AsuLogo.png

Faculty of Engineering, Ain Shams University

Computer Engineering and Systems Department (CESS)

Project Report

CESS Chatbot: LLM-Powered Personal Tutor

Authors:

Ezz Eldin Alaa Eldin Shalaby — 21p0100

Omar Mohammed Korkor — 21p0065

Mohamed Omar Elbialy — 21p0068

Karim Sherif Louis — 21p0223

Abdulrahman Sherif — 21p0098

Ali Mohamed Refaat — 21p0105

2025

Contents

1 Introduction **1**

2 Literature Review **2**

2.1 Overview . . . 2
2.2 Introduction . . . 2
2.3 Scope and Search Strategy . . . 3
2.3.1 Inclusion/Exclusion Criteria . . . 3
2.3.2 Search Strings . . . 3
2.4 Background and Key Concepts . . . 3
2.4.1 From ITS to LLM Tutors . . . 3
2.4.2 The Socratic Tutoring Paradigm . . . 4
2.4.3 RAG Fundamentals for Course Tutors . . . 5
2.4.4 Fine Tuned Models . . . 6
2.4.5 Evaluation Glossary . . . 7
2.5 Thematic Synthesis of Prior Work . . . 9
2.5.1 Theme 1: Pedagogy (Socratic, Hint-First, Feedback) . . . 9
2.5.2 Theme 2: Grounding with RAG over Course Materials . . . 11
2.5.3 Theme 3: Personalization and Adaptive Scaffolding . . . 13
2.5.4 Theme 4: Effectiveness in Real Courses . . . 14
2.5.5 Theme 5: Guardrails and Integrity . . . 14
2.6 Gaps, Challenges, and Emerging Trends . . . 16
2.6.1 Faithfulness and Groundedness . . . 16
2.6.2 Scalability and Model Size Compensation . . . 16
2.6.3 Equity and Reliance . . . 17
2.6.4 Security and Robustness . . . 17
2.6.5 Emerging Trend: “Smaller Models, Smarter Pipelines” . . . 17
2.7 Implications for This Project . . . 18
2.8 Summary . . . 18

3 Methodology **20**

4 System Design **21**
4.1 Design Goals and Requirements . . . 21
4.1.1 Functional Requirements . . . 21
4.1.2 Non-Functional Requirements . . . 21
4.2 High-Level Architecture . . . 22
4.2.1 Main Components . . . 22
4.3 RAG Pipeline Design . . . 23
4.3.1 Document Processing and Chunking . . . 23
4.3.2 Embedding Generation . . . 23
4.3.3 Hybrid Retrieval (BM25 + Dense) . . . 23
4.3.4 Re-ranking and Context Assembly . . . 23
4.3.5 Grounding Strategy . . . 23
4.4 Model Selection . . . 24
4.4.1 Candidate Models Considered . . . 24
4.4.2 Selection Criteria . . . 24
4.4.3 Model Size Tradeoffs . . . 24
4.5 Model Benchmarking Comparison . . . 25
4.5.1 Performance of Small (1–4B), Medium (7B), and Large Models . . 25
4.5.2 Implications for Model Selection . . . 25
4.5.3 Final Model Choice . . . 25
4.6 Pedagogical Control Layer . . . 26
4.6.1 Prompt Templates . . . 26
4.6.2 Guardrails . . . 26
4.7 Personalization Engine . . . 26
4.7.1 User Profile . . . 26
4.7.2 Adaptive Response Generation . . . 27
4.8 Summary . . . 27

Chapter 1

Introduction

This document presents the design, motivation, and technical foundation for the development of the CESS Chatbot — an LLM-powered tutoring assistant for university-level computer engineering courses. The system integrates pedagogical best practices, retrieval-augmented generation, guardrails, and personalization for scalable educational support.

Chapter 2

Literature Review

2.1 Overview

This literature review synthesizes research on intelligent tutoring systems, large language models (LLMs), and retrieval-augmented generation (RAG) as foundations for developing a course-aligned tutoring chatbot for CESS students. It maps key pedagogical strategies, grounding methods, and evaluation metrics, while examining guardrails and integrity mechanisms. The review identifies gaps in current educational LLM applications and derives implications for designing a scalable tutoring assistant tailored to computer engineering education.

2.2 Introduction

Computer Engineering (CE) and Computer Science (CS) programs constantly introduce new technologies and concepts, while enrollments continue to grow. As one-to-one tutoring becomes less feasible at scale, students could benefit from a reliable personal chatbot tutor that helps them make sense of course material, guides them toward solutions, and supports learning new programming languages and techniques. Crucially, such a system must be designed to reduce overreliance on LLMs and be rigorously grounded in university course content and other authoritative sources.

Key Takeaway

Students would greatly benefit from having an LLM that can emulate the role of a 1 on 1 tutor

2.3 Scope and Search Strategy

2.3.1 Inclusion/Exclusion Criteria

- **Databases**: ACM Digital Library, IEEE Xplore, arXiv,ResearchGate
- **Years**: 2023–2025 (inclusive)
- **Scope / Domains**: Intelligent Tutoring Systems (ITS), Computer Science Education, LLM-based tutoring/teaching assistants, Retrieval-Augmented Generation (RAG) & fine tuning applied to course materials
- **Language**: English

Exclusion Criteria

- Outside the databases listed above or outside the year range
- Outside scope (e.g., general chatbots with no educational context; non-LLM systems unless used as baselines in included studies).
- Non-English texts.
- inaccessible full text; duplicates.

2.3.2 Search Strings

Examples: “LLM tutor”, “RAG education”, “course-specific chatbot”, “Socratic prompting programming”, “university educational tutor”, “personal chatbot tutor”, “hallucination mitigation”, “CS tutor”.

2.4 Background and Key Concepts

2.4.1 From ITS to LLM Tutors

Intelligent tutoring systems started long before the age of LLMs and generative AI. Since the 1980s, researchers have tried developing automated systems that try to help tutor students. In 1995, Koedinger et al. [1] introduced the *Cognitive Tutor*, a class of intelligent tutoring systems grounded in cognitive psychology. These systems modeled student knowledge explicitly, provided step-by-step feedback, and used mastery learning to personalize instruction. These techniques demonstrated significant gains in learning speed and retention. This approach established key principles such as student modeling, adaptive feedback, and data-driven refinement that directly inform today’s LLM-based tutoring architectures. Modern large language model (LLM) tutors inherit and generalize these same goals through generative and retrieval–augmented architectures. Rather than hand–crafted cognitive models, LLM tutors encode pedagogical reasoning within large transformer networks trained on vast corpora, enabling open–ended dialogue, Socratic questioning, and hint generation. As summarized by Wang et al. [8], current systems use fine–tuned or retrieval–augmented LLMs for question solving, error correction, and adaptive feedback, often employing chain-of-thought prompting or multi-agent dialogues to emulate stepwise human tutoring. In contrast to traditional ITSs' rule-based models, LLM tutors dynamically generate explanations, adjust difficulty via in-context reasoning, and draw on retrieval–augmented generation (RAG) to ground their responses in authentic course materials. This technological shift marks a transition from *explicit student models* to *implicit knowledge representations* within neural architectures continuing the same educational mission that was first articulated in the Cognitive Tutor.

2.4.2 The Socratic Tutoring Paradigm

The Socratic tutoring paradigm originates from the ancient Greek philosopher Socrates, who emphasized teaching through question-based dialogue rather than direct explanation. His method aimed to stimulate critical thinking, reflection, and self-discovery, encouraging learners to uncover knowledge by reasoning through questions. Instead of transmitting facts, Socrates used guided inquiry to help students challenge assumptions and develop a deeper understanding — a principle that continues to influence modern educational theories and tutoring approaches.

A classic illustration of this method can be seen in dialogues that encourage learners to examine their own reasoning rather than accept assumptions at face value. For example, Shah (2006) [2] presents a short exchange between a professor and a student who claims to "work best under pressure."

Key Takeaway

Example: Socratic Dialogue (adapted from [2])

Professor: The assigned paper is due next Monday. Does anyone have any questions now about the topic so that you can get to work on it?

Student: I don't have any questions now because I'm not going to start working on the paper until Sunday night.

Professor: I appreciate your candor, but why are you going to wait until Sunday night?

Student: Because I always do my best work when I am under pressure.

Professor: Very interesting. Have you ever done a paper when you were not under pressure?

Student: No.

Professor: Then how do you know that you do your "best" work when under pressure when the only work you do is under pressure? You have no basis for comparison.

This exchange demonstrates the essence of the Socratic method—using guided questioning to help learners reflect on their own reasoning and challenge unexamined beliefs. Rather than offering direct instruction, the teacher provokes insight, illustrating how dialogue can transform understanding through self-discovery. Through a series of questions, the professor leads the student to realize the flaw in that belief — demonstrating how Socratic questioning guides reflection and self-evaluation rather than delivering direct instruction. This interaction highlights the enduring purpose of the Socratic method: to foster awareness, reasoning, and intellectual humility through guided inquiry rather than authoritative teaching.

Large Language Models (LLMs) are naturally well-suited for Socratic tutoring due to their conversational fluency and contextual adaptability. Through strategies like Socratic prompting and hint-based scaffolding, LLMs can guide students toward solutions without providing them outright. This aligns with the educational goal of cultivating critical thinking and independent reasoning. Moreover, such dialogue-driven interaction mirrors the classical Socratic exchange, making LLM tutors capable of delivering more human-like, reflective learning experiences.

2.4.3 RAG Fundamentals for Course Tutors

Retrieval-Augmented Generation (RAG) provides the foundational mechanism for connecting large language models with course-specific materials, ensuring that generated responses remain factually grounded and pedagogically aligned. In the context of course tutors, RAG systems enhance accuracy, transparency, and instructional value by integrating retrieval, grounding, and attribution processes.

Chunking involves dividing instructional content into semantically coherent units—such as learning objectives, slide sections, or conceptual paragraphs, so that retrieval aligns with distinct pedagogical ideas. These chunks are transformed into **embeddings**, dense vector representations that enable semantic retrieval based on meaning rather than exact keywords.

To balance precision and conceptual coverage, tutors employ **hybrid retrieval** that combines lexical methods like BM25 with dense embedding search. Retrieved content is then **re-ranked** using heuristic or learned models to prioritize material that is authoritative, pedagogically relevant, and concise enough for effective prompting.

Effective **grounding and attribution** link each generated response explicitly to its source materials, improving transparency and supporting instructor verification. Clear **citation policies**, for instance, referencing slides or readings within answers—help students trace information back to its origin and foster academic integrity.

Finally, RAG-based tutors should include an **uncertainty display** mechanism that surfaces confidence levels or alternative interpretations when retrieved evidence is weak or ambiguous. This promotes responsible AI behavior and supports a reflective, inquiry-based learning experience.

Overall, RAG pipelines for course tutors integrate retrieval precision, grounding, and interpretability to produce responses that are both accurate and educationally meaningful.

2.4.4 Fine Tuned Models

Fine-tuning is a crucial process for adapting pre-trained large language models (LLMs) to specific tasks or domains. In the context of educational applications, fine-tuning allows a general-purpose model to specialize in providing pedagogical guidance, personalized tutoring, and domain-specific knowledge. Unlike training a model from scratch, fine-tuning leverages the knowledge already embedded in the pre-trained model, making it both computationally efficient and effective for a wide range of tasks. This process typically involves training the model on a smaller, task-specific dataset, where only a subset of parameters are updated, allowing the model to learn the nuances of the target application.

Given the growing size of LLMs, traditional fine-tuning methods, which involve updating all model parameters, can be computationally expensive and resource-intensive. To address these challenges, more efficient approaches, such as Low-Rank Adaptation (LoRA), have been developed. LoRA introduces low-rank matrices into the model, leaving the original pre-trained weights frozen. Only these low-rank matrices are fine-tuned, dramatically reducing the number of parameters that need to be updated and making the fine-tuning process more efficient. For educational LLMs, this means that large models can be adapted to specific pedagogical tasks, such as providing Socratic feedback or personalized assistance, without the need for extensive computational resources.

To further optimize efficiency, quantized LoRA (Q-LoRA) introduces quantization techniques that reduce the precision of the low-rank matrices, further reducing the memory footprint and enabling the fine-tuning of large models on hardware with limited resources. This is particularly useful in educational settings, where it is important to deploy fine-tuned models on personal devices or in resource-constrained environments.

For educational applications, fine-tuning does not just focus on adapting the model to a specific domain, but also on aligning the model's behavior with pedagogical objectives. Fine-tuned LLMs can be trained to ask Socratic questions, guiding students to think critically and find solutions independently. The process can also enable personalization, where the model adapts its responses based on the learner's individual progress, knowledge, and needs. Moreover, fine-tuned models can provide formative feedback, helping students understand their mistakes and learn from them, rather than simply providing answers.

While the benefits of fine-tuning for educational LLMs are significant, challenges remain. One key challenge is the need for high-quality, task-specific datasets. For educational fine-tuning, these datasets must reflect real-world student interactions and learning scenarios, which can be time-consuming and expensive to curate. Additionally, fine-tuning on smaller datasets may lead to overfitting, where the model becomes too specialized to the training data and struggles to generalize to new examples. Despite these challenges, fine-tuning remains a powerful tool for adapting LLMs to educational environments, providing an opportunity to deliver more personalized, effective, and pedagogically sound learning experiences.

2.4.5 Evaluation Glossary

When developing educational LLMs, it is essential to assess not only the correctness of responses but also the pedagogical quality, including how well the model helps students understand the material and promotes active learning. Various metrics have been developed to evaluate the performance of LLMs, each offering distinct advantages and limitations, depending on the task and the context in which the model is applied.

BLEU (Bilingual Evaluation Understudy) is a widely used metric for evaluating n-gram overlap between the generated output and reference responses. It is commonly used for machine translation but also applied in educational settings to assess how closely an LLM's response matches the expected form or correct answer. In the context of educational LLMs, BLEU can evaluate the alignment between model-generated feedback or solutions and the reference answers, especially in cases where exact wording is critical. However, BLEU does not account for semantic similarity, meaning responses with the same underlying meaning but different phrasing may score poorly despite being correct.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) measures the recall of n-grams between the generated output and reference answers. Unlike BLEU, ROUGE focuses more on ensuring that relevant content is included in the generated response, making it particularly useful in educational contexts where the focus is on retaining important information. ROUGE evaluates how well the LLM preserves key points from the reference answer but still does not evaluate semantic meaning or logical flow, making it less suitable for assessing the quality of guidance or reasoning in educational feedback.

METEOR (Metric for Evaluation of Translation with Explicit ORdering) improves on BLEU by incorporating synonymy, stemming, and word order, making it more flexible when evaluating the semantic alignment between generated and reference responses. In educational LLMs, METEOR is useful for evaluating responses that may paraphrase a correct answer, as it takes into account variations in phrasing while still preserving the meaning. Since each student and dialogue may differ in phrasing, tone, and expression style, METEOR's ability to handle linguistic diversity makes it particularly suitable for evaluating Socratic questioning and hint-based feedback, where responses aim to guide students without providing the exact answer.

BERTScore uses pre-trained BERT embeddings to compare the semantic similarity between the generated response and reference. By leveraging BERT's contextualized word representations, BERTScore captures the meaning of words rather than just surface-level n-grams. In educational contexts, BERTScore is particularly useful for evaluating Socratic tutoring or feedback, where exact wording may differ, but the semantic content must remain consistent with the learning goals. This metric is advantageous over BLEU and ROUGE for its ability to evaluate responses based on meaning rather than just lexical overlap.

In addition to traditional metrics, newer approaches use LLM-based evaluation to automatically assess the quality of responses. This method typically involves training or prompting a separate LLM to score responses on pedagogical criteria such as helpfulness, creativity, and critical thinking. LLM-based evaluation can scale to large datasets and provides a more nuanced assessment of the model's ability to guide the student through the learning process. For example, the LLM-Score metric evaluates responses based on their alignment with educational goals, such as whether the feedback encourages deeper thinking or active problem-solving.

While automatic metrics like BLEU, ROUGE, and BERTScore provide useful insights into the quality of generated responses, human evaluation remains the gold standard for assessing pedagogical quality. Human evaluators can assess the usefulness, clarity, and correctness of the feedback, as well as its alignment with educational goals. This is especially important for evaluating Socratic dialogue and formative feedback, where the value of the response may not be captured by traditional metrics. However, human evaluation is time-consuming, subjective, and not easily scalable for large datasets.

Focusing on LLM-based evaluation metrics, each metric offers a distinct perspective on the model's performance.

- **BLEU**, **ROUGE**, and **METEOR** focus on lexical and n-gram overlap, with METEOR improving on BLEU by considering synonyms and word order.
- **BERTScore** is better for evaluating semantic similarity, making it more appropriate for complex, meaning-based tasks like Socratic tutoring.
- **LLM-based evaluation** offers an automated approach to measure pedagogical quality, such as helpfulness and creativity, while also being scalable for large datasets.
- **Human evaluation** remains essential for a nuanced, qualitative assessment of pedagogical interactions, particularly when the goal is to measure learning outcomes and critical thinking.

These metrics collectively help provide a comprehensive understanding of how well an educational LLM performs in real-world teaching scenarios, guiding the development of models that not only generate accurate answers but also foster deep learning and critical thinking.

2.5 Thematic Synthesis of Prior Work

2.5.1 Theme 1: Pedagogy (Socratic, Hint-First, Feedback)

A core principle for educational LLMs is guidance over answers. Tutors that simply output solutions whether code, MCQs, or written responses promote overreliance and undermine learning transfer, leading to poorer exam performance. To produce genuine gains, the tutor must embody established pedagogy: Socratic prompting and scaffolding (from hints to partial steps to faded support), formative feedback that targets misconceptions, and explicit alignment with course learning objectives. It should encourage productive struggle (e.g., "why this step?"), calibrate help to the learner's state, and withhold final answers until key reasoning is demonstrated. In short, pedagogy, not mere correctness, must drive interaction design. Several papers have discussed techniques to improve pedagogy [3], [4], [10], [11], [12], [13]. This is certainly important for the scope of out university course tutor. Ross et al. [11] grounded their model development in established educational theories, specifically constructivism and cognitive load theory. They defined nine key pedagogical properties for evaluating AI tutors, with particular emphasis on Socratic guidance (providing conceptual questions rather than explicit solutions) and economy of words (conveying necessary information concisely to reduce cognitive load). Socratic Guidance Implementation: The researchers' fine-tuned models (gpt-4 variants)demonstrated significant improvements in Socratic guidance, with increases ranging from 5.5% to 25.4% compared to base models. This meant the models were more likely to ask guiding questions like "What do you think happens when the loop counter exceeds the array size?" rather than directly stating "You have an off-by-one error." This approach encourages active learning and helps students develop debugging skills independently. Favero et al.[3] also leveraged fine tuned models, they used smaller models (Llama-2-7B/13B) fine-tuned with (Q)LoRA on examples labeled by Socratic category (clarification; probing assumptions; reasons/evidence; implications; alternative viewpoints, etc.). At inference, a role/goal/resources/outcome prompt forces one short Socratic question per turn, suppressing expository answers. The tutor is tested in a dual-agent simulation (Socratic tutor vs simulated learner) across Theory-of-Knowledge problems, with multiple metrics (BLEU/ROUGE/METEOR/BERTScore) and an LLM judge calibrated to critical-thinking quality. The Socratic models significantly outperform “basic” and “random” tutors; 7B performs close to 13B, supporting local/private deployment.

In the context of computer engineering and programming education, the goal of a tutor is not just to correct syntax but to address underlying logical and semantic errors, which novice programmers frequently encounter . Traditional LLM feedback can be misleading on these complex errors, sometimes even making the same mistakes as students .

To maximize the benefits of LLMs while mitigating their limitations in technical accuracy, a dialogue-based tutoring system (DTS) approach is effective [6]. DTS promotes natural language dialogue, encouraging self-reflection and self-explanation, which helps students construct knowledge and identify misconceptions [6].

A robust theoretical foundation for programming tutoring is the Constraint-Based Model (CBM), which is well-suited for ill-defined problems like programming [6]. CBM is based on Ohlsson’s Theory of Learning from Performance Error, which involves two phases: error detection and correction [6].

- Knowledge Representation: Human experts encode domain knowledge as constraints [6]. These constraints, which are prescriptive statements about how a solution *ought* to be, capture both syntactic rules (e.g., assignment expressions) and semantic/logical rules (e.g., using a loop for a range of values) [6].
- Assessment: A student’s solution or answer is checked against these constraints; any violation indicates a misconception [6].
- LLM Integration: The constraints guide the prompts to the LLM, reducing the risk of generating factually incorrect information by grounding the system in expert knowledge [6].

The dialogue manager then uses a Question-Answer strategy, facilitated by Graesser’s five-step tutoring frame, to expose the student’s misconceptions through their reasoning

[6]. Effective feedback, aligned with CBM, should identify where the error is, what constitutes the error, and re-iterate the violated domain principle [6]. For partially correct answers, the system generates follow-up questions to elicit missing information, ensuring a multi-turn, scaffolding conversation [6].

Method at a Glance

Pattern: Fine tune models on data that contains pedagogically sound answers, prompt templates with question decomposition; first offer hints, then progressively deeper guidance; withhold final solutions by policy.

Limitation / Open Issue

Tradeoff between result accuracy and a sound pedagogical approach.

2.5.2 Theme 2: Grounding with RAG over Course Materials

The primary challenge in deploying Large Language Models (LLMs) as educational tutors is mitigating **hallucinations** and ensuring that responses align perfectly with the course-specific curriculum. Retrieval-Augmented Generation (RAG) provides the foundational architecture for connecting the LLM's vast general knowledge to the authoritative course materials, thereby enhancing factual accuracy, pedagogical alignment, and transparency.

Effective grounding begins with a robust retrieval pipeline designed to handle the complexity and interconnectedness of technical knowledge. While conventional RAG relies on simple semantic similarity, this is often insufficient in educational contexts where conceptual relationships are crucial .

- **Knowledge Graph-Enhanced RAG (KG-RAG):** This framework is a significant methodological advancement that integrates a structured **Knowledge Graph (KG)** into the retrieval process to overcome the RAG limitation of retrieving isolated text snippets, which often leads to disconnected or incomplete answers. The KG structures domain knowledge into nodes (concepts) and edges (relationships), aligning with constructivist learning theory where knowledge is built through relationship formation.
  - **Retrieval Method:** KG-RAG uses **Knowledge-Guided Retrieval (KGR)**, which first computes semantic similarity between the user query and KG nodes, and then expands the context by performing a multi-hop traversal across the graph's edges to retrieve related concepts[cite: 2629, 2707, 2584].This ensures the LLM receives a richly interconnected context, which is key to generating a coherent response that maintains logical flow, mirroring expert pedagogical strategies[cite: 2597, 2710].

  - **Impact:** Empirical studies show KG-RAG substantially outperforms standard RAG in terms of **completeness**, **precision** on technical concepts, and **relationship preservation** in explanations.

- **Hybrid and Intent-Based Retrieval:** For Computer Science (CS) and engineering courses, systems utilize hybrid RAG to ensure retrieval accuracy. **CourseAssist** augments GPT-4 with a retrieval engine that classifies user intent and decomposes the question to decide what content to retrieve, leading to responses that are more accurate for CS students. The RAG backbone in **LPITutor** and CourseAssist embeds course documents into a vector space and retrieves top-k passages using cosine similarity to anchor the generation process to authoritative content and minimize hallucination risk.

- **Offline and Efficient RAG:** A critical challenge is enabling RAG in low-resource and offline environments to ensure equitable access . One proposed pipeline pairs a **Small Language Model (SLM)** with an entirely offline RAG system using a pre-loaded vectorized document database built through fixed-size chunking and cosine distance similarity search. However, initial findings show that including RAG context with smaller models like SmolLM can surprisingly lead to a **reduction in accuracy** because the smaller models struggle to effectively manage the extended context window, suggesting the retrieved chunks may act as noise. Future work focuses on exploring advanced techniques like **semantic chunking**, **agentic chunking**, and **meta chunking** to minimize this noise and enhance chunk specialization .

RAG forms the essential layer of accountability and pedagogical control:

- **Content Validation and Traceability:** LPITutor's RAG component ensures factual grounding and minimizes hallucination risk by restricting the LLM's generative space to semantically retrieved, course-relevant content . Furthermore, RAG-augmented systems enhance **transparency** by supporting generated responses with traceable document references or citations, allowing learners to verify the origin of the content .

- **Real-time Knowledge Update:** Systems like Professor Leodar use RAG over fine-tuning because RAG allows the model to access up-to-date and domain-specific information without the need for computationally expensive retraining, which is crucial as course materials are frequently updated . Professor Leodar's knowledge base was continuously updated with weekly homework exercises and solutions, providing timely support .

- **Guardrails and Pedagogical Appropriateness:** RAG supports the core pedagogical policy. CourseAssist defines **pedagogical appropriateness** as whether the system promotes learning by providing hints rather than direct answers. By grounding the LLM's output in verifiable course content, RAG helps prevent the LLM from relying on its general training to "over-assist" students by giving direct solutions, a common problem educators face .

Key Takeaway

Design implication: Use hybrid retrieval → re-rank → MMR dedupe; *always* cite sources line-level or slide-level; show uncertainty when context is weak. For complex conceptual courses like CE/CS, consider integrating a Knowledge Graph to ensure logical coherence across retrieved information.

2.5.3 Theme 3: Personalization and Adaptive Scaffolding

Personalization is an important aspect of a 1-on-1 tutor, as being able to dynamically change the method and level of questioning would be very beneficial for students. A core distinction between a simple chatbot and an effective Intelligent Tutoring System (ITS) is the ability to personalize feedback and guidance to the student's evolving knowledge state. Personalization ensures the tutor is always operating within the **Zone of Proximal Development (ZPD)**, offering productive struggle without frustration. The **LPITutor** model was developed precisely to meet the growing demand for more personalized, scalable, and flexible learning solutions than those offered by conventional, rule-based ITS [9].

The LPITutor framework explicitly achieves personalization by combining **Retrieval-Augmented Generation (RAG)** with a **dynamic, learner-aware prompt engineering strategy** [9]. This design allows the system to adapt the content and style of the Large Language Model's (LLM) output in real-time based on the student's unique profile and interaction history [9].

The overall architecture is comprised of three key components that interact to ensure adaptive tutoring [9]:

- **LLM Engine and RAG Backbone:** The system uses an LLM (based on the GPT-3.5 architecture) as its central generative engine. To overcome hallucination and ensure factual consistency, the LLM is controlled by RAG, which retrieves verified documents from a specialized document database. The retrieved, reliable evidence is then used to augment the prompt, increasing the accuracy and contextual relevance of the LLM's output.

- **Document Database:** This stores the authoritative course materials that serve as the single source of truth for the tutor . This content is indexed and retrieved based on semantic similarity to the student's query .

- **User Profile Database:** This database stores all the essential, real-time data needed for personalization . This metadata includes the learner's knowledge level, query history, and preferred explanation styles . This is the critical component that transforms the general LLM into a personal tutor .

The true personalization aspect of LPITutor lies in the **dual-layer prompt structure** that dynamically leverages the user profile data to control the LLM's pedagogical output :

- The **Static Layer (Pedagogical Template):** This contains pedagogically aligned templates that define the general instructional intent (e.g., "Explain the concept of [topic] using analogies").
- The **Dynamic Layer (Learner Metadata and RAG):** This layer injects the personalized data from the User Profile Database (knowledge level, history) and the relevant retrieved content from the Document Database into the static template.

By blending the static pedagogical *intent* with the dynamic learner *context*, the LLM is controlled to automatically select and generate prompt variations, thus adjusting the **style, depth, and complexity** of its responses to exactly match the learner's needs . This dynamic adaptation is key to implementing an adaptive "hint-first" policy, as the system can choose the appropriate scaffolding level—from a minimalist Socratic question for a knowledgeable student to a more detailed analogy for a struggling beginner—based on its real-time assessment of the user profile [9].

Key Takeaway

Design implication: Implement a dynamic prompting engine that utilizes student history/profile to adjust the difficulty and style of the generated hint. This ensures that the scaffolding is adaptive, operating within the student's Zone of Proximal Development.

2.5.4 Theme 4: Effectiveness in Real Courses

Summarize classroom/semester-long deployments: measures used, duration, findings, and threats to validity (selection bias, novelty effects).

2.5.5 Theme 5: Guardrails and Integrity

A consistent finding across the reviewed literature is the necessity of robust guardrails to ensure that LLM-based tutors operate within well-defined educational boundaries. Unlike general-purpose chatbots, pedagogical tutors must intentionally avoid providing full solutions, generating assessments, or offering content that bypasses the learning objectives of a course. Without such constraints, tutors risk promoting overreliance, enabling academic dishonesty, and undermining skill development—especially in programming-heavy curricula where students may be tempted to offload problem-solving to the model.

Several studies emphasize that even highly capable models (e.g., GPT-4, Gemini Pro) frequently violate pedagogical intent by giving direct answers when prompted or subtly revealing solution structure despite explicit instructions. This reinforces the need for system-level guardrails that restrict the model's behavior beyond what prompting alone can guarantee. For example, CourseAssist implements explicit "pedagogical appropriateness" checks, refusing to give full solutions and instead guiding students through structured hints. Similarly, Favero et al. and Ross et al. rely on fine-tuned Socratic templates that cap answer length, require follow-up questions, and penalize direct answer generation.

In the context of university-level CE/CS courses, guardrails serve multiple integrity functions:

- **Scope Restriction:** The tutor must limit its reasoning to course-approved materials retrieved through RAG. This prevents the model from hallucinating content outside the syllabus and ensures alignment with instructor expectations.
- **Solution Withholding:** Final answers, complete code implementations, and assignment solutions must be withheld. Instead, the tutor provides progressive scaffolding that encourages productive struggle and reflection rather than shortcutting the learning process.
- **Verification Layers:** For programming tasks, sandboxed code execution, unit tests, or static analysis tools can validate student submissions and provide structured feedback, reducing the risk of the model inventing faulty code explanations.
- **Academic Integrity Protections:** Guardrails prevent plagiarism by blocking requests to generate essays, solve graded assignments, or reproduce instructor-only materials. The system should encourage citation of course texts rather than AI-generated answers.
- **Oversight and Monitoring:** Several frameworks incorporate instructor dashboards, allowing educators to review student–tutor interactions. This transparency supports responsible use, facilitates debugging of model behavior, and ensures alignment with course policy.

Across the literature, a growing trend favors using smaller, fine-tuned models precisely because they respond more predictably to guardrails. Large proprietary models tend to override instructions or leak unintended information due to their general-purpose training. In contrast, smaller models paired with strict prompt templates, RAG-constrained grounding, and refusal policies can maintain tighter behavioral control. This observation motivates the design of the proposed tutor, where guardrails operate not as optional enhancements but as core architectural components that shape how the model interacts with students and how reliably it adheres to the educational mission of the system.

2.6 Gaps, Challenges, and Emerging Trends

While the literature demonstrates rapid progress in LLM-based tutoring systems, several notable gaps remain, especially in the area of deploying smaller, resource-efficient models that can match or exceed the pedagogical performance of larger proprietary models. These gaps inform key directions for the development of a course-grounded CESS tutoring assistant.

2.6.1 Faithfulness and Groundedness

Even with modern retrieval-augmented generation (RAG) pipelines, models frequently produce partially or fully ungrounded statements. This is especially pronounced in *smaller* models such as LLaMA-2–7B or Phi–2/3B, which have limited internal world knowledge and are more sensitive to noisy or overly long retrieved context. Prior work shows that while RAG improves factual alignment, it does not fully prevent hallucinations unless paired with strict prompting templates, re-ranking mechanisms, and explicit citation requirements. Emerging research highlights the need for *groundedness metrics* and UI cues that signal uncertainty to learners, enabling more transparent and pedagogically responsible interactions.

2.6.2 Scalability and Model Size Compensation

A recurring challenge is the scalability of maintaining course-specific knowledge bases (KBs) across semesters. Course content frequently changes, making continuous updates to embeddings, chunking strategies, and retrieval scoring necessary. While large proprietary models (e.g., GPT-4, Gemini Pro) offer strong zero-shot performance without specialized tuning, they are expensive, difficult to deploy at scale, and often misaligned with pedagogical requirements (e.g., tendency to "solution dump").

Several papers attempted to address this by using *smaller* open-source models (7B–13B) that were fine-tuned on Socratic question–answer pairs or pedagogically labeled datasets. Favero et al. fine-tuned 7B and 13B LLaMA models to follow specific Socratic templates, achieving near-parity with larger models in critical-thinking tasks. Similarly, Ross et al. showed that pedagogical alignment can improve small-model performance more than increasing parameter count. However, a major research gap remains: the field lacks systematic evidence on whether *combined* RAG + fine-tuning strategies can consistently compensate for reduced model size across technical university subjects.

2.6.3 Equity and Reliance

Most studies evaluate learning gains in controlled experiments, but few examine long-term effects on student independence, transfer to exams, or overreliance on AI tutors. Smaller models, once fine-tuned for hint-first scaffolding, tend to adhere to pedagogical constraints more faithfully than large models, which often override instructions to provide full answers. Yet there is limited empirical work measuring whether such pedagogical alignment actually supports equitable learning outcomes across students with different prior knowledge, or whether smaller models might oversimplify explanations due to capacity limits.

2.6.4 Security and Robustness

Using smaller, locally deployable models introduces different security considerations compared to centralized API-based LLMs. Systems must guard against prompt injection, leakage of course materials, and unintended sharing of proprietary content from the knowledge base. As institutions consider embedding tutors within LMS platforms, evaluating defenses against context manipulation, jailbreak attempts, and poisoned retrieval documents becomes essential. Current research offers limited guidance on securing RAG pipelines—especially when chunked course data are stored on student devices or departmental servers.

2.6.5 Emerging Trend: "Smaller Models, Smarter Pipelines"

Across the literature, an emerging trend is clear: instead of relying on ever-larger models, research is shifting toward building *smarter pipelines* around smaller models. Methods such as:

- LoRA and Q-LoRA fine-tuning on Socratic or hint-first datasets,
- hybrid retrieval (BM25 + embeddings),
- re-ranking to reduce context noise,
- agentic chunking and semantic chunk specialization,
- strict citation and grounding templates,

have shown that well-designed scaffolding can enable models as small as 3B–7B parameters to outperform larger models on pedagogical metrics. However, evidence remains fragmented, and no study offers a full evaluation pipeline combining RAG, fine-tuning, and tutoring-style alignment specifically for engineering or computer science courses. This represents a central research opportunity and motivates the design objectives behind the proposed CESS tutor.

2.7 Implications for This Project

Map literature insights to system:

1. **Tutor Policy:** Socratic, hint-first; rubric-aligned feedback; no direct final solutions.
2. **RAG Pipeline:** Hybrid retrieval + re-ranker + citation enforcement; uncertainty display.
3. **Verification:** Code execution sandbox + tests for programming courses; worked-example checklists for theory courses.
4. **Evaluation Plan:** Learning outcomes, process metrics, quality/safety, and student/TA surveys.

2.8 Summary

Tables and Figures (Templates)

RAG Prompt Sketch (for appendix)

Listing 2.1: System prompt sketch for a course-grounded, hint-first tutor

```
You are a course tutor. Never give final solutions. Use only
   retrieved context.
If insufficient context, say so and ask for clarification. Cite
   slide/page IDs.

[User Question]
{question}

[Retrieved Context]
{chunks_with_ids}

[Output Requirements]
1) Diagnose misconception (if any).
2) Give a scaffolding hint (max 3 sentences).
3) Offer a follow-up question.
4) Cite sources: (Slide X, Week Y, p.Z).
```

Chapter 3

Methodology

Chapter 4

System Design

This chapter describes the overall architecture of the CESS tutoring system, the design rationale behind key components, and the selection of the underlying models, retrieval methods, and fine-tuning strategies. The design is guided by pedagogical requirements, system constraints, and insights from the literature review.

4.1 Design Goals and Requirements

4.1.1 Functional Requirements

- Provide course-grounded tutoring aligned with CESS curricula.
- Support Socratic, hint-first dialogue.
- Retrieve and cite authoritative content from course materials.
- Detect student misconceptions and provide corrective feedback.
- Personalize explanations based on user profile and interaction history.

4.1.2 Non-Functional Requirements

- High reliability and minimal hallucinations.
- Modular architecture for extendability.
- Transparent behavior with source citations.
- Data privacy, academic integrity, and guardrail compliance.
- Efficient performance on moderate hardware (preferably small models).

4.2 High-Level Architecture

Figure 4.1 presents the overall system architecture.

architecture-diagram.png

Figure 4.1: High-level architecture of the CESS tutoring system.

4.2.1 Main Components

1. **User Interface Layer** Chat interface, session management, personalization settings.
2. **Tutoring Engine** Applies pedagogical templates, hint-generation logic, and guardrails.
3. **RAG Pipeline** Retrieval, re-ranking, context assembly, citation management.
4. **LLM Core** The selected base model with (optional) fine-tuned LoRA adapters.
5. **Knowledge Base** Vector database, chunked course slides, textbook excerpts, lecture notes.

6. **Analytics & Logging** For instructor dashboards, performance metrics, and model auditing.

4.3 RAG Pipeline Design

4.3.1 Document Processing and Chunking

- Semantic chunking of slides, readings, and assignments.
- Chunk size: typically 250–500 tokens.
- Overlap strategies to preserve context continuity.

4.3.2 Embedding Generation

- Choice of embedding model (e.g., InstructorXL, bge-small, or local embeddings).
- Handling multilingual or math-heavy content (if applicable).

4.3.3 Hybrid Retrieval (BM25 + Dense)

- BM25: keyword precision for code, formulas, terminology.
- Dense retrieval: semantic matching for conceptual queries.
- Weighted or multi-stage retrieval pipeline.

4.3.4 Re-ranking and Context Assembly

- Cross-encoder or LLM re-ranking.
- Maximal Marginal Relevance (MMR) to reduce redundancy.
- Slide-level citation metadata included in each chunk.

4.3.5 Grounding Strategy

- Tutor restricted to use retrieved content only.
- Uncertainty statements when context is weak.
- Citation formatting aligned with academic integrity policies.

4.4 Model Selection

4.4.1 Candidate Models Considered

- **Large models**: GPT-4-Turbo, Gemini-Pro, Claude 3 — high accuracy but costly and inconsistent with guardrails.
- **Medium open models**: LLaMA-3 (8B/70B), Mistral 7B — strong reasoning, good baseline.
- **Small/efficient models**: Phi-3, LLaMA-3-Instruct-8B, Qwen-7B — suitable for local deployment and tighter guardrail control.

4.4.2 Selection Criteria

- Pedagogical controllability (ability to follow hint-first style)
- RAG grounding performance
- Model size vs hardware limits
- Ease of fine-tuning (LoRA/Q-LoRA support)
- Hallucination resistance
- Cost and deployment feasibility

4.4.3 Model Size Tradeoffs

In general, language models exhibit scaling behaviour: as the number of parameters increases, average performance on broad benchmark suites tends to improve, and very large models can display emergent abilities that are absent at smaller scales. This would suggest that GPT-4-class models should always dominate smaller 7–8B models. However, for a *narrow, course-specific tutoring task*, the performance gap can be substantially reduced—or even bridged—by combining a small or medium model with task-specific adaptation (fine-tuning) and a retrieval-augmented generation (RAG) pipeline over high-quality local materials.

Liu et al. (2024) empirically demonstrate this in a CS1 setting: a locally deployed 7B model (neural-chat-7b-v3, a fine-tuned Mistral-7B variant) equipped with a RAG layer over course content achieves scores comparable to, and in some evaluation settings slightly higher than, GPT-4-32k on real student help-desk conversations, while running on a single consumer GPU and keeping all data on-premise.[5] This supports our design choice: instead of relying exclusively on large, cloud-only models, we can use a small/medium open model that fits our hardware budget and then recover most of the performance through

(i) Socratic-style fine-tuning and (ii) RAG grounding on the course's slides, assignments, and exemplar solutions.

4.5 Model Benchmarking Comparison

To evaluate the suitability of our candidate models, we compare their performance using standard reasoning and knowledge benchmarks frequently used in recent literature. According to the survey *A Survey of Small Language Models* (2024), the core benchmarks include MMLU, HellaSwag, ARC, PIQA, and Winogrande. These benchmarks assess multi-domain knowledge, commonsense reasoning, scientific reasoning, physical reasoning, and coreference resolution.[7]

4.5.1 Performance of Small (1–4B), Medium (7B), and Large Models

Table 4.2 summarizes key results extracted from Table 8 of the survey (p. 38) and from the efficiency metrics in Table 9 (p. 39). Small models such as Phi-3.5-mini (3.8B) deliver strong performance (e.g., 69.0 MMLU, 81.4 HellaSwag), approaching the lower range of 7B models, while maintaining substantially lower memory and compute requirements. Medium-scale models (7B) consistently achieve higher accuracy (75–78 MMLU), but require 3–10× more memory (p. 39). Large proprietary models (e.g., GPT-4o, Claude 3.5) achieve state-of-the-art results but are significantly more costly and unsuitable for on-premise deployment.

4.5.2 Implications for Model Selection

The survey notes that small models can match or nearly match large models on *domain-specific tasks* once combined with high-quality training, retrieval augmentation, or distillation techniques. For our course-specific tutoring system, this suggests that a 7B or sub-7B open model can achieve competitive pedagogical performance when integrated with Socratic fine-tuning and RAG. Medium models offer the best accuracy, while small models provide superior deployment efficiency.

4.5.3 Final Model Choice

(A placeholder for your final decision)
Examples:

- Base model: LLaMA-3-Instruct 8B
- With Q-LoRA adapters for Socratic tuning

Table 4.1: Comparison of representative small and medium models according to *A Survey of Small Language Models* (2024), based on Table 8 (p. 38) and Table 9 (p. 39).

Table 4.2: Compact comparison of small and medium models.

Model	**Params**	**MMLU**	**Hella**	**ARC**	**Mem (GB)**	**Tok/s**
Phi-3.5-mini	3.8B	69.0	81.4	87.4	∼12	32
LLaMA 3.2	3B	63.4	69.8	78.6	—	—
Gemma 2	2B	57.8	61.1	76.7	—	—
Falcon-7B	7B	75.9	47.5	78.5	27.8	14.9
LLaMA 2–7B	7B	76.8	48.5	76.7	27.8	14.9
MPT-7B	7B	77.6	46.5	77.3	—	—
GPT-4o*	—	∼90	—	—	Cloud	—
Claude 3.5*	—	> 90	—	—	Cloud	—

- Trained on: 1200+ Socratic/hint-first samples + course-specific reasoning cases

4.6 Pedagogical Control Layer

4.6.1 Prompt Templates

- Socratic structure
- Misconception diagnosis
- Hint-first scaffolding
- Follow-up question generation

4.6.2 Guardrails

- No-solution policy for graded materials
- Restricted generation using RAG evidence only
- Refusal templates for violations

4.7 Personalization Engine

4.7.1 User Profile

- Knowledge level
- Interaction history
- Preferred explanation style

- Weak topics (tracked via tagging)

4.7.2 Adaptive Response Generation

- Varies difficulty
- Chooses explanation depth
- Selects analogies, examples, or visuals based on user needs

4.8 Summary

This chapter presented the architectural blueprint of the CESS tutoring system, detailing the RAG pipeline, model selection process, pedagogical control layer, and personalization components. These design decisions form the basis for implementation and experimentation in later chapters.

Bibliography

[1] J. R. Anderson, A. T. Corbett, K. R. Koedinger, and R. Pelletier, "Cognitive tutors: Lessons learned," *The Journal of the Learning Sciences*, vol. 4, no. 2, pp. 167–207, 1995. DOI: 10.1207/s15327809jls0402_2

[2] P. Boghossian, "The socratic teaching method: A therapeutic approach to learning," *Critical Thinking Across the Disciplines*, vol. 26, no. 3, pp. 4–9, 2006. [Online]. Available: https://www.researchgate.net/publication/298107288_The_Socratic_Teaching_Method_A_Therapeutic_Approach_to_Learning

[3] L. Favero, J. A. Pérez-Ortiz, T. Käser, and N. Oliver, "Enhancing critical thinking in education by means of a socratic chatbot," *arXiv preprint arXiv:2409.05511*, 2024. [Online]. Available: https://arxiv.org/abs/2409.05511

[4] T. Feng, S. Liu, and D. Ghosal, *Courseassist: Pedagogically appropriate ai tutor for computer science education*, arXiv:2407.10246, v3, 2024. [Online]. Available: https://arxiv.org/abs/2407.10246

[5] S. Liu, Z. Yu, F. Huang, Y. Bulbulia, A. Bergen, and M. Liut, "Can small language models with retrieval-augmented generation replace large language models when learning computer science?" In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024)*, New York, NY, USA: Association for Computing Machinery, 2024, pp. 388–393. DOI: 10.1145/3649217.3653554

[6] J. Perez and E. Ong, "Designing an LLM-based dialogue tutoring system for novice programming," in *Proceedings of the 32nd International Conference on Computers in Education (ICCE 2024)*, A. Kashihara et al., Eds., Asia-Pacific Society for Computers in Education, 2024. [Online]. Available: https://library.apsce.net/index.php/ICCE/article/view/4954

[7] F. Wang et al., "A survey of small language models," *arXiv preprint arXiv:2411.03350*, 2024. [Online]. Available: https://arxiv.org/abs/2411.03350

[8] S. Wang et al., "Large language models for education: A survey and outlook," *arXiv preprint arXiv:2403.18105*, 2024.

[9] Z. Liu, P. Agrawal, S. Singhal, V. Madaan, M. Kumar, and P. K. Verma, “LPITutor: An LLM based personalized intelligent tutoring system using RAG and prompt engineering,” *PeerJ Computer Science*, vol. 11, e2991, 2025. DOI: 10.7717/peerj-cs.2991

[10] K. Qian et al., *Dean of llm tutors: Exploring comprehensive and automated evaluation of llm-generated educational feedback via llm feedback evaluators*, 2025.

[11] E. Ross, Y. Kansal, J. Renzella, A. Vassar, and A. Taylor, *Supervised fine-tuning llms to behave as pedagogical agents in programming education*, arXiv:2502.20527, v1, 2025. [Online]. Available: https://arxiv.org/abs/2502.20527

[12] A. Scarlatos, N. Liu, J. Lee, R. G. Baraniuk, and A. Lan, *Training llm-based tutors to improve student learning outcomes in dialogues*, 2025.

[13] S. Song et al., *Cultivating helpful, personalized, and creative ai tutors: A framework for pedagogical alignment using reinforcement learning*, 2025.