity concepts such as vulnerability discovery and program analysis. They promote hands-on, inquiry-based learning and develop critical problem-solving skills that reflect real-world security scenarios. However, CTFs focused on static analysis are notoriously difficult to master. Unlike dynamic analysis, which relies on observing a program's behavior during execution, static analysis requires deep reasoning about code structure, data flow, and potential vulnerabilities without running the program. Despite their value, there are limited structured learning pathways to guide students through these complex topics, leaving many learners to rely on sporadic guidance that is often unavailable.

Artificial Intelligence (AI) has the potential to bridge this gap by offering scalable, adaptive, and personalized tutoring support. Yet, current large language models (LLMs) struggle to provide the depth, consistency, and accuracy required in this domain. Their limitations are significant. First, they lack grounding in authoritative content, often producing plausible but inaccurate explanations. Second, they provide inconsistent responses to similar questions, which undermines the learner's trust. Third, their domain knowledge is shallow because concepts like dataflow analysis, abstract interpretation, or taint propagation are rarely represented accurately in general-purpose models. Fourth, they lack a structured explanation path, meaning learners cannot explore prerequisite knowledge or concept hierarchies essential to mastering static analysis. Finally, most AI agents remain passive responders rather than active tutors capable of engaging learners through guided questioning and adaptive feedback.

In general, there are no specific tools available to help students visualize or conceptualize Capture The Flag (CTF) challenges without direct guidance from instructors. Our client, Dr. Lan Zhang, an Assistant Professor at Northern Arizona University, teaches SE450, a software engineering course focused on software testing through CTF challenges. To solve these challenges, students use a disassembly tool that translates binary files into human-readable code. From there, they must write Python programs that leverage Angr and SimProcedure to analyze and solve the given problems. Currently, there are no clear or accessible educational resources that provide structured support, aside from instructor or tutor intervention. As a temporary solution, the instructor utilizes ChatGPT for assistance with CTF tasks.

The envisioned product is an AI tutoring agent specifically designed to support static analysis CTF challenges. It will be a web-based tutoring system that integrates knowledge graphs and Retrieval-Augmented Generation (RAG) to provide accurate, consistent, and structured explanations. Both students and instructors will use this system. Instructors will have the ability to edit and expand the knowledge graph. In contrast, both students and instructors will be able to generate knowledge graphs based on provided information or new content. Students will interact with a large language model (LLM) through a text-based chat interface, receiving consistent and conceptually grounded answers across sessions. The system will use an LLM enhanced with RAG, combining curated static analysis CTF documentation and explanatory materials with a static analysis knowledge graph that encodes key concepts, relationships, and example

challenges. Through a web interface, students can ask questions about CTF problems and receive AI-generated tutoring responses that reference relevant knowledge graph nodes. The system will also support context-aware follow-up interactions, allowing students to ask clarifying questions or explore deeper conceptual layers. By anchoring explanations in the knowledge graph, the system ensures consistency, accuracy, and alignment with authoritative content over time.

Our system will enable students to better understand the broader context and structure of CTF challenges, helping them recognize how different problems and techniques are interconnected. It will also allow both students and instructors to categorize, visualize, and locate related challenges efficiently. Ultimately, this system will empower students to explore static analysis CTFs more effectively, providing them with expert-level guidance at scale. The AI tutoring agent will help learners deepen their conceptual understanding of program analysis, vulnerability detection, and formal reasoning, without requiring constant human supervision. Instructors will benefit from reduced teaching overhead, while students will gain learning support. The long-term goal is to cultivate a more confident, capable generation of developers and security researchers who can approach complex software security problems with independence and insight.

2. Technological Challenges

Now that we have established the impact of our project, we are ready to turn to a more careful analysis of the AI tutoring agent. In the next section, we discuss the technological challenges for our product. At this early stage in the project, we are in the process of analyzing the key technological challenges, identifying possible alternatives, and selecting which of those alternatives are the most promising solutions. In creating this software, there will be several technological challenges ranging from the hosting and development of the full-stack web-based application to the intricacies of the AI tutoring agent. At the early stages of this project, the main technological challenges involve determining how to develop the web application and what existing tools can effectively and efficiently meet the requirements for the AI tutoring agent for static analysis CTF challenges. Specifically, the challenges involved with the implementation of the knowledge graph, the LLM, the backend, and the frontend.

2.1 Knowledge Graph Framework

One of the main focuses of our project is to include a domain-specific knowledge graph that is curated to contain the relationships between the relevant information for specific static analysis CTF concepts, problem-solving strategies, and solutions to specific challenges. It is theorized that utilizing a knowledge graph database rather than using a simple similarity search over documents will yield greater accuracy and consistency for answering educationally structured questions relating to CTF problems. However, a knowledge graph is very conceptual and not commonly used in software databases. Furthermore, this knowledge graph database

would need to be secure for reading and writing since the intention is that teachers should be able to edit it, and the LLM should be able to use it to generate knowledgeable and relevant answers.

2.2 Knowledge Graph Retrieval Augmented Generation

By default, LLMs do not explicitly utilize context from documents, much less a knowledge graph database. Rather, the responses generated by LLMs are based on the large amounts of data that the model was trained on. For general tasks, LLMs can give accurate and reliable responses. But, for domain-specific questions, such as those relating to static analysis and CTF challenges, the concepts are not covered well in generic models. This leads to incorrect or imprecise responses to these questions. Thus, for our project, there exists a need for a method such that an LLM can leverage a curated knowledge graph when generating its responses in order to avoid these issues.

2.3 Backend Infrastructure

As for most web applications**,** we also need to decide what framework we will use for our backend. But, for our application specifically, we need a framework that will allow us to securely host the LLM and knowledge graph for users to query and edit, respectively. This framework will need to be the "glue" of our application, as it is the medium through which we will be implementing our LLM query pipeline and connecting users to the knowledge graph database. Due to the conceptual difficulty of implementing an effective AI tutoring agent, we believe that it is important for the backend to not only be capable of efficiently managing the model and database, but also to it will allow for seamless interaction between the components of our system.

2.4 Frontend Knowledge Graph Interface

As for our frontend framework, it is essential that it is capable of both hosting the "chatbot" interface, which we envision to be very basic at this point, as well as the knowledge graph "editing" interface. For the latter, we envision an intuitive editor that will allow instructors to create new nodes for each concept and relate concepts using labeled edges, similarly to a web-based diagram editor,but with ultimately less freedom since only a single structure is expected to be created. It is also vital for our system to allow for "subgraphs" within the knowledge graph for concepts surrounding a specific CTF topic, such as procedures for specific problems, information about commonly used tools, etc. Therefore, our chosen frontend framework should be capable of supporting such an interface, ideally with a straightforward implementation.

2.5 Frontend Chatbot Interface

Continuing on the technological challenges for the frontend framework, the additional user-facing interface necessary for our project is a chat for users to converse with the AI tutoring agent. This is arguably the most essential component of our software, since it is expected that the

majority of users will be students who need help with solving static analysis CTF challenges, and the extent of these users' interaction with our system is solely using the chat to ask questions. So, our chosen framework for this interface should be responsive and lightweight, so that our product will be able to run efficiently on the web browser of any device, such that user satisfaction is high.

2.6 Application Database

Another requirement for our system would be that it has some way to securely store user information and chat logs. One of the requirements of deploying an effective conversational LLM is that it requires explicit context given in the background of each prompt in order to appear as though it "remembers" past conversations. So, past conversations will need to be stored in a way that is secure and expandable. Furthermore, it may be beneficial for the overall user experience for these past conversations to be stored as separate "chats" for users to save and revisit, without losing the progress of the information given to them by the LLM. Then, in order for these features to remain secure for each user, this database must also store user-identifying information, such as a username and password, along with additional user information if further customization is preferred. Thus, we must select a database framework that allows for secure storage, updating, and retrieval of large-scale data, while integrating seamlessly with both the LLM and our overall technology stack.

3. Technology Analysis

In our early exploration of the project, we have identified six core system aspects that require important design decisions to be made: the knowledge graph database, the knowledge graph information retrieval, the backend, the knowledge graph frontend interface, the chatbot frontend interface, and the chat and user information database. In order to determine the design decisions for each of these aspects, a thorough analysis is required. For each, we will go over the specific characteristics needed for our system, analyze multiple solution alternatives, and then ultimately decide on the approach that we will go forward with in the development of our system.

3.1 Issue 1 - Knowledge Graph Database Framework

For our project to be successful, there needs to be an effective way to store and model information related to CTF challenges and static analysis, and constructively represent the relationship between two concepts. This requires more than just data storage; it needs a structured framework that allows relationships, dependencies between concepts to be understood and navigated. In relation to this, the system needs to incorporate a practical knowledge model that supports structured representation, reasoning, and scalable integration of new information.

3.1.1 Desired Characteristics

- Efficiency: The system should support incremental updates, so when new data is added, the graph doesn't need to be completely rebuilt.
- Scalability: Designed to grow with new data sources, automatically integrating new nodes and relationships.
- Reliability: Tracks provenance and source credibility for each relationship or entity, improving data trustworthiness.
- Accessibility: Provides a web-based visualization interface, allowing users to explore connections and relationships interactively.

3.1.2 Alternatives

- Neo4j:
    - Neo4j is a native graph database, which means that it implements a true graph model. Instead of using a "graph abstraction" on top of another technology, the data is stored in Neo4j.
- Azure Cosmos DB - CosmosAIGraph:
    - CosmosAIGraph applies the power of Azure Cosmos DB to create AI-powered knowledge graphs. This technology integrates graph database capabilities with AI to provide a way to manage and query complex data relationships.
- LlamaIndex - Property Graph Index:
    - The Property Graph Index is a labeled property graph representation that enables modeling, storage, and querying of a knowledge graph. There is the ability to categorize nodes and relationships into types with associated metadata and express complex queries using the Cypher graph query language

3.1.3 Analysis

- Neo4j
    - Efficiency: Highly efficient for graph operations with minimal overhead for traversing large networks. Supports incremental updates and optimized indexing for relationships.
    - Scalability: Handles medium to large-scale datasets effectively, though vertical scaling may be required for extremely large graphs.
    - Reliability: Provides strong ACID compliance, ensuring consistent and reliable data management.
    - Accessibility: Comes with Neo4j Browser and Bloom for intuitive visual exploration. Requires setup and maintenance but offers good integration with web applications via REST APIs.
- Azure Cosmos DB - CosmosAIGraph
    - Efficiency: Optimized for distributed workloads and real-time data. Can integrate seamlessly with Azure AI services for automated updates.
    - Scalability: Suitable for large-scale institutional deployment.

    - Reliability: Enterprise-level reliability and redundancy built into Azure infrastructure. Tracks data provenance and offers robust monitoring tools.
    - Accessibility: Requires an Azure subscription and configuration, but provides strong cloud-based accessibility and integration with visualization frameworks such as GraphRAG dashboards.
- LlamaIndex - Property Graph Index
    - Efficiency: Lightweight and adaptable. Works directly with existing large language models and retrieval pipelines, enabling flexible updates without database overhead.
    - Scalability: Depends on backend integration; can scale effectively when connected to modern vector databases or graph backends like Neo4j
    - Reliability: Reliability depends on underlying data sources and indexing strategy. Less mature than Neo4j or Cosmos DB for long-term persistence.
    - Accessibility: Easily integrates with LLM-based applications and web interfaces, supporting dynamic querying and interactive exploration.

3.1.4 Chosen Approach

Knowledge graph database	Efficiency	Scalability	Reliability	Accessibility
Neo4j	4	4	4	4
Azure Cosmos DB - CosmosAIGraph	2	2	2	1
LlamaIndex	3	2	2	4

Neo4j was selected as the preferred approach for implementing the knowledge model. Based on the rating, Neo4j allows for an efficient, reliable, and strong visualization capability. Due to it being a native graph database, Neo4j allows for fast travel of complex relationships between entities, such as vulnerabilities and analysis techniques, making it well-suited for CTF knowledge and static analysis. It supports incremental updates and ACID compliance, ensuring scalability and data integrity.

3.1.5 Proving Feasibility

To prove feasibility, we will begin by developing a prototype Neo4j knowledge graph that models static analysis concepts such as control flow, data flow, and vulnerability patterns. This prototype will include example CTF challenges represented as interconnected nodes and

relationships. Cypher queries will retrieve related concepts and demonstrate how the AI tutoring agent can leverage these relationships to generate accurate, context-aware responses. The prototype will be integrated through the web-based interface that visualizes the graph. Successful validation will confirm that Neo4j can efficiently store, query, and visualize static analysis concepts.

3.2 Issue 2 - Query the Knowledge Graph

Our LLM will connect to the knowledge graph to obtain relevant information to answer domain-specific questions given by users. This is a vital function for our system since it is ultimately what will allow our AI tutoring agent to outperform popular LLMs, particularly in the agent's response regarding static analysis CTF challenges, as well as positioning our product as a unique and efficient teaching tool. The desired result for this functionality is that given a domain-specific query from the user, the LLM should be able to automatically reference the knowledge graph database, retrieve documents from the relevant concepts in the database, and conceptualize how the retrieved documents relate to each other to generate a fully structured and contextualized response.

3.2.1 Desired Characteristics

- Speed: Having quick lookup speeds is essential to having an efficient system, especially given that users are unlikely to be satisfied with a chatbot that is extremely slow to reply, even if it generates accurate responses.
- Simplicity: As developers of this project, it is also important that this functionality is simple both conceptually and in integration. For the purposes of designing and presenting our solution for the design of the overall LLM pipeline, having a simpler retrieval system will make it easier for us to conceptualize and fully comprehend the fine details. Having a simpler system will make development and maintenance equally easier, since there will be fewer points for failure and fewer edge cases to test for.
- Graph-Awareness: It is important that the retrieval is able to effectively and accurately utilize the knowledge graph structure of the database. In order for our system to effectively take advantage of the curated knowledge graph, the retrieved documents must contain the context of the nodes and relationships that they were retrieved from. This would allow the LMM to effectively structure the response using the full relevant context of the knowledge graph, rather than reciting information from the documents.

3.2.2 Alternatives

- Basic Rag:
    - One alternative is using basic retrieval augmented generation with a vector search over a static graph database. This would involve using the Python library, LangChain, to do a basic similarity search query over the database. This is compatible with Neo4j.

- GraphRAG:
    - GraphRAG, an architectural pattern designed around Neo4j's graph database that utilizes both vector and graph search, and would be implemented through Python.
- Memento MCP:
    - Memento MCP is a knowledge graph memory system for LLMs. This was found when researching non-RAG strategies for allowing the LLM to interact with a knowledge graph database. This was recently released and is a non-Python implementation approach. The main idea behind this approach is that it acts as a memory for the LLM, rather than just a simple document storage.

3.2.3 Analysis

- Basic Rag
    - Speed: The Basic RAG approach was the fastest by a noticeable margin. Since it only performs a straightforward embedding search and retrieval, there's very little computational overhead. Results are returned almost instantly, which makes it ideal for quick searches or prototype systems.
    - Simplicity: This setup was also the easiest to implement. It required minimal configuration beyond creating the embeddings, storing them, and defining a retrieval function.
    - Graph-Awareness: Due to the simplicity of this method, it provides the system with no awareness of relationships between documents or entities. It treats all data as isolated chunks of text, so the LLM will not have any context of the relationship of the information retrieved beyond the text itself.
- GraphRAG
    - Speed: This configuration includes additional processing steps due to its use of graph traversal and relational retrieval, making it slightly slower than Basic RAG. Despite this, the search times were reasonable and responsive.
    - Simplicity: Setting up GraphRAG requires defining a graph schema and a retrieval pipeline to allow retrieval to leverage both embeddings and graph structure. Even so, the overall process was still straightforward to implement into a predefined knowledge graph database.
    - Graph-Awareness: Compared to Basic Rag, this approach provides the LLM with contextual understanding. It provides the system with the most relevant documents based on a query, along with information about how those documents relate to one another. This provides the LLM with a more interconnected view of the knowledge space.
- Memento MCP
    - Speed: The Memento MCP system is the slowest of the three, primarily because of its more complex pipeline and the overhead of interacting with the memory server. Queries require the system to perform updates or multi-step lookups.

- Simplicity: Integrating the MCP framework into a Python-based environment is challenging, as it operates more as an external memory service than as a library you can directly embed into the application. It also includes several advanced features such as confidence decay, memory updates, and temporal awareness.
- Graph-Awareness: This method is capable of treating the knowledge graph as an evolving memory system, rather than as static reference data. So, the LLM can store, query, and update the graph directly through the Model Context Protocol, allowing it to maintain a persistent and temporally aware understanding across sessions.

3.2.4 Chosen Approach

Framework	Speed	Simplicity	Graph-Awareness
Basic RAG	5	5	0
GraphRAG	4	4	4
Memento MCP	3	2	5

We analysed each of these alternative approaches by downloading all of the necessary packages and testing them with a very basic Neo4j database running on a CPU under Windows OS, following basic tutorials that we could find online. The basic RAG approach was the easiest to implement a working version of, while the Memento MCP approach was the most convoluted to get working. Furthermore, the Memento MCP was difficult to implement directly into a Python program, and the use of executing external commands is required. We roughly timed the search times, although they were all similar due to the dataset being so small, and closely examined the features of each alternative. Based on this analysis, we have chosen to move ahead with the GraphRAG approach because it maintains a good balance between quick retrieval times and simplicity, while still giving the LLM the knowledge graph context that is necessary for our system. Although Memento MCP comes with several advanced knowledge graph features, these are not entirely necessary for the use case of our project since we only want to use the knowledge graph as a contextual document database for domain-specific information to start.

3.2.5 Proving Feasibility

We have tested that GraphRAG works with our knowledge graph database of choice, Neo4j, and it was developed with the intention of doing so. We have also tested to ensure that it is compatible with an OpenAI LLM model implemented through Python, our chosen backend language. We plan on further validating this approach with a more complex knowledge graph database that better represents the structure of the database, which we will be working with,

which is one with designated subgraphs and edges connecting concepts to specific CTF problems.

3.3 Issue 3 - Effective Backend

The backend framework is the foundation of our tutoring platform, acting as the central hub that connects the user interface with the large language model and with the knowledge graph. The backend is responsible for handling all communication between these systems, ensuring that user inputs are processed efficiently and accurately. Its responsibilities surrounding this include managing authentication, routing API requests, interfacing with the Neo4j database, and connecting responses from the LLM through a facilitation framework like LangChain.

The main challenge in selecting the backend framework lies in determining which solution can best meet the project's technical demands while aligning with the team's Python-based infrastructure. Because both students and instructors will use the system, it needs to support concurrent queries and scalable data handling. The chosen backend will not only affect the stability of the product but also influence the long-term adaptability of the entire platform.

3.3.1 Desired Characteristics

- Maintainability: The backend framework needs to support modularity within its integration, allowing the reuse and decoupling of code
- Security: Security is crucial for the backend system to integrate due to its responsibility to handle sensitive information, such as passwords and personal information. Support for external backend-supported libraries needs to be considered.
- Performance: For this project, high performance is necessary for ensuring LLM responsiveness for use queries. Due to the real-time interaction nature of our system and the extensive pipeline needed for information to be retrieved and generated, a quick backend framework would allow for a smooth and responsive experience.

3.3.2 Alternatives

- FastAPI
    - Fast API is a modern and high-performance framework for building APIs with Python in mind. FastAPI supports both asynchronous and synchronous operations across systems, making the framework a strong choice for REST API integrations.
- Node.js with Express

- NodeJS with Express is another strong and reliable framework that is a commonly used tool stack for building fast and scalable web applications. Together, they can provide a strong foundation for effective and efficient RESTful API services to be made available.

3.3.3 Analysis

- FastAPI
  - Maintainability: Offers excellent maintainability through its clean, Pythonic structure and automatic OpenAPI documentation. Because it operates entirely within the same language ecosystem as the LLM, LangChain, and Neo4j, it avoids the need for translation layers or middleware, simplifying debugging and long-term upkeep.
  - Security: Provides robust, built-in support for secure authentication and authorization mechanisms such as OAuth2 and JWT. Its strong typing and validation features reduce common security vulnerabilities, while its integration with Python libraries ensures consistent and maintainable security practices across the platform.
  - Performance: Delivers high performance through its asynchronous, non-blocking architecture. It can process multiple concurrent requests from students and instructors efficiently, reducing latency and ensuring responsive interactions with both the LLM and the knowledge graph.
- Node.js
  - Maintainability: Node.js with Express provides a mature and scalable backend option, but it introduces complexity when paired with Python-based systems. Integrating the LLM and Neo4j through middleware adds extra layers for data handling and debugging, making long-term maintenance more demanding.
  - Security: Node.js benefits from a wide selection of mature security packages and middleware such as Passport.js and Helmet, offering strong options for authentication and data protection. However, the heavy reliance on third-party modules requires diligent dependency management to maintain consistent security standards.
  - Performance: Node.js offers strong scalability and real-time responsiveness due to its event-driven, non-blocking architecture. It can efficiently handle many concurrent user connections, making it ideal for high-traffic applications, though additional translation layers may introduce latency when interacting with Python-based AI components.

3.3.4 Chosen Approach

Backend Framework	Maintainability	Security	Performance

FastAPI (Python)	5	5	5
Node.js (Express)	4	4	4

After evaluating the alternative backend options, FastAPI was selected as the backend framework for this project due to its strong compatibility with the current Python-based technology stack and its asynchronous operations. FastAPI will serve as the central communication layer between the ReactJS frontend and the Neo4j knowledge graph, orchestrating AI responses through the OpenAI LLM via chain links from LangChain. This native Python integration eliminates the need for translation layers or inefficient middleware, simplifying interaction with AI libraries and graph database drivers, while enabling efficient processing of multiple concurrent requests from students or teachers. The chosen approach balances performance, maintainability, and integration with the existing stack, while leaving room for future expansion, such as more advanced knowledge graph manipulation, enhanced caching for data-heavy queries, and extended AI interaction patterns.

3.3.5 Proving Feasibility

The selected FastAPI backend has been validated for integration with the Neo4j knowledge graph, OpenAI LLM via LangChain, and the ReactJS frontend. Initial testing confirmed that core operations such as AI query handling, RAG-based retrieval, and knowledge graph lookups work reliably within the Python environment. Containerization with Docker ensures easy deployment and compatibility with the MCP server setup.

Preliminary load testing remains the next step in indicating that FastAPI can handle multiple concurrent requests efficiently with low latency. Results would demonstrate the backend's ability to meet the requirements for performance, scalability, and maintainability for this project. Further evaluation will focus on optimizing performance under concurrent load, ensuring robust error handling, and validating the system’s scalability and reliability in a realistic teaching environment.

The prototype will serve as the primary testing platform for this validation. During prototype development, the backend will be exercised through simulated user interactions from both student and instructor interfaces, measuring response times, data throughput, and stability under concurrent requests. This iterative testing process will provide early insight into real-world performance and guide refinements before full deployment.

3.4 Issue 4 - Interactive Visualization and Editing of Knowledge Graphs

Knowledge graphs are a powerful framework that enables the structuring and connectivity of data, allowing users to visualize and represent key relationships. However, the ability to make additions and deletions to the graph cannot be easily done through natural language queries, making the concept of graph manipulation difficult to overcome. Ideally, the user, regardless of experience, can make any changes to the relations or sets within the tree to either provide more context to the model or to update current information. Therefore, the process of being able to upload chosen documents that are necessary for our implementation, in addition to having the ability to easily maneuver the document throughout the graph, is are vital function of our solution. Within the graph view and database, the corresponding document will behave as a node within a relation and act as an additional source of context.

3.4.1 Desired Characteristics

- Ease of Use: The graph interpretation provides a user interface with a shallow learning curve, with capabilities to explore, expand, and modify the graph
- Dynamic: The UI generated from the framework allows users to interact with the graph in real-time actions with little to no latency.
- Maintainability: The software used to generate the graph interface is not prone to diminishing libraries and has minimal overhead
- Security: The chosen framework prohibits unauthorized access to the database through the UI interface
- Accessibility: The software does not require a considerable payment to host and utilize within our application, which poses an obstacle to production

3.4.2 Alternatives

- Neo4j Bloom:
    - A Neo4j-supported UI tool designed to be dynamic and explored through user-friendly operations such as natural language and visual queries. This tool, in particular, was introduced in 2018 and has several use cases for a variety of users, from inexperienced to professionals in the field. The tool incorporates the queries received from the user and translates them into the equivalent Cypher query.
- Linkurious Enterprise:
    - A built-in web application that allows users to collaborate with other users on a Neo4j graph. To use this application, a license must be acquired to use it freely, and costs vary from feature to feature. This application offers a graph user interface and analytics platform for massive connected databases and datasets that offer compatibility for several different database systems, offering a variety of database selections.
- KeyLines:

- A tool built a JavaScript SDK, providing more configuration to developers and freedom to implement their own tools. Cambridge Intelligence created the SDK for the purposes of visualizing and exploring graphs that communicate with a respective database through respective websockets or REST services. Unlike the other tools mentioned above, this application must be built from the ground up and tested thoroughly through our own applications; however, we as developers would be allowed more freedom in implementing our features the way envisioned by the team.

3.4.3 Analysis

- Neo4j Bloom
  - Ease of Use: The Bloom framework is user-friendly, as its core features involve ease of exploration, evaluation, and configuration of the graph representation with no Cypher query experience required of the user. Ideal for business users who just want to explore the tree without prior knowledge of Cypher.
  - Dynamic: Supports real-time querying and expansion directly from the graph database, additionally with features to support filtering, style, and simple pattern searches dynamically.
  - Maintainability: Very low overhead for our software solution, as we are planning to implement our system with the Neo4j database. All versioning and upgrades to the tool are managed by Neo4j. There is less flexibility to this tool as it ties directly to a Neo4j database environment.
  - Security: As part of the Neo4j environment, authentication is managed directly by the Neo4j software, as security is inherited directly from the database layer. Overall, the application is generally secure and managed directly from the database layer and can be improved upon within the backend application.
  - Accessibility: For an enterprise-level application and feature of this tool, prices vary per cluster and storage usage. For our project application, the free access version provides enough integration and features for a minimum viable product.
- Linkurious Enterprise
  - Ease of Use: Highly intuitive UI with natural language queries to the database directly. The software is designed to benefit both technical and non-technical users in its application, alongside collaborative efforts from multiple users on one graph. There is a steeper setup curve than the prior technology due to a broader feature tree offered by this application.
  - Dynamic: Strong analysis of nodes in real-time with the options to view, expand, and set up alerts within the tree. Visually adapts to broader and larger datasets than other technologies and supports real-time and scheduled alerting features.

    - Maintainability: Both modular and enterprise-ready applications that can integrate with a variety of databases, offering Docker deployments for such applications.
    - Security: Built with enterprise features in mind, and offers several role features, IDs, and granular permissions. Regulates effectively the controls of what users can and cannot do within the graph application view, along with audit trails governing feature changes.
    - Accessibility: Linkurious Enterprise offers solely enterprise-level plans that run $5.50 an hour for the cheaper plans, with no free application versioning.
- KeyLines
    - Ease of Use: A JavaScript SDK for building custom graph visualizations, allowing developers to make their tools as user-friendly as possible with flexibility. End users' entire experience with the tool will depend on how the SDK was developed by the team.
    -
    - Dynamic: The SDK provides an extremely dynamic implementation as animations, time-based filtering, real-time interactions, and customizations are supported by the framework. Ideal for highly interactive applications like the one we're developing.
    - Maintainability: Required dev-sourced updates, fixes, upgrades, and adaptations to potential requirements changes. Offers long-term flexibility for the team; however, the system's maintainability relies on group efforts to support our ideal integrations.
    - Security: The Keylines application is secure as its own tool; however, the security of our platform will depend on the features implemented by the team to develop as part of the expansive customization of the tool.
    - Cost: The KeyLines developers do not provide transparency in their application pricing; however, other sources list the product as $4,500 a year per license. For a more accurate evaluation of the product, the team needs to be contacted to receive an evaluation.

3.4.4 Chosen Approach

Framework	Ease of Use	Dynamic	Maintainability	Security	Cost
Neo4j Bloom	4	4	5	4	4
Linkous Enterprise	5	4	2	5	1
KeyLines	4	4	3	3	2

Based on the analysis done in the prior section, Neo4j Bloom would be the ideal tool for our application. It provides all the required characteristics, such as Ease of use, graph visualization with a dynamic property, maintainability of the resource, security of the application against false alterations to the graph, and cost of use to develop our product; no license purchasing is required for the software.

3.4.5 Proving Feasibility

Our preliminary analysis of the desired technology from above proves that Neo4j Bloom is our best choice due to the ease of integration with the Neo4j database. Current testing of a connection between Bloom and a prototype Neo4j database confirms compatibility and the featured no-language subqueries. We will expand our testing further by incorporating a larger dataset to evaluate the speed and effectiveness of the product and incorporate it into our project, where we will aim to illustrate the built-in effect UI and its capabilities to explore the graph to non-technical users.

3.5 Issue 5 - Chatbot Interface

A core integration of an AI agent is the frontend, as it defines how users will interact with the model that will ultimately shape the overall experience of the user. Due to this core component, selecting the best frontend framework in which we will integrate our UI is a vital aspect of the minimum viable product. The issue arises when selecting the appropriate framework job, as certain frameworks are more effective for certain use cases than others, in addition to quality-of-life features that can elevate the simplest of applications to more professional workflow environments.

3.5.1 Desired Characteristics

- Performance: Our frontend must be able to withstand the number of users coming to the site at any given time. Particularly, messages should be shown promptly by both the user and the AI agent, in addition to little to no latency when navigating through the webpage.
- Compatibility: Another core feature of the frontend is to be easily accommodable for users who are viewing the webpage from devices of varying screen scales, e.g, phone, tablet, computer, etc. As we can expect several users to come to our page for their requests, the frontend framework needs an adaptive layout for different devices.
- Maintainability: Lastly, our frontend framework must be able to support modularity and reusable/decoupled components to effectively and efficiently maintain our frontend systems for current and future developments.

3.5.2 Alternatives

- React:
    - React is a commonly used frontend JavaScript framework developed by Meta for the creation of dynamic webpages and offers a variety of UI libraries with a huge

support community. It also provides great integrations with APIs, utilizing a virtual DOM that keeps re-rending smoothly across live updates. Several big companies like Netflix and Uber use this framework to host their services.

- Vue:
    - Vue is designed for a user-friendly JavaScript framework that is both simple and powerful. The developer, Evan You, emphasizes Vue for its approachability and ease of integration into software platforms, providing an easier stepping stone for developers in dynamic web development.
- Next:
    - Next.js is built upon the React framework that behaves more like a full-stack system, which essentially performs the same, if not better than, the React framework. Next excels in the area of real-streaming, API communication, and scalable deployment.

3.5.3 Analysis

- React
    - Performance: React offers high performance through its virtual DOM. The virtual DOM solely updates components that change and does not regenerate adjacent elements that have not been modified by the user. An application like this is ideal, as the chat log the user will have with the AI agent will constantly be changing, and it reduces overhead by not reloading the entirety of the components.
    - Compatibility: The React ecosystem includes its React Native tool, allowing developers to reuse components and logic and format it for respective media with a potential for web app development.
    - Maintainability: React strongly supports component-based architecture, making the code modular and easier to maintain throughout the software lifecycle. In addition, the presence of a strong community support and constant updates from a high-end company like Meta ensures long-term stability.
- Vue
    - Performance: Vue.js is well known for being a lightweight and fast application, suiting the ideal components of our project. With reactive data-binding and very minimal overhead, this application would work effectively as users navigate the page. However, there are limitations in dynamic responses and their toll on the overall performance of the frontend due to high stress.
    - Compatibility: Vue's internal frameworks, such as Vuetify and Quasar, help create mobile-friendly interfaces for web page applications with decent potential for mobile app development or integrations.
    - Maintainability: Vue utilizes a more single-file modularity system that can be easily maintained with clear documentation on how features of the frontend

features behave. The ecosystem is very stable and the core API functionality rarely changes, ensuring its long-term stability.

- Next
    - Performance: Next.js performs very efficiently and can incorporate all the features and behaviors of React with its own specialized features. These features include server-side rendering, static site generation, and script optimizations, which improve load times and overall responsiveness of the application.
    - Compatibility: Since Next.js is built upon the React ecosystem, the application includes its React Native tool, allowing developers to reuse components and logic and format it for respective media with a potential for web app development.
    - Maintainability: Next.js offers modularity and reusability to the same scale that React handles these operations, in addition to holding several conventions within the framework that would naturally require third-party software to operate, such as routing and bundling.

3.5.4 Chosen Approach

Framework \ Characteristic	Performance	Compatibility	Maintainability
React	4	5	4
Vue	3	4	4
Next	5	5	5

From the table above, Next.js is by far the most desired approach as it incorporates and builds upon the React framework and provides excellent performance and maintainability for our desired implementation. Moreover, Next.js's features allow for a more enterprise-ready implementation, allowing the extension of our project to a broader and more demanding network and environment. Vue and React would both be easy to implement into our systems; however, much more work and external dependencies would be required of our proposed tech stack.

3.5.5 Proving Feasibility

We have tested a slim model of Next.js utilizing its app route handler feature to proxy a small backend Python FastAPI service. Once we move the integration into our products, we will begin to implement more of its core advanced features, like token streaming. The team has experience with React services, and utilizing this newly built framework is very similar to React, allowing for a shallow learning curve. Additionally, with the framework integrated in our project, we aim to test the bounds of dynamic real-time chats with an OpenAI model with history, where we will version and improve upon its capabilities.

3.6 Issue 6 - Database

For the AI tutoring agent to be effective, it must be capable of maintaining a conversational interaction with users. Essentially, our model must be able to implicitly reference the context of previous messages that were exchanged, and have the agent recognize this and generate responses using this context. This requires the LLM to use conversational memory techniques, providing the model with the conversation history for each new prompt. However, this is where the technological challenge lies: it would be inefficient and insecure to store full conversations in our system's main memory or to require that full conversations be sent from the frontend to the backend of our application each time a new query is made. Thus, we will need a database framework to securely store this information such that our backend can easily locate where to retrieve the appropriate conversational context for a given user's query. This would have the additional benefit of allowing our system to store different "chats" for users to revisit without losing their past conversations.

3.6.1 Desired Characteristics

- Speed: As mentioned with previous issues, the overall responsiveness of the LLM is very important for our system. So, since nearly every query will require the context of the conversation to be retrieved from this database, it is vital that querying the database and retrieving the large amounts of information can be done quickly.
- Compatibility: The database should be able to be incorporated easily into Python backend applications and work well for storing messages specifically. For the ease of maintenance of our system, it should be uncomplicated to implement the querying and storage functionalities into our application so that it can be understandable and simple to test. It should also be compatible with storing the specific data required for our project, which is user and message data. Ideally, to support this, it should be both relational and capable of querying and storing threads of information.
- Security: Although it is not expected that users will be entering confidential information into our system, it is still important that users feel safe entering any amount of information into our system, such that it cannot be maliciously accessed by attackers. So, the database should be fully secure from allowing any user data to be accessed by any other users or attackers.
- Scalability: As the number of users grows, the amount of chats that will need to be stored will also increase substantially, so our chosen database framework must be able to maintain high performance through built-in optimization techniques, especially since chat data is particularly high in volume.

3.6.2 Alternatives

- SQLite
    - A lightweight, local database that runs within the application. It is typically used in development and lightweight applications because setup is extremely simple

and requires very few resources. It offers full SQL support using only a single file.

- PostgreSQL
    - A powerful, state-of-the-art database system known for its security, scalability, and robustness. It offers several advanced features and capabilities that go beyond standard SQL, such as JSON storage, full-text search, and complex queries. It is a popular database choice for several production applications.
- MongoDB
    - Unlike the previous alternatives mentioned, this database framework is document-oriented and does not use SQL. It stores data as JSON documents organized in collections. It is known to be excellent at handling large amounts of semi-structured data and easy to scale across multiple servers, making it a popular choice for applications that require speed and flexibility.

3.6.3 Analysis

- SQLite
    - Speed: Single-user queries and local operations are very fast since there is virtually no overhead. However, performance tends to decline as the user base size and query complexity increase.
    - Compatibility: There is a built-in Python module that supports SQLite operations, which makes it very easy to set up and start using with our application. But it is not ideal for storing large message chains since performance drops as its file size grows. So, if multiple users were using the AI tutoring agent at the same time, the performance would significantly decrease since each query requires a large amount of text to be read and written to the database.
    - Security: The simplicity of this framework comes with a lack of security. It has very little built-in security, relying mostly on user file permissions and lacking encryption.
    - Scalability: As previously mentioned, SQLite uses a single file as the database, so it is meant to be deployed on local systems and not scaled past that.
- PostgreSQL
    - Speed: It is very efficient for complex queries and larger operations, which our application will most likely require, given the volume of data that needs to be stored and retrieved.
    - Compatibility: The complex query abilities and data size flexibility would make this a good option for storing and querying messages and user data. Furthermore, there are several pre-existing Python modules that would make utilizing PostgreSQL in our application functions very straightforward, such as SQLAlchemy.

    - Security: This framework is known for its robust, out-of-the-box security. It offers several features such as network security, user permissions, data monitoring, and data encryption.
    - Scalability: As mentioned before, this database framework is known for its high-volume data performance capabilities. It is also simple to add more resources to a PostgreSQL server or distribute work across multiple servers using pre-existing extensions.
- MongoDB
    - Speed: It is extremely fast for simple document reading and writing. But it can be slow for more complex queries and multiple-document retrieval, since scanning is required for each lookup, which can be inefficient.
    - Compatibility: Like the previous alternatives mentioned, there exist Python modules that would make MongoDB simple to implement into our system. As for overall compatibility with our data, it excels at storing long, unstructured chat logs, but the lack of relationality would cause a lot of repeated information.
    - Security: Despite having a history of being insecure in the past, modern-day MongoDB includes several of the same security features as PostgreSQL, such as network security, user permissions, data monitoring, and data encryption. So, with proper configuration, this framework would suffice for the security needs of our application.
    - Scalability: It is also designed to be highly scalable and easily distributed across multiple servers for optimal performance as the amount of data increases. Furthermore, since MongoDB allows fora flexible document schema, it is simple to adapt the database as data inevitably evolves.

3.6.4 Chosen Approach

Framework	Speed	Compatibility	Security	Scalability
SQLite	4	3	2	1
PostgreSQL	5	4	5	4
MongoDB	3	3	5	5

Based on this analysis, we decided that PostgreSQL would be the best database for our application. Although SQLite would be easy to deploy, the security and scalability issues would be a major fault for our system, since we would want our user base to feel safe using the product, and the amount of storage needed for individual chat logs can potentially be very large. Additionally, MongoDB's lack of relationality makes it less ideal for our system despite its

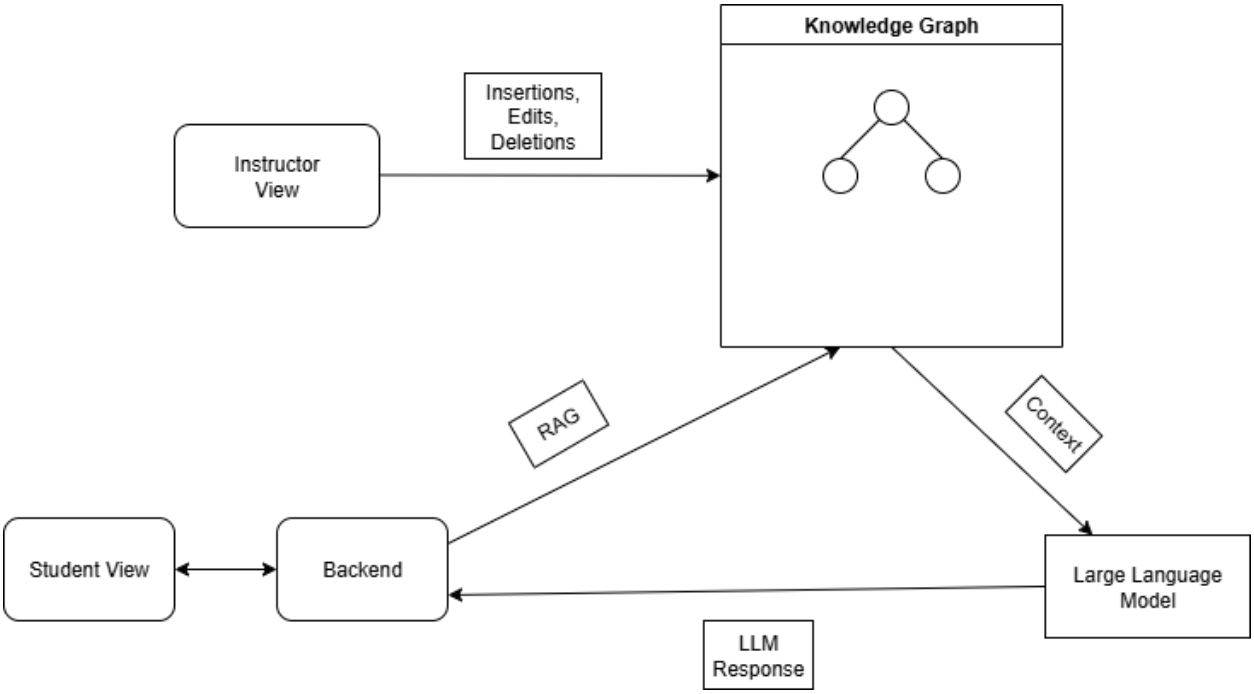
various advantages. Since we ultimately want to relate users to chats and chats to chatlogs in our database, an SQL approach is believed to be more effective, especially since we do not expect our data to be largely unstructured. Thus, PostgreSQL offers the speed, compatibility, security, and scalability that are required for our system without any fundamental downsides for our system.

3.6.5 Proving Feasibility

We have researched and created small deployments of each alternative to come to this decision, but moving forward, we will need to analyse the effectiveness of PostgreSQL once integrated into the prototype of our full system. In the initial steps, we will conduct further analysis of the speed and compatibility of the database framework through observing the ease of maintenance in our system and testing for performance deviations of the LLM once the database is implemented in the pipeline. Unexpected performance issues with this framework may arise in the true environment of our system, so these steps would help validate our choice.

4. Technology Integration

Considering the requirements and design implementations discussed above, our proposed system has several layers of complexity. To provide a simpler and easily digestible view of our proposed system, we have laid out the following diagram, which can be found below. This diagram displays a high-level view of our technological implementation that will serve as the skeleton for which the project will follow.



Each of the views listed on the diagram on the left-hand side is a React-based frontend service that will be shown to the users. The student view will be the default display that appears similar

to chat sessions with LLMs, and the instructor view can be accessed with the right credentials in a user session. From the instructor's view, the user can make any necessary changes to the knowledge base through Neo4j's Bloom tooling and keep the context up to date for the course of the class. The main use case for this project will reside in the default student view, where they will make requests through a chat box in natural language, similar to how one would prompt an LLM. Their request will get sent to the backend, which will retrieve relevant information from the knowledge graph and provide the content to our LLM. Additional prompting will be included within the backend service to improve upon guardrails, ensuring that the response is as expected and referenced before returning to the user.

5. Conclusion

This project was initiated to resolve a critical issue within cybersecurity education by developing an AI tutoring agent capable of guiding students through complex static analysis Capture the Flag (CTF) challenges. Existing AI tools lack the contextual precision, adaptability, and domain understanding required for teaching these concepts. This analysis examines the manner in which the proposed system addresses this gap. By integrating a structured Neo4j knowledge graph with a large language model (LLM), it enables contextually grounded, adaptive tutoring that enhances both learning outcomes and instructional efficiency.

Throughout this technological feasibility analysis, we examined the project's key technological challenges: knowledge graph management, LLM integration, backend framework selection, and system interoperability. Challenges are evaluated against defined criteria, including performance, maintainability, scalability, and compatibility with the broader Python-based architecture and current expected tech stack. The technical foundation for this project is expected to look like this: Neo4j for graph data modeling, GraphRAG for contextual retrieval, FastAPI (Python) for backend orchestration, Neo4j Bloom for intuitive visualization and editing, Next.js for chatbot integration, and PostgreSQL for application and chat storage. Together, these components form a scalable and maintainable infrastructure tailored to the needs of AI-driven tutoring for instructors and students.

The selected design choices are expected to provide efficient coordination between the ReactJS frontend, the Neo4j knowledge graph, and the OpenAI large language model via LangChain. Containerization with Docker is anticipated to enable consistent environment replication, simplified dependency management, and seamless deployment across development, testing, and production, ensuring that components run reliably and predictably in every setting. Collectively, these technological decisions establish a feasible and maintainable foundation for continued development on this project.

Looking ahead, the project will advance by formalizing the requirements for backend orchestration, knowledge graph integration, and LLM retrieval to ensure alignment with the

defined technical criteria. Prototyping efforts will focus on refining the FastAPI backend, extending the GraphRAG retrieval pipeline to accommodate complex knowledge graph structures, and enhancing the instructor-facing interface for intuitive editing and teaching. These next steps will establish the foundation for a fully integrated AI tutoring assistant, enabling reliable, contextually informed responses while maintaining an extensible system architecture suited for iterative development and eventual production deployment.