

Developing a Socratic Artificial Intelligence Tutor Using Retrieval Augmented Generation and Pedagogical Prompting

Micah Okpara¹; Blessing Chimezie Uzo^{1*}; Ebere Nwanneka Chukwuemeka²;
Ponsak Stephen Bande¹; Fortune Nkem Okpara³

¹Department of Computer Science, Faculty of Physical Sciences, University of Nigeria, Nsukka,
Enugu State, Nigeria

²Department of Computer Science Education, Federal College of Education, Eha-Amufu,
Enugu State, Nigeria

³Department of Software Engineering, Federal University of Technology Owerri, Owerri,
Imo State, Nigeria

Corresponding author: Blessing Chimezie Uzo^{1}

Publication Date: 2026/10/02

Abstract: Generative artificial intelligence (AI) applications such as ChatGPT present a distinct challenge for computer science (CS) students. While they offer individualized learning support, unregulated use risks facilitating academic shortcuts and eroding critical thinking skills. This article introduces Lion-AI, a chatbot tutor developed to augment the CS educational experience. Lion-AI integrates retrieval-augmented generation (RAG), rooted in university-specific course materials, with an engineered Socratic interaction policy. Unlike general-use tools, Lion-AI is designed to classify student requests, redirect direct-solution requests through diagnostic questioning, and provide guidance through one-question-at-a-time interaction. The study describes a method for creating ethically constrained AI tutors in CS education. It presents the system's three-tier architecture, the RAG pipeline, and the transparent interaction policy behind the tutor, positioning Lion-AI as a deployable proof of concept for combining curriculum grounding with explicit Socratic constraints. A controlled classroom study is planned as the next step.

Keywords: Artificial Intelligence in Education (AIED); Large Language Models (LLMs); Retrieval-Augmented Generation (RAG); Prompt Engineering; Computer Science Education; Socratic Method.

How to Cite: Micah Okpara; Blessing Chimezie Uzo; Ebere Nwanneka Chukwuemeka; Ponsak Stephen Bande; Fortune Nkem Okpara (2026) Developing a Socratic Artificial Intelligence Tutor Using Retrieval Augmented Generation and Pedagogical Prompting. *International Journal of Innovative Science and Research Technology*, 11(9), 2369-2379.
<https://doi.org/10.38124/ijisrt/26sep959>

I. INTRODUCTION

Public access to generative artificial intelligence (AI) changed substantially when OpenAI introduced ChatGPT in late 2022. Large language models (LLMs) can generate human-like text and functional code, and their use has become relevant in many educational contexts [1], [2]. In computer science education, the emergence of generative AI raises an interesting paradox: on the one hand, AI provides learners with personalized support and explanations; on the other hand, it creates serious ethical dilemmas regarding academic integrity and skill development [3], [4]. Research findings suggest that uncontrolled use of general-purpose AI tools can encourage low-order or surface-level learning, particularly in introductory programming courses [5], [6].

The use of general-purpose tools by students can enable them to skip debugging and other difficult tasks that help them develop programming skills. Nevertheless, it is not an option to ban AI altogether, as students must be proficient in using AI in their future profession [7]. The key question, therefore, is not whether students should use AI but how their use of AI can be structured to support learning.

The problem is, however, that refusal does not mean teaching. A tutor who refuses to assist leaves the student's confusion unresolved, whereas a tutor that answers every question takes away the productive struggle through which understanding develops. Good tutoring is something in between. It identifies the learner's current understanding, formulates a targeted question, gives a minimal but useful

hint, and gradually reduces support as the learner progresses. This is the tradition of Socratic tutoring, in which questioning rather than answering is the primary instructional act [8], [9].

This article describes Lion-AI, a custom-built AI chatbot tutor for CS education. Lion-AI combines two technical methods. The first is retrieval-augmented generation (RAG), which grounds responses in a curated knowledge base of CS course materials from the target institution. The second is pedagogical prompt engineering, organized as an explicit interaction policy rather than as a single instruction to “avoid giving answers.” This article makes three contributions. First, it documents Lion-AI as a curriculum-grounded Socratic tutor with request classification, a graduated support ladder, questioning strategies, and academic-integrity constraints. Second, it presents a three-tier architecture showing how RAG and the interaction policy are combined in a deployable prototype. Third, it articulates the challenges of enforcing Socratic guardrails through prompting alone, providing a transparent description of the interaction policy for future empirical validation. The work should therefore be read as a design and implementation report; a controlled classroom study of learning outcomes remains the necessary next step.

II. RELATED WORK

Following the development of fully generative models, research on the application of AI technology within learning environments has increased significantly, including studies on opportunities and risks in computer science education.

➤ *AI Tutors in Computing Education*

LLMs have rapidly become part of the computing education landscape. The advantages of ChatGPT, Gemini, Copilot, and similar programs are utilized by students when they want to learn programming concepts, understand error codes and their meanings, create programming examples, get help debugging their programs, and even find complete solutions to their problems [3], [10]. Thus, AI tools can provide immediate support when a teacher or teaching assistant is unavailable [3], [10]. The emergence of GPT technology has prompted several studies. Becker et al. [3] point out possibilities and risks that AI systems create for education. Lau and Guo [4] show that programming instructors expect to redesign their courses as students increasingly use AI-generated code and explanations. Leinonen et al. [6] show that LLMs can improve programming error messages, while Denny et al. [5] describe how students use conversational AI for debugging, conceptual questions, and solution generation.

Recent reviews confirm the maturity of the field. Denny et al. [10] argue that computing educators must reconsider help-seeking, programming assignments, and assessment when LLMs can generate plausible code. Raihan et al. [11] review LLM use across CS and computer engineering education. Tabarsi et al. [12] review 52 student-facing LLM tools in undergraduate computing education and find that prompt engineering, multi-stage pipelines, guardrails, scaffolding, and problem-based learning are now common. The question for computing educators is therefore no longer

whether students will use AI, but how AI assistance should be designed so that it supports, rather than replaces, reasoning, debugging, decomposition, testing, and reflection.

➤ *Challenges with General-Purpose LLM Tutors*

General-purpose LLMs are designed to respond helpfully and efficiently. In programming education, this can be problematic. A student can request a complete assignment solution, and the system may provide one without requiring the student to explain reasoning, identify a bug, test an approach, or reflect on an error. Although this may help with task completion, it can reduce engagement with the cognitive processes programming education is meant to develop [3], [5]. Kasneci et al. [7] identify hallucination, bias, plagiarism, over-reliance, and lack of transparency as risks of educational LLM use. Rudolph et al. [2] argue that educators should redesign tasks and tools to support pedagogical goals rather than simply prohibit AI. Valle Torre et al. [13] show that unguided LLM use can encourage executive help-seeking, where students seek answers rather than understanding, and that such patterns do not self-correct without intervention.

Even when LLMs are prompted to behave as tutors, their behavior may not consistently support learning. Cardell-Oliver et al. [14] found that LLM-led Socratic dialogues could scaffold critical thinking, but also observed unfounded praise and reformulation of student answers rather than guidance toward self-correction. Furthermore, Crompton and Burke [15] note that the majority of AI educational applications are still in preliminary stages and do not sufficiently address ethical, pedagogical, and contextual considerations. These findings suggest that simply refusing to give a complete answer is not enough. A useful AI tutor must also diagnose understanding, select appropriate questions or hints, respond to misconceptions, and adapt when the learner remains stuck.

➤ *Retrieval-Augmented Generation and Pedagogical Prompting*

RAG improves the relevance and factual grounding of LLM responses by retrieving relevant passages from an external document collection and providing them as context to the model [16]. RAG has been refined for domain-specific settings and can reduce hallucinations by grounding generation in external sources [17]. Prompt engineering similarly shapes LLM behavior, including by encouraging step-by-step reasoning [18] and more process-oriented explanations [7].

However, RAG is now established in computing education rather than novel. Yu et al. [19] describe a locally hosted small language model combined with RAG in a large computing course with more than 250 students and nearly 2,000 student questions. Tabarsi et al. [12] likewise identify retrieval grounding and prompt guardrails as recurring components of student-facing LLM tools. The implication for Lion-AI is that retrieval answers what the tutor should know about a course, but not how the tutor should teach. A retrieval-grounded tutor can still provide complete solutions or poor scaffolding. Lion-AI therefore treats RAG as curriculum-grounding infrastructure and locates its

pedagogical contribution in the interaction policy layered on top of it.

➤ *Socratic Pedagogy in Intelligent Tutoring*

The Socratic method is based on guided inquiry rather than direct transmission of answers. A tutor asks questions intended to reveal the learner's current understanding, identify misconceptions, and support the learner in reasoning toward a conclusion. Stevens and Collins [8] describe Socratic tutoring as a goal-directed process involving diagnosis and correction of gaps in the learner's knowledge. Research in programming education provides evidence that this method can support code comprehension and learning. For instance, Tamang et al. [9] compared free self-explanation with Socratic tutoring for source-code comprehension and reported higher learning gains in the Socratic tutoring condition. Similarly, Oli et al. [20] found that scaffolded self-explanation supported by an intelligent tutoring system was particularly beneficial for learners with lower prior knowledge. These studies demonstrate that withholding an answer, providing a hint, and conducting a Socratic dialogue are not the same thing.

More recent LLM-based systems attempt to apply these concepts. STAP, described by Xie et al. [21], is a Socratic tutor which analyzes student code through the processes of checking, correcting, complementing, and segmenting. Thong and Thakkar [22] track whether misconceptions are introduced, reinforced, weakened, or resolved across dialogue turns. According to Mao et al. [23], students need help to respond appropriately to Socratic questions. Together with the findings of Cardell-Oliver et al. [14], this work indicates that instructing an LLM to “be Socratic” is not sufficient; the intended behavior must be specified as an interaction policy and verified against actual dialogue.

➤ *Existing Course-Specific and Socratic Systems*

Course-specific AI assistants are increasingly common. The creation of a customized AI tool “CodeHelp” by Liffiton et al. [24] illustrates one example of an AI-based system that can provide scalable support in programming education. This system shares similar goals to the “CS50 Duck” developed by Harvard [25] in how they have both incorporated context-specific information into their respective systems along with implementing instructional protocols that guide users away from producing complete coded solutions and instead toward generating hints and explanations related to coding concepts.

Cogniti, developed at the University of Sydney, allows educators to create custom AI agents supplied with course resources and embedded in Canvas [26], [27]. In one reported deployment, Cogniti was configured as a Socratic tutor for a first-year musicology course and generated 251 student conversations over five weeks [26]. Bruno [28] combines RAG over course notes with no-code guidance, while Saksham AI [29] provides Socratic hints and adaptive feedback to engineering students in India. On the other hand, earlier studies on virtual teaching assistants (VTAs), such as the KNUST-Bot in Ghana [30], showed that, even if the systems utilized are rule-based chatbots, these applications can improve student performance through providing access to course information in a timely fashion. However, these VTAs do not have the level of flexibility or conversational capabilities as those systems built on LLMs.

Table 1 summarizes selected systems relevant to Lion-AI. Its purpose is to make explicit that course-specific AI tutoring, retrieval grounding, answer-limiting guardrails, and Socratic prompting are established design directions rather than novel contributions.

Table 1 Selected Course-Specific and Socratic AI Tutoring Systems

System	Context	Course grounding	Pedagogical approach	Evaluation
CodeHelp [24]	Programming	Course and assignment context	Guardrails against solution disclosure	52 students, 12 weeks
CS50 Duck [25]	Introductory CS	CS50 course materials	Course-aligned AI support	Large course context
Cogniti [26], [27]	Multiple disciplines	Educator-supplied LMS resources	Configurable Socratic agent	251 conversations
Bruno [28]	Programming	RAG over course notes	Guided, no-code support	75-student user study
STAP [21]	Programming	Student code/task analysis	Socratic scaffolding pipeline	Formative evaluation
Saksham AI [29]	Coding education	Context-aware support	Socratic hints and feedback	1,170 registered users
Lion-AI	CS education	RAG over institutional materials	Request classification, graduated support, one question per turn	Design and implementation report

➤ *Positioning Lion-AI and Remaining Gap*

Existing research shows that curriculum grounding, answer-limiting guardrails, and Socratic interaction can each support responsible AI use in computing education. Lion-AI is therefore not presented as the first course-specific LLM tutor, the first RAG-based programming assistant, or the first Socratic AI tutor. Its contribution is narrower: it documents how retrieval, request classification, graduated assistance, and turn-by-turn questioning were combined in a CS tutoring prototype. This contribution is relevant because many systems

state that they use Socratic methods or withhold complete answers without fully explaining how they identify request types, determine the appropriate level of assistance, select questioning strategies, or respond to stuck or frustrated learners. Lion-AI provides a transparent description of this interaction policy and its technical implementation. The present work should not be interpreted as evidence that Lion-AI improves learning outcomes; that question requires a controlled classroom study.

III. SYSTEM DESIGN AND ARCHITECTURE

The development of Lion-AI is based on the design science approach, which emphasizes the creation of artifacts to solve real-world problems [31]. The system was built in a modular structure to support pedagogical control and technical scalability. The core design principles, three-tier architecture, RAG pipeline, and Socratic interaction policy are described below.

➤ Design Principles

There are five main design principles that drove the development of Lion-AI based on previous findings related to the role of AI in education and intelligent tutoring systems [32], [33].

- **Pedagogical Alignment:** The first principle is that Lion-AI's pedagogical model must align with accepted learning theories so that students can reach their own conclusions. Therefore, Lion-AI is designed to lead the student through a process of guided inquiry rather than direct answer provision [34].
- **Contextual Relevance:** The need to align with contextual relevance means that the information that Lion-AI provides to a user will be based on the particular curriculum that exists at the school or institution where the user is studying. This is accomplished by using RAG, as it allows the system to access resources relevant to the user's course before generating an answer [16].
- **Academic Integrity:** The system will actively discourage academic misconduct through the use of ethical guardrails that prohibit the system from providing complete code or answers to "homework-type" questions. The use of constraint-based prompt engineering is the basis for these ethical guardrails [7].
- **Modularity and Expansion:** The system's architecture and design must also accommodate future growth and expansion, including the ability to integrate multi-modal inputs (e.g., code snippets, diagrams, etc.) into the design of the system and the integration of Lion-AI with Learning Management Systems (LMS). The modularized nature of Lion-AI permits all of the components of the RAG pipeline, LLM backend, etc., to be updated separately. Therefore, an updated RAG pipeline will not necessitate an update to the LLM backend and vice versa [35].
- **Graduated Assistance:** Withholding assistance is not, by itself, pedagogically useful. A tutor that only refuses may push a student toward an unconstrained general-purpose model. Lion-AI is therefore designed to provide the least direct assistance that can move the student forward without giving away the final answer.

➤ Three-Tier Architecture

Lion-AI is a web application utilizing a three-tier architecture [35]. The separation of concerns enhances the security, scalability, and maintainability of the application. The front end, which is the presentation tier, contains all user interactions and visual feedback to users. The front end is built with HTML5, CSS3, and JavaScript in combination with the React.js framework. The front end has a clean and responsive chat interface that allows users to input text and receive AI-generated responses formatted as conversations. When users submit input, they receive a real-time status indicator while the response is being generated and can see the history of their conversations via session management. The minimalist design of the front end is intentional and serves to reduce the cognitive load on users and provide a focus on the learning task [36].

The middle tier contains all business logic and the AI-backed processing and serves as the bridge between the user interface and the backend sources of data. The middle tier is comprised of two components: the web backend and the AI backend. The web backend uses Node.js and Express.js to handle incoming HTTP requests from users, authenticate users, and manage user sessions. The web backend serves as the entry point for the AI backend. The AI backend is written in Python and is responsible for handling queries sent by the web backend, ensuring that user queries conform to the requirements of the computer science curriculum being studied using the RAG pipeline and the large language model (Gemini API). Before a query is processed, the AI backend performs input sanitization: it redacts obvious personally identifiable information such as email addresses and phone numbers, screens for unsafe content, and applies custom validators intended to reduce the effectiveness of prompt-injection attempts. It then applies the system prompt and pedagogical constraints to the query in order to generate the final output returned to the web backend. Since the web backend and AI backend are implemented separately from one another, it is possible to continue to upgrade the RAG and LLM components without any changes to the rest of the logic in the web frontend.

All permanent data used by Lion-AI is stored in a data tier. The data tier includes a structured relational database (PostgreSQL) for storing user accounts, conversation and message records, and system metadata. Additionally, the data tier includes a vector database that contains embedded representations of the institution's computer science course materials and is used to provide the AI with an efficient method of searching for relevant content during the RAG process. The use of a vector database to store embedded content during the RAG process is well documented in research literature [16], [17] as an approach to increasing the factual accuracy and relevance of LLM outputs. Fig. 1 provides a high-level overview of the three-tier architecture and the data flow between components.

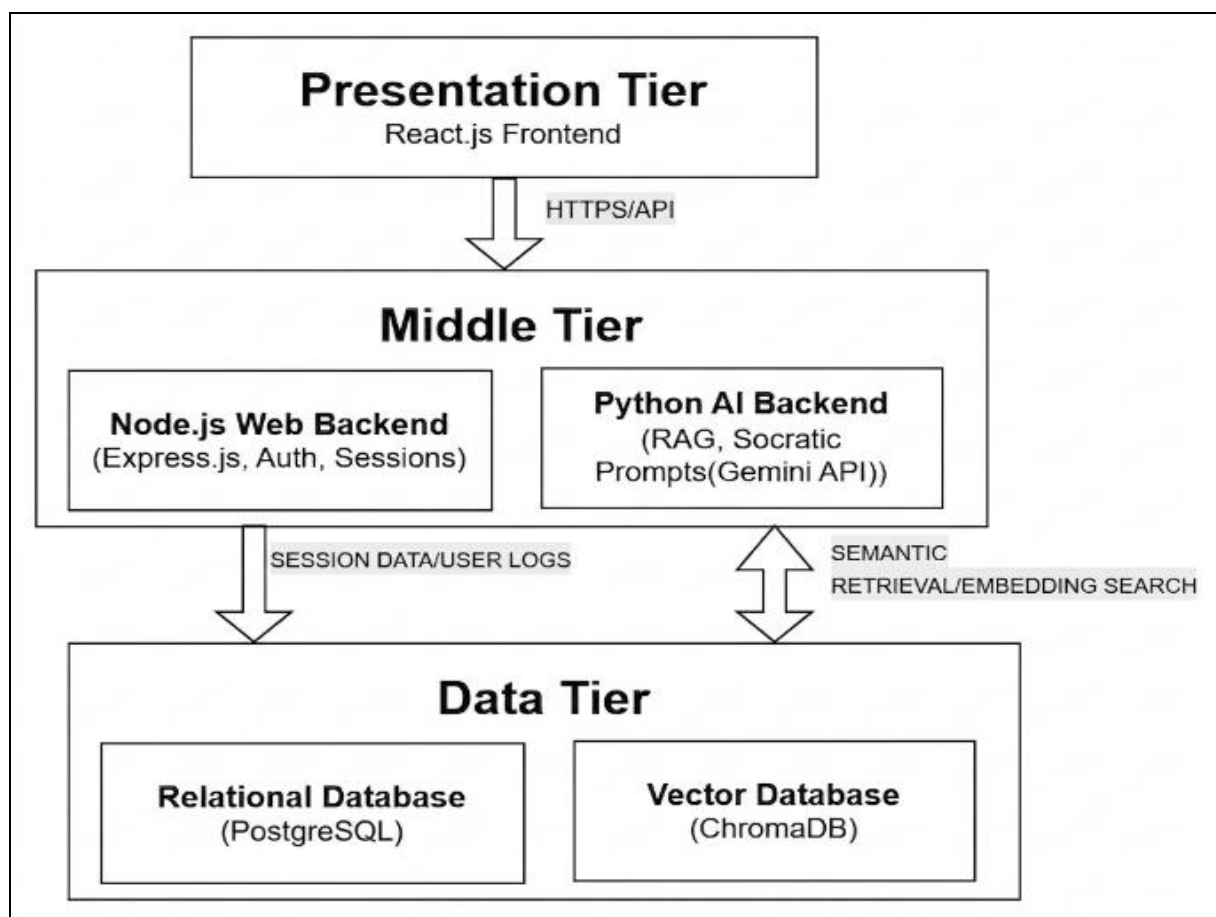


Fig 1 Three-Tier Architecture of Lion-AI, Illustrating the Flow of Data Between the Presentation, Middle, and Data Tiers.

➤ Retrieval-Augmented Generation (RAG) Pipeline

The RAG pipeline is a key aspect of Lion-AI since it ensures that responses can be grounded in content taken from authoritative sources that are relevant to the courses. The RAG pipeline is broken down into three main steps: document ingestion and preprocessing; query processing and retrieval; and response generation.

In the deployed prototype the knowledge base consists of 32 PDF documents drawn from departmental lecture notes and textbook-derived materials spanning first-year to final-year courses, with a total size of approximately 31.8 MB. Pages with fewer than 25 characters are discarded as noise. The remaining text is segmented into 800-character chunks with 80-character overlap, split primarily on newline boundaries. Chunks are embedded using BAAI/bge-small-en-v1.5 via FastEmbed and stored in ChromaDB, persisted to disk. The use of sentence embeddings supports semantic similarity search [37]. Each incoming query is first normalized and classified into one of three routes: out-of-scope refusal, direct generation, or RAG-routed response. Only the third category triggers vector search. For retrieval, ChromaDB returns the top three chunks ($k = 3$), and a relevance threshold of 0.35 is applied. If the best chunk falls below this threshold, the system avoids injecting weak context and instead answers without retrieved context while indicating that the course materials do not appear to cover the topic. This query-processing and retrieval workflow is summarized in Fig. 2.

Response generation occurs with the combination of a user query and the top retrieved document chunks from ChromaDB. This forms a single prompt that is provided to the language model (Gemini 2.5 Flash in the deployed version) for response generation. Explicit instructions are provided in the prompt for using these retrieved materials as context and to be cautious about generating complete solutions (described in Section III-D) so that the generated response draws on the materials retrieved and scaffolds the student's understanding of the topic. The prompt further instructs the model to disclose when it is drawing on retrieved course material and when it is not. The RAG pipeline is built using the LangChain framework [38], which provides modular components for loading, embedding, retrieving and creating prompts for generating responses. The deployed prototype used LangChain 1.2.0 and ChromaDB 1.4.0.

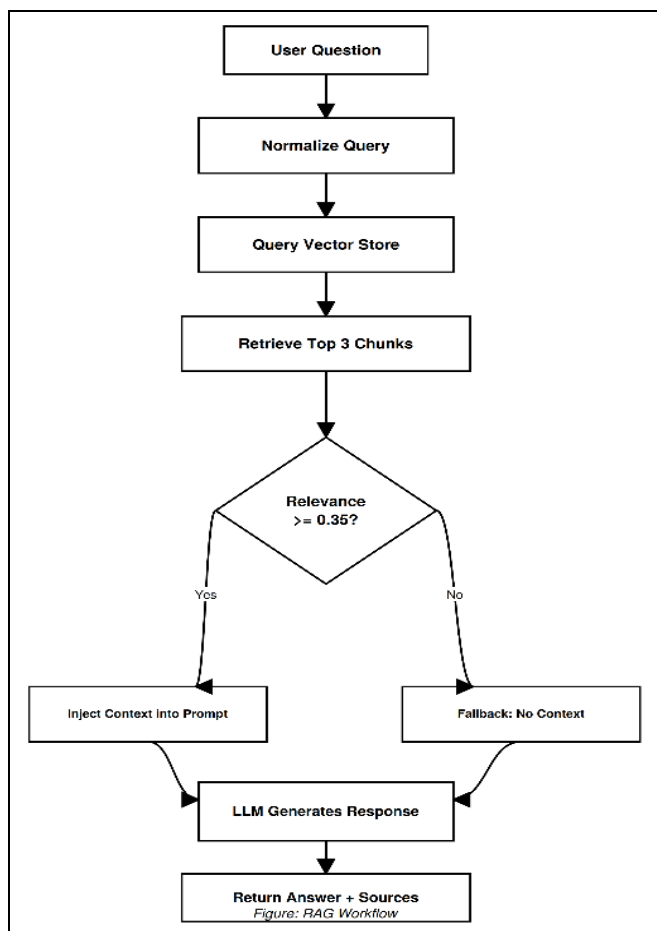


Fig 2 Query-Processing and Retrieval Workflow, Showing Route Classification, Top-Three Retrieval, and the 0.35 Relevance Threshold.

➤ Socratic Interaction Policy and Pedagogical Prompting

Prompt engineering involves developing the prompt, or “input prompt,” that is used with an LLM [18]. Lion-AI uses prompt design to help implement the Socratic tutoring paradigm and ensure the system does not give students direct answers. The system prompt is an instructional framework that is prepended to every student request when sent to the LLM. The prompt begins by defining the role of the LLM, where the LLM acts as a Socratic tutor for computer science students. For example: “You are Lion-AI, a Socratic tutor designed to help computer science students at our institution learn programming and CS concepts. Your goal is to guide students toward understanding, not to provide direct answers or complete code.”

Beyond this role definition, an instruction to “be Socratic” is not sufficient to produce Socratic behavior, as the studies discussed in Section II-D make clear. Lion-AI therefore encodes an explicit interaction policy. A single unified prompt handles every topic; behavior changes with the state of the conversation rather than by substituting a different template for each course. The policy has five components:

- Request classification: The tutor classifies each student message into one of five categories, which determines the response mode (Table 2).
-

Table 2 Request Classification and Corresponding Tutor Response Mode

Request class	Typical signal	Tutor response mode
Direct-answer request	"Give me the code for..."	Decline the complete solution, then ask a diagnostic question
Genuine confusion	"I don't understand..."	Explain in plain language, then ask a comprehension check
Partial reasoning	Student supplies code or a partial explanation	Respond to the attempt, identify the gap, ask a targeted question
Off-topic	Question outside computing or technology	Decline politely, state scope, redirect
Frustration	Repeated failure or exasperation	Acknowledge difficulty, move up the ladder immediately

- Graduated support ladder: Rather than applying a single refusal rule, the tutor escalates assistance as the student demonstrates continued difficulty (Table 3). The ladder terminates before the complete solution.

Table 3 Graduated Support Ladder

Level	Assistance provided
1	Ask what the student already understands
2	Point to the relevant concept without applying it
3	Decompose the problem and ask which part is unclear
4	Present an analogous worked example
5	Give one partial step, labeled as partial, never the final answer

- Questioning strategies: The tutor selects among six strategies: decomposition, analogy, counterexample, edge-case reasoning, self-explanation, and prediction. The tutor asks one question per turn.
- Academic-integrity constraints: According to the intended policy, no complete solution should be supplied for work the student indicates is graded. Where a student states that the task is for practice or revision, the policy permits more direct explanation. The system cannot independently

verify such a claim and relies on the student's account. The constraint is therefore a friction mechanism rather than an enforcement mechanism. Its consistency in practice requires systematic evaluation.

- **Presentation and localization:** Explanations are pitched at an accessible reading level. The tutor draws on locally familiar examples and interprets local terminology correctly.

Lastly, to ensure that the LLM considers the documents retrieved through the use of RAG, the LLM is given an explicit directive in the prompt: "Decide what information to use for your answer based on the following context. You should not use the context if it has no relationship to your answer: {context}." Representative excerpts include: "I'm here to help you think, not to write it for you. Let's break this down. What is the first step you think we should take?" and "If the student is still stuck, give one partial worked step, never the final answer, and say clearly that's what you're doing." The complete system prompt is dynamically constructed to combine the user request and retrieved RAG context and is provided to the LLM. The extent to which the deployed prototype adheres to this policy in practice is the subject of the planned classroom study.

➤ User Interface Design

The user interface for Lion-AI is designed with simplicity, ease of use, and a focus on the learning task. Interactions are primarily conducted through a text message exchange, which is natural for students and reflects the

conversational style of LLMs. Messages are presented in a threaded view, with messages from the user on the right side of the screen and messages from the AI displayed on the left side of the screen, mimicking the interface of most popular messenger programs. AI messages highlight essential concepts, offer guidance, and indicate where a response draws on retrieved course material.

At the bottom of the screen is an area for the students to enter their questions, and submissions can be completed using either the "Send" button or the Enter key. All submitted questions are validated for non-emptiness and must fall within the maximum allowable length established by Lion-AI. Students may also access their conversation history and clear the chat to restart. A collapsible sidebar lists previous conversations by title and date and provides controls for starting a new conversation, saving, clearing, switching to dark mode, and logging out. Conversation history is stored in the system database and remains accessible to the authenticated user.

A per-response rating widget and settings page were specified during design but were not implemented in the deployed version. Adding in-product response rating is a priority for the next iteration. The UI for Lion-AI is created using React.js and has been developed based on cognitive load theory [36] to limit all unnecessary items in the interface and to maintain the student's focus on learning activities. Fig. 3 shows the chat interface.

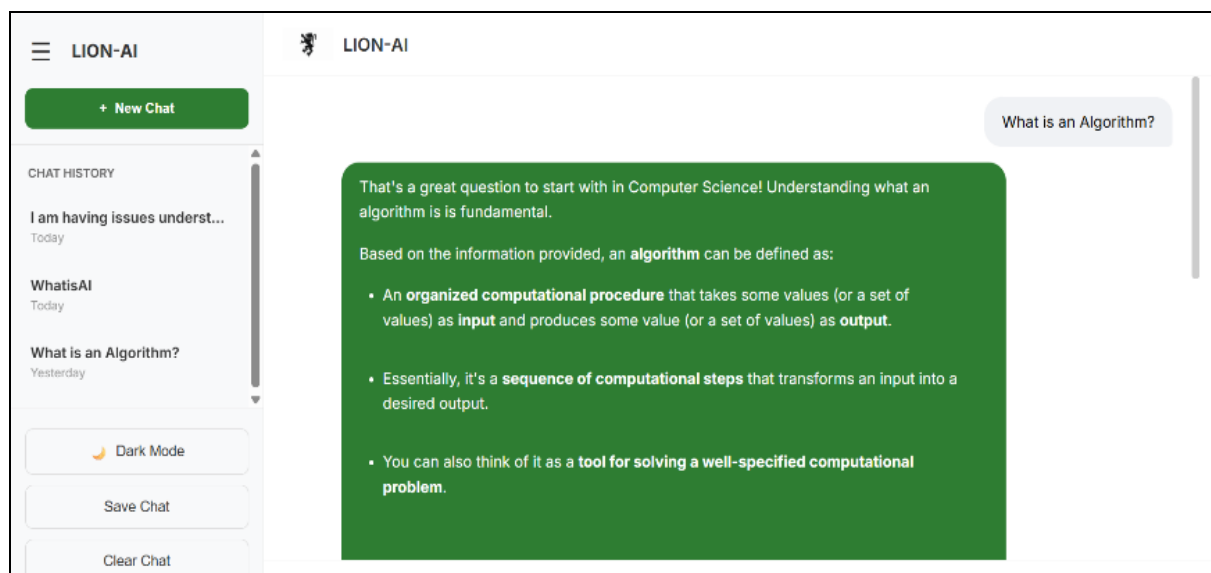


Fig 3 The Lion-AI Chat Interface, Showing the Conversation Pane, Conversation-History Sidebar, and Session Controls.

➤ Security and Privacy Considerations

Due to the sensitive nature of the information collected about students and the academic interactions, Lion-AI takes several precautions to protect the security and privacy of student data. Access to the system is limited to registered and authenticated students and faculty members of the institution. All passwords are hashed with the bcrypt method prior to being stored in the PostgreSQL database, and session tokens are invalidated as soon as students log out of the platform.

All API requests to the Gemini LLM are conducted over a secure HTTPS connection, and API keys are stored on the server and are protected from exposure to the client side.

Because Lion-AI relies on a hosted commercial model, student messages and relevant conversation history are transmitted to the Gemini API. Before transmission, an automated redaction step removes email addresses, telephone numbers, and other straightforwardly identifiable patterns.

This reduces exposure but is not full anonymization, since identifying information written in free-form prose may remain. Institutions considering a similar design should weigh this against locally hosted alternatives [19].

Conversations are stored in PostgreSQL and associated with the authenticated account that created them. Users may delete their own conversations, but no automatic retention limit is currently enforced. The deployed prototype did not display an in-application privacy notice. A clear privacy notice, retention policy, and consent step are prerequisites for wider classroom deployment [7].

IV. IMPLEMENTATION

The process of creating Lion-AI involved implementing a conceptual model to produce a working prototype that integrates the generative abilities of LLMs with pedagogy within a defined software environment. This section describes the implemented system.

➤ *Development Environment and Technology Stack*

Lion-AI was implemented using a full-stack development environment. The AI backend of Lion-AI was developed using Python. The web backend uses Node.js and Express.js. The frontend was implemented using HTML, CSS, JavaScript, and React.js. The structured relational data of Lion-AI is stored in PostgreSQL, and the vector data is stored in ChromaDB. The deployed version employs LangChain 1.2.0, ChromaDB 1.4.0, and the Gemini 2.5 Flash model deployed on a cloud server.

➤ *Core System Modules and Integration*

Lion-AI consists of four connected modules. The User Interface Module manages user interaction and authentication. The User Interface Module interacts with the Lion-AI Web Backend Module, which is a secure means of routing user requests to the system and managing user sessions. The AI Backend Module is responsible for logic pertaining to pedagogy. The AI Backend Module oversees the entire RAG data transfer process. The AI Backend Module uses the Database Management Module to retrieve relevant curriculum segments from the vector database and add them to the user's request. It also combines the retrieved material with the Socratic system prompt before sending the request to the LLM.

The integration of these four modules establishes the technical feasibility of combining curriculum retrieval with an explicit Socratic interaction policy in a deployable prototype. However, determining whether the system consistently adheres to that policy across multi-turn conversations, and whether its use improves student learning, requires a formal classroom evaluation which is planned as the next stage of this work.

V. DISCUSSION, LIMITATIONS, AND FUTURE WORK

The design and implementation of Lion-AI demonstrate the feasibility of developing an LLM-based tutor that prioritizes productive struggle over immediately available solutions. The design process demonstrated the use of a RAG pipeline as well as a way to engineer prompts based on restrictions. The deployment of Lion-AI within a university computer science context indicates the potential of institution-aligned AI tutors to support regular instruction. A recognized challenge, however, is that relying on prompt-based policies alone may not be sufficient for sustained multi-turn tutoring. As the wider literature shows, during longer conversations an LLM tutor can lose dialogue state, give excessive praise, or provide code too directly. These concerns indicate the need for explicit tracking of what the learner has attempted, which misconceptions remain unresolved, and what level of assistance has already been provided.

Several limitations remain. Firstly, the current prototype is limited to one institution's curriculum, which limits its immediate generalizability to other departments and institutions. The knowledge base comprises 32 documents, and expanding this scope is necessary for broader deployment. Secondly, and most significantly, the present study mainly explores the design and implementation phase, as no controlled user study has yet been conducted to determine the actual learning outcomes or students' attitudes towards Lion-AI. Thirdly, enforcing the Socratic interaction policy through prompting alone has known limitations, and its consistency in practice has not yet been formally measured. Fourthly, the system is text-based only and it does not serve students who work with diagrams or visual debugging. Finally, the prototype transmits messages to a commercial model provider and did not display an in-application privacy notice.

Future work will include a formal study with computer science students to assess learning outcomes and self-efficacy associated with Lion-AI use. This research will entail independent coding of the transcripts according to the Socratic rubric. From the technical perspective, the studies will be geared toward the solid quantifiable method of testing Lion-AI in terms of retrieval accuracy and response time. Among the immediate priorities are explicit dialogue-state tracking and an instructor-facing mechanism for designating assessed work. Another future direction is expanding Lion-AI's capacity to support multiple modalities, including image and source-code uploads, and integration with the institution's Learning Management System. Additionally, an instructor-facing interface that allows staff to configure pedagogical parameters and update curriculum materials without developer involvement would enhance the system's long-term institutional utility. Finally, the risk of API-quota failures raises the question of whether locally hosted small language models [19] could provide a more sustainable basis for institutional tutoring where inference costs are a significant constraint.

VI. CONCLUSION

The present paper documents the design and implementation of an AI-based coding tutor named Lion-AI, which has been developed to balance the generative capabilities of LLMs with the ethical obligations of computer science education. The Lion-AI design combined retrieval-augmented generation (RAG) with an explicit Socratic interaction policy to offer a guided, context-specific learning experience in which students are discouraged from relying on academic shortcuts when solving coding problems. By implementing request classification, a graduated support ladder, questioning strategies, and turn-by-turn interaction constraints, Lion-AI provides a transparent interaction policy that goes beyond simply instructing an LLM to “be Socratic.”

At the same time, the design acknowledges that a Socratic prompt can encourage promising tutoring behaviors but does not, by itself, guarantee consistent Socratic tutoring. The behavior of an LLM tutor must therefore be examined turn by turn and supported by continued evaluation and explicit dialogue-state management.

Lion-AI should consequently be regarded as a proof of concept for combining curriculum retrieval with an explicit Socratic interaction policy, rather than as evidence of improved student learning. Future studies should examine whether the system improves students’ conceptual understanding, problem-solving ability, self-efficacy, and academic-integrity practices. As AI continues to proliferate throughout higher education, systems such as Lion-AI may help maintain technology’s role as a complement to, rather than a substitute for, the cognitive work necessary to promote authentic learning.

ACKNOWLEDGMENT

The authors thank the Department of Computer Science, University of Nigeria, for access to institutional course materials and computing resources used in this supervised undergraduate project.

➤ Funding

No funds, grants, or other financial support were received for conducting this study or preparing the manuscript. The system was developed using standard institutional and open-source computing resources.

➤ Competing Interests

The authors declare no financial competing interests or relationships between themselves and any individual or organization that might be perceived to have influenced the work detailed in this paper.

➤ Ethics Approval And Consent To Participate

Not applicable. This paper describes the design and technical implementation of a software system. No human participant data, interaction logs, or user feedback were analyzed for research purposes in this study.

➤ Data Availability

The university class materials used to construct the RAG knowledge base are proprietary institutional material and therefore are not publicly available.

➤ Materials Availability

The knowledge base underlying the retrieval-augmented generation (RAG) component was constructed from departmental teaching materials (lecture notes and instructor-prepared study resources) spanning multiple undergraduate courses. These materials are internal institutional resources and are not publicly available. The materials were used with departmental awareness as part of a supervised undergraduate project. No excerpts of these materials are reproduced in this article.

➤ Code Availability

The software artifact created (Lion-AI source code and prompt architecture) may be obtained from the corresponding author upon request, subject to institutional approval and licensing limitations.

REFERENCES

- [1]. A. Gilson et al., “How does ChatGPT perform on the United States Medical Licensing Examination (USMLE)? The implications of large language models for medical education and knowledge assessment,” *JMIR Medical Education*, vol. 9, Art. no. e45312, 2023, doi: 10.2196/45312.
- [2]. J. Rudolph, S. Tan, and S. Tan, “ChatGPT: Bullshit spewer or the end of traditional assessments in higher education?” *Journal of Applied Learning and Teaching*, vol. 6, no. 1, pp. 342–363, 2023, doi: 10.37074/jalt.2023.6.1.9.
- [3]. B. A. Becker et al., “Programming is hard—or at least it used to be: Educational opportunities and challenges of AI code generation,” in *Proc. 54th ACM Tech. Symp. Comput. Sci. Educ. V. 1*, 2023, pp. 500–506, doi: 10.1145/3545945.3569759.
- [4]. S. Lau and P. Guo, “From ‘Ban it till we understand it’ to ‘Resistance is futile’: How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot,” in *Proc. 2023 ACM Conf. Int. Comput. Educ. Res. V. 1*, 2023, pp. 106–121, doi: 10.1145/3568813.3600138.
- [5]. P. Denny, V. Kumar, and N. Giacaman, “Conversing with Copilot: Exploring prompt engineering for solving CS1 problems using natural language,” in *Proc. 54th ACM Tech. Symp. Comput. Sci. Educ. V. 1*, 2023, pp. 1136–1142, doi: 10.1145/3545945.3569823.
- [6]. J. Leinonen et al., “Using large language models to enhance programming error messages,” in *Proc. 54th ACM Tech. Symp. Comput. Sci. Educ. V. 1*, 2023, pp. 563–569, doi: 10.1145/3545945.3569770.

- [7]. E. Kasneci et al., “ChatGPT for good? On opportunities and challenges of large language models for education,” *Learning and Individual Differences*, vol. 103, Art. no. 102274, 2023, doi: 10.1016/j.lindif.2023.102274.
- [8]. A. L. Stevens and A. Collins, “The goal structure of a Socratic tutor,” in *Proc. 1977 Annu. Conf.*, 1977, pp. 256–263, doi: 10.1145/800179.810212.
- [9]. L. J. Tamang et al., “A comparative study of free self-explanations and Socratic tutoring explanations for source code comprehension,” in *Proc. 52nd ACM Tech. Symp. Comput. Sci. Educ.*, 2021, pp. 219–225, doi: 10.1145/3408877.3432423.
- [10]. P. Denny et al., “Computing education in the era of generative AI,” *Commun. ACM*, vol. 67, no. 2, pp. 56–67, 2024, doi: 10.1145/3624720.
- [11]. N. Raihan et al., “Large language models in computer science education: A systematic literature review,” in *Proc. 56th ACM Tech. Symp. Comput. Sci. Educ. V. 1*, 2025, pp. 938–944, doi: 10.1145/3641554.3701863.
- [12]. B. Tabarsi et al., “Herald of advancement, disruption, or both: A systematic literature review on student-facing LLM tools in undergraduate computing education,” *TechRxiv*, 2025, doi: 10.36227/techrxiv.176463808.80840600/v1.
- [13]. M. Valle Torre, M. Specht, and C. Oertel, “LLM chatbots in high school programming: Exploring behaviors and interventions,” in *Proc. 41st ACM/SIGAPP Symp. Appl. Comput.*, 2026, pp. 79–86, doi: 10.1145/3748522.3779870.
- [14]. R. Cardell-Oliver et al., “LLM-led Socratic dialogues on ethics in computing,” in *Proc. 31st ACM Conf. Innov. Technol. Comput. Sci. Educ. V. 1*, 2026, pp. 121–127, doi: 10.1145/3803400.3809325.
- [15]. H. Crompton and D. Burke, “Artificial intelligence in higher education: The state of the field,” *Int. J. Educ. Technol. Higher Educ.*, vol. 20, no. 1, Art. no. 22, 2023, doi: 10.1186/s41239-023-00392-8.
- [16]. P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in *Adv. Neural Inf. Process. Syst.* 33, 2020, pp. 9459–9474.
- [17]. Y. Gao et al., “Retrieval-augmented generation for large language models: A survey,” *arXiv*, 2023, doi: 10.48550/arXiv.2312.10997.
- [18]. J. Wei et al., “Chain-of-thought prompting elicits reasoning in large language models,” in *Adv. Neural Inf. Process. Syst.* 35, 2022, pp. 24824–24837.
- [19]. Z. Yu et al., “Integrating small language models with retrieval-augmented generation in computing education: Key takeaways, setup, and practical insights,” in *Proc. 56th ACM Tech. Symp. Comput. Sci. Educ. V. 1*, 2025, pp. 1302–1308, doi: 10.1145/3641554.3701844.
- [20]. P. Oli et al., “Exploring the effectiveness of reading vs. tutoring for enhancing code comprehension for novices,” in *Proc. 39th ACM/SIGAPP Symp. Appl. Comput.*, 2024, pp. 38–47, doi: 10.1145/3605098.3636007.
- [21]. W. Xie et al., “STAP: A Socratic tutor for adaptive programming with pedagogical scaffolding,” in *Proc. 2025 2nd Int. Symp. Artif. Intell. Educ.*, 2025, pp. 584–587, doi: 10.1145/3775073.3775165.
- [22]. A. Thonge and A. Thakkar, “A good rubber duck does not quack: Designing Socratic scaffolding in AI tutors,” in *Proc. 31st ACM Conf. Innov. Technol. Comput. Sci. Educ. V. 1*, 2026, pp. 196–202, doi: 10.1145/3803400.3809368.
- [23]. L. Mao et al., “Designing scaffolding cards to facilitate LLM-based Socratic instruction: An exploratory study of response strategies to support learning,” in *Proc. 2026 CHI Conf. Hum. Factors Comput. Syst.*, 2026, Art. no. 72, pp. 1–23, doi: 10.1145/3772318.3791696.
- [24]. M. Liffiton et al., “CodeHelp: Using large language models with guardrails for scalable support in programming classes,” in *Proc. 23rd Koli Calling Int. Conf. Comput. Educ. Res.*, 2023, Art. no. 8, pp. 1–11, doi: 10.1145/3631802.3631830.
- [25]. R. Liu et al., “Teaching CS50 with AI: Leveraging generative artificial intelligence in computer science education,” in *Proc. 55th ACM Tech. Symp. Comput. Sci. Educ. V. 1*, 2024, pp. 750–756, doi: 10.1145/3626252.3630938.
- [26]. Q. Liang, “Cogniti, the Socratic AI tutor, and its impact on student learning in a conservatory musicology course,” M.S. thesis, The Univ. Sydney, Sydney, Australia, 2025.
- [27]. Univ. Sydney, “Cogniti,” 2026. [Online]. Available: <https://cogniti.sydney.edu.au/>. Accessed: [July 28, 2026]
- [28]. J. Daugdaug et al., “Bruno: A RAG-based programming learning assistant for guided, no-code problem solving,” in *Proc. 2025 9th Int. Conf. Educ. E-Learn.*, 2025, pp. 147–158, doi: 10.1145/3789859.3789864.
- [29]. H. Goyal et al., “Saksham AI: Advancing AI-assisted coding education for engineering students in India through Socratic tutoring and comprehensive feedback,” *ACM J. Comput. Sustain. Soc.*, vol. 4, no. 2, Art. no. 6, 2026, doi: 10.1145/3788679.
- [30]. H. B. Essel et al., “The impact of a virtual teaching assistant (chatbot) on students’ learning in Ghanaian higher education,” *Int. J. Educ. Technol. Higher Educ.*, vol. 19, Art. no. 57, 2022, doi: 10.1186/s41239-022-00362-6.
- [31]. A. R. Hevner et al., “Design science in information systems research,” *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.
- [32]. V. Aleven et al., “Example-tracing tutors: Intelligent tutor development for non-programmers,” *Int. J. Artif. Intell. Educ.*, vol. 26, no. 1, pp. 224–269, 2016, doi: 10.1007/s40593-015-0088-2.
- [33]. K. VanLehn, “The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems,” *Educ. Psychol.*, vol. 46, no. 4, pp. 197–221, 2011, doi: 10.1080/00461520.2011.611369.
- [34]. R. Paul and L. Elder, *The Miniature Guide to Critical Thinking: Concepts and Tools*. Foundation for Critical Thinking Press, 2007.

- [35]. R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 8th ed. McGraw-Hill, 2014.
- [36]. J. Sweller, "Cognitive load theory," in *The Psychology of Learning and Motivation: Cognition in Education*, J. P. Mestre and B. H. Ross, Eds. Academic Press, 2011, vol. 55, pp. 37–76, doi: 10.1016/B978-0-12-387691-1.00002-8.
- [37]. N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. 2019 Conf. Empir. Methods Nat. Lang. Process. 9th Int. Joint Conf. Nat. Lang. Process.*, 2019, pp. 3982–3992, doi: 10.18653/v1/D19-1410.
- [38]. H. Chase, "LangChain," GitHub, 2022. [Online]. Available: <https://github.com/langchain-ai/langchain>. Accessed: [July 30, 2026]