

COURSE FORGE AI IN COMPUTERS: AN AGENTIC, MODULAR, AND EXPLAINABLE EDUCATIONAL ASSISTANT USING LLMS AND RAG

LENIN MARKSIA U^{1*}, JEYASHANTHI J², JEGADEESH A³, KALIAPPAN M⁴, MARIAPPAN E⁵, ANGEL HEPZIBAH R⁶, RAMANTH M⁷, MEGA G⁸

¹Assistant Professor, Department of Electronics and Communication Engineering, Francis Xavier Engineering College, Tirunelveli, Tamil Nadu, India.

²Associate Professor, Department of Electrical and Electronics Engineering, Pulloor, Kariapatti, Virudhunagar, Tamil Nadu, India.

³Assistant Professor, Department of Computer Science and Engineering, Infant Jesus College of Engineering, Thoothukudi, Tamil Nadu, India

⁴ Professor, Department of Artificial Intelligence and Data Science, Ramco Institute of Technology, Rajapalayam, Tamil Nadu, India

⁵.Associate Professor, Department of Artificial Intelligence and Data Science, Ramco Institute of Technology, Rajapalayam, Tamil Nadu, India

⁶Associate Professor, Department of Artificial Intelligence and Data Science, Ramco Institute Technology, Rajapalayam, Tamil Nadu, India

⁷ Associate Professor, Department of Artificial Intelligence and Data Science, Ramco Institute Technology, Rajapalayam, Tamil Nadu, India

⁸. UG Student, Department of Artificial Intelligence and Data Science, Ramco Institute Technology, Rajapalayam, Tamil Nadu, India

E-mail: ¹marksia.lenin@gmail.com, ²ssanthisiddhu@gmail.com, ³jeganjn@gmail.com, ⁴jeraldinosephkanagaraj@gmail.com, kaliappan@ritrjpm.ac.in, mariappan@ritrjpm.ac.in, angel@ritrjpm.ac.in, ramath@ritrjpm.ac.in, megha@ritrjpm.ac.in

ABSTRACT

The demand for personalized, interactive learning remains a core challenge in modern education. This paper presents CourseForge AI, a modular and explainable academic assistant designed using Large Language Models (LLMs), autonomous agents, and Retrieval-Augmented Generation (RAG). The system includes specialized agents for curriculum-aligned Question and Answer, note generation, flashcards, tutoring, and coding assistance and all powered by semantic retrieval over academic content. Unlike static chatbots, Course Forge AI exhibits agentic reasoning and decision-making, allowing it to dynamically retrieve and contextualize content in response to user input. By leveraging vector-based semantic search model includes LanceDB and explainable agent workflows through Phidata, the system ensures that its outputs are grounded, interpretable, and relevant to syllabus-specific learning. Experimental validation shows strong performance in user engagement, relevance, and accuracy. This work contributes a scalable blueprint for AI-assisted education that is both intelligent and trustworthy

Keywords: *Agentic AI, AI Assisted learning, Curriculum-Aligned Learning, Large Language Models (LLMs), Retrieval-Augmented Generation (RAG)*

1. INTRODUCTION

The landscape of education has witnessed a transformative shift with the rise of Artificial Intelligence (AI), enabling systems to personalize, adapt, and automate the learning process. Despite the availability of static online resources, learners often struggle with finding context-aware,

interactive, and skill-building platforms tailored to their academic goals. CourseForge AI addresses this challenge by delivering an intelligent assistant capable of understanding academic intent, generating contextual content, and offering real-time evaluation feedback. At its core, the system integrates Google's Gemini large language model (LLM) with modular NLP-driven agents, each

responsible for a specific learning function ranging from automated problemstatement generation to notes synthesis, code evaluation, and flashcard creation. Thisresearch explores the end-to-end development and deployment ofCourse Forge AI,focusing on how NLP is leveraged to create a conversational, intelligent, and personalized learning assistant. The system is not only useful for students preparing for technical interviews or academic exams but also scalable to educational institutions looking for intelligent content delivery platforms.

2. LITERATURE REVIEW

The intersection of artificial intelligence and education has evolved rapidly, driven by the advancements in conversational agents, machine learning, and retrieval-based natural language processing. Prior research has explored both the opportunities and constraints of deploying AI systems within academic and learning contexts.

In broader domains, the use of intelligent systems to address complex problems is well-documented. Ghosh et al. [1], for example, examined how AI models can be applied to healthcare issues, emphasizing decision support and interpretability. While this study is not focused on education, its insights into responsible AI integration are highly transferable. Similarly, Wiggins and McTighe [2] outlined core pedagogical principles for curriculum design, a foundation that remains essential for aligning AI-generated outputs with learning objectives. The societal shift toward AI-augmented work and learning environments has also been documented in global industry reports. Microsoft and LinkedIn's annual trend analysis [3] emphasized the urgent need for AI tools that complement, rather than replace, human capability—especially in professional training and education. Supporting this view, various commentators have argued for cautious adoption of generative AI in classrooms. Lang, Schroeder, and Lederman [4] emphasized the importance of aligning AI with academic integrity, learner support, and instructional purpose.

Research into educational chatbots and learning agents has seen growing momentum. A study on context-aware chatbots [5] demonstrated that embedding instructional context into agent conversations enhances learner interaction, especially in video-based learning

environments. Other projects have focused on simulating human-like communication. For example, [6] proposed a dialog-driven robotic tutor that adjusts its responses using structured linguistic patterns. Similarly, the chatbot described in [7] was trained using encoder-decoder neural architectures to reflect human conversational tone and adapt to learner inputs. Technical insights into the real-world challenges of chatbot development were captured in [8], which mined developer discussions to uncover common limitations in adaptability and design flexibility. Localization has also been addressed; for instance, [9] built a bilingual chatbot for university admissions, tailored to Bangladeshi language contexts and academic workflows.

Several works have explored domain-specific applications of chatbots. TABot [10][11] provided support for software engineering students by offering structured teaching assistant functionality. Meanwhile, [12] investigated the legal implications of generative models like ChatGPT, analyzing governance concerns and institutional risk. Further educational innovations include Tutor Chat [13], which was designed to support dyslexic learners through guided AI dialogue, and the college-focused chatbot in [14][15], aimed at improving institutional communication and student support services. Outside formal education, intelligent agents have also been developed for wellness applications. For example, [16] introduced a conversational agent to aid individuals in managing mental health conditions, highlighting the expanding relevance of AI-driven dialogue systems across multiple sectors.

Zhang and Sorin [17] provided one of the most comprehensive evaluations of educational AI agents, exploring both the strengths and unresolved challenges of integrating chatbots into structured academic settings. Their findings point to the need for systems that are not only responsive and intelligent but also aligned with pedagogical structures and learner needs. While existing systems demonstrate meaningful progress, most remain limited by static retrieval, narrow contextual awareness, or lack of curriculum alignment. The solution proposed in this workCourseForgeAIaims to overcome these constraints through an agentic, tool-augmented architecture that supports modular, scalable, and syllabus-specific educational interactions.Despite the advancements in AI-assisted learning tools, several limitations persist across current implementations:

- **Absence of Autonomous Reasoning:** Most existing platforms rely on static workflows, lacking agentic capabilities such as when to seek external information or how to respond with multi-layered logic.
- **Weak Curriculum Integration:** Few systems integrate directly with institutional materials like textbooks, syllabi, or lecture notes, reducing their contextual relevance in formal education settings.
- **Limited Transparency:** Responses often lack explain ability, diminishing user trust and making it challenging for educators to assess or verify the system's accuracy.
- **Rigid Architectures:** Many platforms follow monolithic designs, making it difficult to scale or incorporate modular features like flashcards, problem generation, or content evaluation.
- **Poor Handling of Complex Queries:** Current LLM tools frequently fail in scenarios requiring multi-step reasoning, document cross-referencing, or persistent contextual adaptation during extended conversations.
- **Insufficient Personalization:** Most AI learning assistants do not dynamically adapt to individual user progress, subject mastery, or interaction history, limiting the delivery of personalized learning experiences.[16][17]

These gaps emphasize the need for a system that is modular, context-aware, explainable, agentic, and aligned with real academic content. The proposed solution, CourseForge AI, is designed to fill this void by combining generative AI with subject-specific knowledge bases and modular learning agents.

3. PROPOSED WORKS

CourseForge AI has been developed following a modular, agent-based design philosophy to deliver an adaptive, transparent, and curriculum-aware learning platform. Central to its design is the use of advanced Natural Language Processing (NLP) methods, with Google's Gemini Large Language Models (LLMs) providing the generative backbone for various academic support tasks. This section explains the system's architectural blueprint, its core building blocks, and the methodological strategies employed to achieve its educational objectives. The system architecture is organized into four interconnected layers. The user interface layer is realized through a Flask-powered web application that enables learners to

engage seamlessly with features such as programming exercises, flashcard generation, and personalized tutoring. Sitting behind this interface, the application layer orchestrates the system's operational logic, handling user sessions, managing requests, and coordinating the execution of feature-specific processes. At the centre of the system lies the agent layer, a collection of specialized AI agents crafted with unique prompts and task-specific instructions. These agents leverage the Gemini LLM to handle diverse tasks — from creating programming challenges and reviewing code submissions to summarizing academic material and delivering guided explanations. To improve the precision and contextual relevance of their outputs, these agents interface with the knowledge layer, which houses vector databases containing academic materials such as textbooks, course content, and research documents. Through semantic search and context-matching, the system can generate outputs closely aligned with the learner's curricular needs.

The operational flow begins when a user provides an input — whether by choosing a learning topic, uploading a document, or submitting a code snippet. This input is routed to the relevant agent, which formulates a specialized prompt and engages with the LLM to generate an appropriate response. If necessary, the agent supplements the output by querying the embedded knowledge base, ensuring that the delivered content is both accurate and aligned with the user's educational context. The final output is then formatted and presented to the user through the interface, with options for saving or downloading materials such as notes and flashcards. In summary, the design approach of CourseForge AI prioritizes adaptability, explainability, and modularity. By structuring the system around independent, task-focused agents, it becomes possible to extend or enhance functionalities without overhauling the system's core architecture. Additionally, the platform's commitment to transparency ensures that learners not only receive answers but also understand the reasoning process behind AI-generated outputs. This combination of modular design and deep curriculum integration makes CourseForge AI a compelling tool for modern, AI-driven education. The proposed system integrates Gemini's LLM and embedding models with an agentic framework to deliver intelligent, personalized academic assistance. The implementation is centered on three key

components: document embedding, semantic retrieval, and agent-guided language generation. The pipeline begins with educational content extraction from structured files, primarily HTML-formatted textbooks or course documents. Each document is parsed and segmented into coherent chunks using an HTML parser such as BeautifulSoup. These chunks are then embedded using Gemini's embedding API, which converts the text into high-dimensional semantic vectors. The resulting embeddings are stored in LanceDB, a high-speed, columnar vector database designed for fast similarity search. Each record is stored alongside metadata such as the document source and chunk identifier, facilitating traceable retrieval. When a user query is submitted, the input is passed through Gemini's embedding model to obtain a vector representation of the query. A top-k similarity search is then performed against the LanceDB index using cosine similarity. The search returns the most relevant document chunks, ranked by semantic closeness to the query. This retrieval process is encapsulated in a search document function, which is exposed as a callable tool within the agent's reasoning framework. This modular design allows the agent to access external knowledge only when needed.

At the core of the system lies an agent built using the Phidata Agent architecture, adapted to use Gemini as the underlying LLM. This agent is configured with a specific system prompt that encourages reasoning, tool invocation, and multi-step thought processes. Upon receiving a query, the agent evaluates whether to respond directly or invoke the `search_docs` tool. If retrieval is deemed necessary, the agent fetches relevant context from LanceDB and integrates it into the final answer formulation. This agentic behavior allows the model to dynamically switch between internal knowledge and retrieved content, offering more accurate and syllabus-aligned responses.

The system supports several intelligent academic functions, each implemented as a modular agent tool:

- Textbook Q&A using RAG (Retrieval-Augmented Generation)
- Note generation from structured topics
- Flashcard and quiz creation from textbook sections
- Chatbot-style tutoring with follow-up support
- Code generation and evaluation,

prompted by conceptual programming queries. All modules share a common backend for retrieval and reasoning, promoting reusability and modular expansion.

3.1 Architecture of an AI-powered learning platform

The proposed system is developed as a highly modular academic assistant, with each function encapsulated as an independent, agent-driven module. All modules leverage a shared foundation that includes Gemini's LLM for reasoning, LanceDB for vector-based retrieval, and structured tool invocation via an agent architecture. This modularity ensures that each feature is independently scalable and collectively contributes to a unified educational support platform.

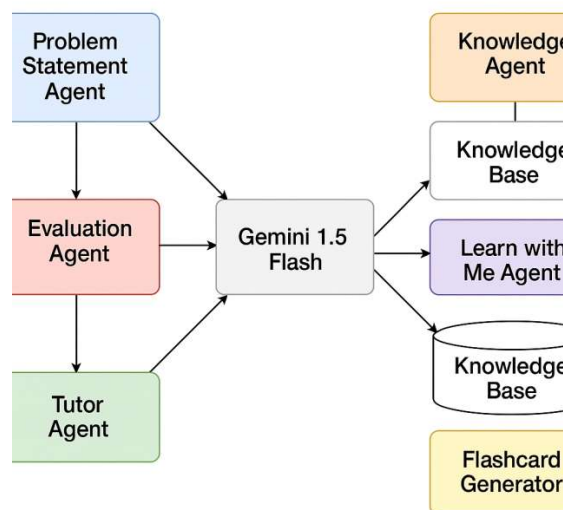


Fig1: Layered Architecture Of An AI-Powered Learning Platform

This system integrating client interfaces, a Flask web server, application logic, AI services such as Gemini API, agents, embedding, and structured storage solutions. It presents an intelligent educational assistant suite powered by the Gemini 1.5 Flash model, designed to enhance academic learning, tutoring, and content generation through specialized agents. The suite comprises multiple task-specific agents, each performing targeted educational functions. The Problem Statement

Agent generates structured Python programming problems, while the Evaluation Agent assesses student-submitted code and offers feedback with solution control logic. The Tutor Agent generates multi-page, well-structured academic notes, and

the Knowledge Agent provides comprehensive answers by leveraging external knowledge bases. Additionally, the Learning Assistant offers direct and concise academic support with structured formatting for ease of understanding. A key component is the flashcard generation system, which transforms raw PDF content into structured question-answer flashcards. This is achieved by chunking the input text and prompting the Gemini model to generate flashcards in JSON format, with fallback mechanisms for degraded output. A mathematical abstraction governs the logic of flashcard creation and ensures robust operation under rate limitations. Supporting diagrams such as UML activity diagrams and state machines model the system behavior, enabling a clear understanding of workflow transitions, fallback decisions, and content validation. This intelligent suite demonstrates a modular and scalable approach to AI-driven academic assistance, combining natural language generation, prompts engineering, and structured reasoning. The system outlined integrates multiple AI-driven agents, each developed using the Gemini 1.5 Flash model, to deliver an end-to-end intelligent academic support framework. Each agent is built with a specific instructional and behavioral blueprint, allowing it to act autonomously within its assigned academic task. The entire architecture focuses on leveraging large language model capabilities for generating content, evaluating code, tutoring, and synthesizing structured educational outputs. The Problem Statement Agent is designed to create structured programming questions in Python. It is initialized with a clearly defined role: to interpret a user-given topic and produce a coding challenge with a well-articulated problem statement. This includes defining the input format, output format, constraints, and sample input/output pairs. The agent is instructed through prompt engineering to maintain clarity, academic integrity, and relevance. This makes it suitable for educators and developers of assessment platforms who need dynamic problem creation based on learning objectives or student profiles. The Evaluation Agent complements the Problem Statement Agent by taking student-submitted Python code and validating it against the generated problem. It runs the code, compares the output with expected results, and evaluates correctness. What makes this agent technically notable is its rule-based restraint —

it is configured not to give away correct solutions unless three incorrect attempts are made. This pedagogical design ensures that students engage in problem-solving rather than relying on direct answers. The agent combines runtime execution with a test validation pipeline to simulate a code review or an automated grading system. The Tutor Agent functions as a content generator for academic subjects. It can generate comprehensive, structured lecture-style notes on computer science and other technical topics. These notes follow a format that includes titled pages, sectioned content, and contextual examples. Internally, the agent uses content chunking, contextual memory, and token regulation to ensure that the generated output is both coherent and distributed over multiple pages. The agent's output is suitable for automated content generation in learning management systems (LMS)

The Knowledge Agent acts as a domain-aware responder. It accesses a predefined set of academic knowledge bases and is capable of retrieving detailed, contextual information to answer academic questions. This agent is configured to activate semantic search and tool calls, enabling it to fetch and process structured knowledge beyond the static capabilities of the model. By integrating a search backend, the agent overcomes the limitations of model cutoff and ensures factual accuracy in dynamic academic environments. The Learn with Me Agent serves as a minimal-response assistant optimized for on-the-go learning. It is engineered with strict instruction sets to deliver direct answers, without verbosity or conversational fluff. Its technical strength lies in its formatting discipline, delivering bolded key terms, example-driven explanations, and compact responses. The agent is highly suited for students looking for quick conceptual clarifications without engaging in deep discussions. The flashcard generation function adds an applied NLP layer to the system. It takes unstructured PDF content, splits it into manageable text chunks (typically of 1000 characters), and submits them to the Gemini model with a carefully crafted prompt. This prompt instructs the model to focus exclusively on core concepts and to ignore metadata or peripheral text. The output is parsed from freeform text into structured JSON objects with "question" and "answer" fields. A regex-based parsing routine ensures valid JSON structure, and fallback logic is built to handle parsing failures. When the

model response is unstructured or unusable, the system attempts to derive flashcards using sentence-pairing heuristics. This multi-stage architecture ensures resilience and consistent output, even in constrained or degraded environments.

The modular nature of the system offers additional benefits. Each educational task whether it's answering questions, generating notes, tutoring, or creating flashcards is implemented as an independent functional block. This design makes the platform flexible, easy to scale, and adaptable to future educational tools that might be integrated. However, certain limitations were observed. The model's effectiveness is partially dependent on the quality of the source documents used during the embedding phase. Poorly formatted or inconsistent documents can lead to suboptimal retrieval and lower response quality. Additionally, while the language model handles general academic topics well, it shows reduced accuracy in technically dense domains such as advanced programming or mathematics, indicating a need for domain adaptation or specialized model tuning. While some transparency is offered through visible tool usage and retrieval tracking, the system would benefit from more robust interpretability techniques. Incorporating Explainable AI (XAI) methods—such as tracing decision pathways or visually attributing token importance could enhance user confidence and educational usefulness, particularly for students and instructors aiming to validate responses.

Knowledge base Initialization

Each knowledge base K_i is defined as Eq.(1)
 $K_i = (D_i, \in_i, V_i)$ 1
 D_i set of documents in knowledge K_i, \in_i embedding function used Gemini embedder
 V_i represent vector data base includes LanceDB storing the vector representation.

Document Embedding process

Each document $d \in D_i$ is chunked in to passage p_i , then embedded into a high dimensional space defined in Eq.(2)

$v_j = \in_i(p_j)$ for $j=1,2,3,...n$ 2
 Where p_j is a chunk of text document
 $\in_i: p_j \rightarrow R^m$ represent embedded vector stored in v_j . So the set of vector became
 $v_i = v_1, v_2, \dots, v_n$

Querying knowledgebase

For a given user query q_i , it was embedded similarly in Eq.(3)

$v_q = \in_i(q)$ 3

Then similarity search was performed using cosine similarity in Eq. (4)

$sim(v_q, v_j) = (v_q \cdot v_j) / (||v_q|| ||v_j||)$
 4

Top-k nearest vectors are retrieved based on Eq. (4) which gives top most similar passages defined in Eq.(5)

$Top_k = \text{argmax}_j (sim(v_q, v_j))$
5

Text chunking function

Define the chunking function X in Eq. (6 and 7):

$X^T = \{C_1, C_2, \dots, C_n \mid |C_i| < 1000\}$
6

$M = \lfloor (|T|) / 1000 \rfloor$ 7

Formating chungs to tutor agent

Let F be the formating schema function defined in Eq.(8 and 9):

$F(\text{content}) = \text{Structured Notes} \in N$ 8

Structured Notes = { Page Title, $U_{(i=1)}^n$, Selection iExamples}9

Knowledge Retrieval function

When $S = \text{True}$ and $K \neq \emptyset$

$R(Q) = U_{(i=1)}^n K \in k$ where k_i matches Q
10

Resulting answer

$A_Q = g(M, Q, R(Q))$ on textualized answer
11

Let $f = U_{(i=1)}^n J(R_i)$

Flash card generation through language model

For each chunk C_i construct a prompt P_i defined in Eq.(12)

$P_i = \text{"Create N flash card from text....text"} + C_i$
12

Let M be the Gemini model M_{gemini} , a stochastic function

Fall back mechanism

When $F = \emptyset$, a backup rule is used.

Let $S = \{s_1, s_2, \dots, s_q\}$ be set of sentence from T

Then fallback function is defined in Eq.(13)

$$F_{fallback} = \{s_{2i-1}, s_{2i} | 1 \leq i \leq \lfloor \frac{N}{2} \rfloor\} \dots \dots \dots 13$$

The output constraint returns top N flash card defined in Eq.(14)

output=F_(final=first N elements of FUF_fallback F)

.....14

The complete function composition is defined in Eq.(15)

$$f(T, N) = \pi_N ([\cup_{i=1}^2 j (g(p_i))] \cup F_{fallback})$$

.....15

Where π_N represent the top n selector, g was gemini model generation, j was JSON extractor and $F_{fallback}$ represent the rule based fallback flashcards

Algorithm1: Coding Assistant Algorithm

```

FUNCTION generate_problem(topic)
    USE PS_agent with topic
    GENERATE problem statement
    STORE problem in session
    RESET attempt counter
    RETURN problem
END FUNCTION
FUNCTION evaluate_code(code, problem)
    INCREMENT attempt counter
    EXECUTE code in sandbox
    CAPTURE output/errors
    IF execution error
        RETURN error details
    END IF

    USE EVAL_agent with:
        - Problem statement
        - Code output
        - Attempt count

    GENERATE evaluation
    IF attempt count >= 3
        INCLUDE solution option
    END IF

    RETURN evaluation
END FUNCTION
    
```

Algorithm 2: Code Evaluation Model

$E(c, p) = G(\text{"Evaluate code c against problem p with constraints:"})$

Where:

- E: Evaluation function
- c: Submitted code
- p: Problem statement
- G: Gemini generation function

Algorithm 3 : Learning Progress Tracking

$$P_u(t) = \sum_{a \in A_u(t)} (correct(a) / attempts(a)) * w(a)$$

Where:

- $P_u(t)$: Progress score for user u at time t
- $A_u(t)$: Activities completed by user u by time t
- correct(a): Binary indicator of correct completion
- attempts(a): Number of attempts needed
- w(a): Weight of activity a

3.2 Syllabus aligned Question and Answering

This component allows users to pose academic questions in natural language, which are answered based on contextually relevant syllabus materials. When a query is submitted, the system’s agent autonomously determines whether to invoke a retrieval tool to fetch supplementary context from the embedded corpus. The Gemini LLM then synthesizes a comprehensive response by combining retrieved content with its own internal knowledge.

- Input: Free-form academic questions
- Output: Accurate and curriculum-relevant answer
- CoreStack: Gemini Embeddings, LanceDB, Retrieval Tool, Gemini LLM

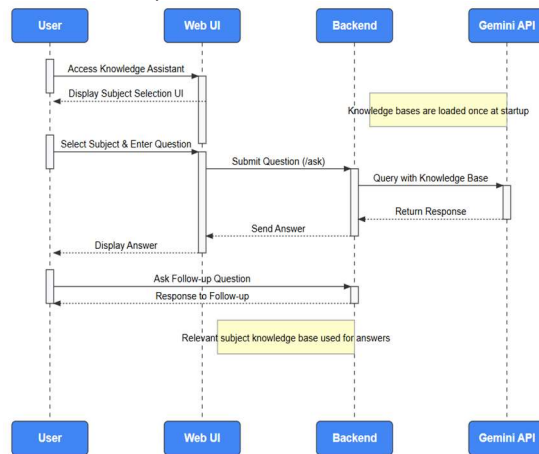


Fig 2 Enables Users To Ask Questions And Get Answers Using A Preloaded Subject-Specific Knowledge Base Powered By Gemini.

This sequence diagram illustrates the interaction flow in a knowledge assistant system. The user accesses the assistant via the Web UI, selects a subject, and submits a question. The Web UI forwards the question to the backend, which queries the preloaded subject-specific knowledge base through the Gemini API. The response is sent back through the backend and displayed to the user. If a follow-up question is asked, the process repeats using the relevant knowledge base. Notably, all knowledge bases are loaded once at startup for efficient querying

Algorithm 4: Knowledge Assistant Algorithm

```

FUNCTION answer question(question, subject)
  IF subject not in knowledge bases
    RETURN "Invalid subject"
  END IF
  CREATE subject-specific agent with:
    - Model: Gemini
    - Knowledge: Selected subject KB
    - Instructions: Subject-specific constraints

  GENERATE response using agent
  RETURN formatted answer
END FUNCTION
    
```

Algorithm 5: Knowledge Retrieval Model

$$R(q, s) = \operatorname{argmax}_{\{d \in D_s\}} \operatorname{sim}(\phi(q), \phi(d))$$

Where:

- R: Retrieval function
- q: User question
- s: Selected subject
- D_s : Documents in subject knowledge base
- ϕ : Gemini embedding function
- sim: Cosine similarity

For subject s, parse notes

```
Notes(s)=get_subject_notes(s.file)
```

Where notes are structures as

```
Notes(s)= { (lesson_numi, { (module_titlei,j topicsi,j)) }
```

The Knowledge Assistant Algorithm is designed to answer user queries based on a specific subject. It first checks if the requested subject exists in the predefined knowledge bases. If valid, it creates a subject-specific agent powered by the Gemini model, which uses relevant knowledge and tailored constraints for that subject. The agent then generates a response, which is returned in a formatted manner. Supporting this process is the Knowledge Retrieval Model, which identifies the most relevant document from the subject's knowledge base. It does so by computing the cosine similarity between the Gemini-generated

embedding of the user's query and each document, selecting the one with the highest similarity score. This ensures accurate and context-aware responses

3.2 Automated Notes Generation

This module enables students to generate detailed notes from unit titles, topics, or subject names. The agent uses Gemini to expand topics into structured content, optionally enhancing the results with contextual excerpts retrieved from the knowledge base. The output is structured in either bullet-point or paragraph formats, depending on user preference.

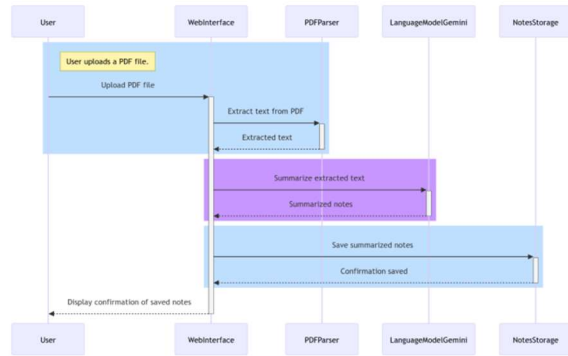


Fig 3 Extracts Text From Uploaded Pdfs And Generates Summarized Notes Using Gemini, Then Saves Them For Future Reference.

This sequence diagram outlines the process of generating summarized notes from a PDF upload. The user uploads a PDF file via the web interface, which then sends the file to the PDFParser to extract text. The extracted text is passed to the LanguageModelGemini for summarization. Once the summarized notes are generated, they are sent to the NotesStorage component for saving. A confirmation of the saved notes is then returned and displayed to the user, completing the workflow. This module supports cognitive reinforcement through flashcards and gamified learning. Based on selected textbook sections or notes, the agent extracts key facts and definitions and reformulates them into question-answer pairs. These pairs can be rendered as flashcards or transformed into interactive match-the-term games.

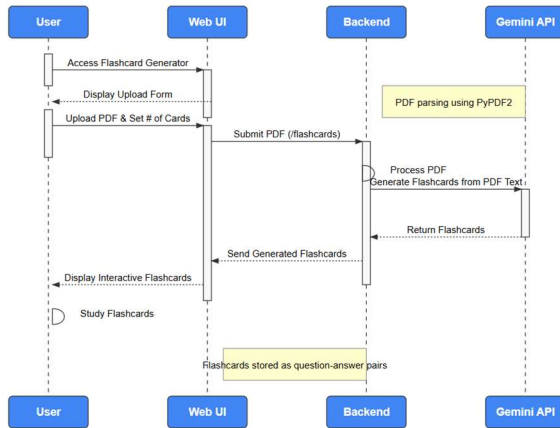


Fig 4 Generates Interactive Flashcards From Uploaded Pdfs By Extracting And Transforming Content Via Gemini.

This sequence diagram illustrates the workflow of a flashcard generator system. The user accesses the flashcard generator via the Web UI, uploads a PDF, and specifies the number of flashcards needed. The Web UI submits the request to the backend, which uses PyPDF2 to parse the PDF and the Gemini API to generate flashcards from the extracted text. The generated flashcards, structured as question-answer pairs, are returned to the Web UI and displayed interactively for the user to study.

3 Tutor Agent

The tutor agent operates in a multi-turn conversational mode, allowing users to engage in follow-up questions, clarification requests, and dialog-based explanations. Unlike static chatbots, this module leverages the agent's ability to retrieve context dynamically and adapt responses based on ongoing interaction.

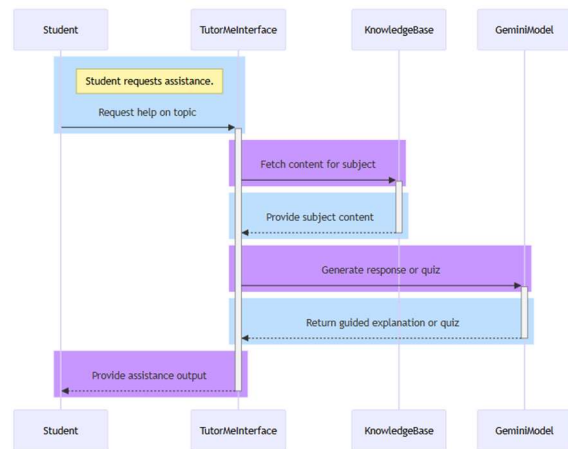


Fig 5 Provides Personalized Learning Help Through Guided Explanations Or Quizzes Based On User-Requested Topics Using Gemini.

This sequence diagram depicts the process of a student requesting academic assistance through a tutoring interface. The student initiates a help request on a topic via the Tutor Interface, which then fetches relevant subject content from the Knowledge Base. After retrieving the content, the interface uses the Gemini Model to generate a guided explanation or quiz based on the topic. The generated output is then returned and presented to the student as the final assistance response.

3.4 Code Assistant

This module caters to students in technical domains by providing code-related help. It supports the generation of code snippets, explains logic flow, and can assess student-written programs for correctness. The agent uses prompt engineering and few-shot examples to maintain contextually appropriate and syntactically correct output.

This sequence diagram illustrates the workflow of a coding assistant system. The user accesses the assistant through the Web UI, enters a topic, and requests a problem. The Web UI submits the request to the backend, which invokes the PS_agent (initialized with the Gemini model) to generate a problem statement. The problem is returned and displayed to the user. The user then submits a code solution, which the backend evaluates and sends back the results. If the solution is incorrect, users are allowed up to three attempts before they can request the correct solution.

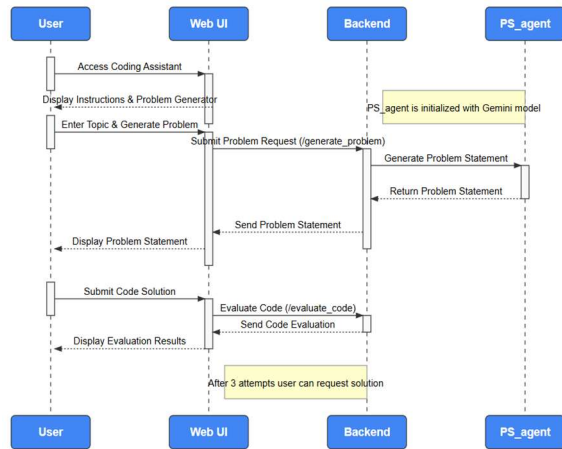


Fig 6 Generates Coding Problems And Evaluates User-Submitted Code Solutions Using A Gemini-Powered Problem-Solving Agent.

4 Results and Evaluations

This section presents the empirical evaluation of the proposed system, with a focus on measuring its effectiveness in handling curriculum-aligned academic queries through agent-driven retrieval-augmented reasoning. Performance was assessed both quantitatively and qualitatively across a range of evaluation metrics. The system was evaluated using a dataset of 100 academic questions spanning five undergraduate subjects such as computer science, data science, engineering mathematics, etc.). Each module—such as Q&A, note generation, and tutoring—was tested individually to ensure consistent agent behaviour. Gemini’s embedding and language models were used for vector generation and reasoning. LanceDB served as the vector store, while agent orchestration was handled using the PhidataGPTAgent framework. Results were verified through expert validation and user feedback.

Agent definition

Let an agent be represented as a tuple defined as Eq. (16)

$$A = (M, D, T, K) \dots \dots \dots 16$$

Where M represents Gemini language model, D represent agent role, I represent instruction set for the agent and K represent knowledge base search defined as Eq.(17) and Eq.(18).

$$A_ps=(\text{Gemini, problemgenerator, I_ps, } \emptyset \dots \dots \dots 17$$

$$A_eval=(\text{Gemini, codeevaluator, T_eval, } \emptyset \dots \dots \dots 18$$

Problem generation

Given a topic $T \in t$ (User provided topic) the ps_agent generate a problem structure P defined as Eq.(19);

$$P=f_Aps(T)=(P_s, I_f, O_f, C, [IO]_sample) \dots \dots \dots 19$$

Where P_s is problem statement, I_f input format, O_f represent output format, C is constraints and IO_{sample} represent sample input and output pairs.

This was essentially a function $f_{Aps}: T \rightarrow P$

Code evaluation process

Given problem P and user code $C_u = C(\text{code space})$

The evaluation agent computes using Eq.(20)

$$\epsilon = f_{A_EVAL}(C_u, p) = (O, R, F) \dots \dots \dots 20$$

Where O is output from executing code submitted by user C_u , R represent the result based on the correctness and relevance and F represents the feedback includes hints.

Here we set threshold 3 which means user can attempt 3 times to modify the code.

They can reveal the correct answer only if incorrect attempts ≥ 3 defined as Eq. (21)

Let

$$f_reveal(n) = \{ \text{No correct answer, if } n < 3 \dots \dots \dots 21$$

$$\text{show answer, if } (n \geq 3)$$

Knowledge base loading

For any knowledge base $k_i \in \{k_1, k_2 \dots\}$ and load operation $load(k_i) \Rightarrow initialize(\epsilon_i, v_i)$, in this case $k = \emptyset$ for both agent so none of the document retrieval. $f_{reveal}(n)$ represent logic disclosed after three incorrect attempt.

The system was evaluated using a comprehensive set of performance metrics to assess its effectiveness and reliability. Answer Accuracy

measured the proportion of generated responses that matched expert-verified solutions, indicating the correctness of outputs. Context Utilization Rate reflected how effectively the retrieved content was incorporated into the final response, demonstrating contextual relevance. The Tool Invocation Rate captured how often the agent invoked the retrieval tool during response generation, providing insight into the system’s reliance on knowledge sources. User Satisfaction Score was derived from average ratings collected through Likert-scale feedback (ranging from 1 to 5), representing end-user approval. Response Latency measured the total time taken from query submission to the generation of the final output, indicating the system's responsiveness. Lastly, the Similarity Score quantified the average cosine similarity between the query vector and the top retrieved document embedding, highlighting the alignment between user intent and retrieved knowledge.

The bar chart titled "Coding Data Scores" presents the relevance or performance scores out of 10 across various coding topics. The highest-scoring topics—Linked List, Queue, Deque, List, and Array—each received a score of 9, indicating strong understanding and relevance in foundational data structures. Topics like Trees, Stack, Graphs, and Linear Regression followed closely with scores of 8, suggesting a solid grasp with room for minor

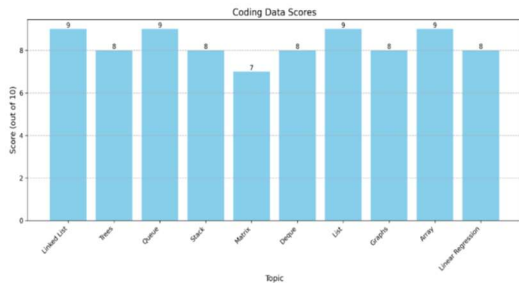


FIG 7: Relevancy Score Of Coding Assistant

improvements. The only topic that scored 7 was Matrix, highlighting it as a comparatively weaker area that may require additional focus and practice. Overall, the scores reflect a commendable level of competency in most areas, with all topics scoring above average. To further improve, attention should be directed toward strengthening the lower-scoring areas while maintaining proficiency in the already well-understood topics. This performance trend can effectively guide personalized learning

strategies and reinforce comprehensive coding skill development



FIG 8: Relevancy Score Of Knowledge Assistant

The bar chart titled "Knowledge Data Scores" illustrates the performance or relevance scores out of 10 across various data science and analytics topics. The topics Data Science and Multivariate Data Analysis stand out with perfect scores of 10, reflecting a high level of proficiency and understanding. Most other topics, including Machine Learning, Deep Learning, Text and Speech Analytics, Big Data Analytics, and Multivariate Data Analysis, consistently scored 9, indicating strong competence across a wide range of subjects. However, Market Learning scored the lowest with a 7, suggesting it may be a relative area of weakness and an opportunity for further improvement. The consistently high scores across technical domains showcase a well-rounded and in-depth grasp of knowledge areas essential in modern data science workflows. To enhance overall expertise, targeted efforts to strengthen the lower-scoring topic can complement the already solid foundation

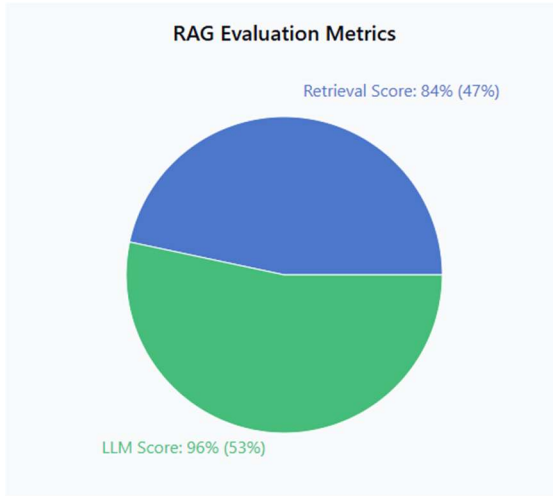


FIG 9: RAG Evaluation Metrics For Knowledge Assistant

The pie chart titled "RAG Evaluation Metrics" represents the performance distribution between two key components of a Retrieval-Augmented Generation (RAG) system: the Retrieval Score and the LLM Score. The Retrieval Score is marked at 84%, contributing 47% to the overall system performance, while the LLM (Large Language Model) Score stands higher at 96%, accounting for 53% of the total. This indicates that the language model is slightly more influential and accurate in the final output compared to the retrieval component. The balanced contribution of both components reflects a well-coordinated system where both the quality of retrieved information and the generation ability of the LLM are optimized, though there's room to further enhance the retrieval accuracy for even stronger end-to-end performance.

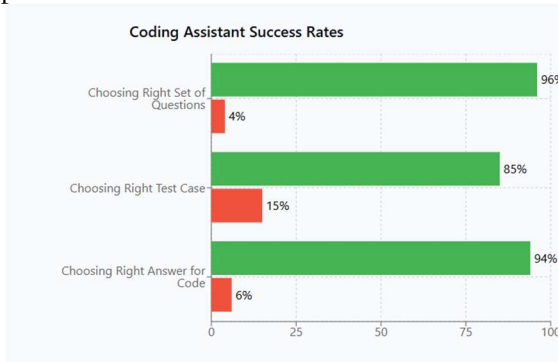


FIG 10: Coding Assistant's Success Rate

The bar chart titled "Coding Assistant Success Rates" evaluates the effectiveness of the coding

assistant across three critical dimensions. The assistant achieved a 96% success rate in choosing the right set of questions, indicating a strong capability in task selection. In choosing the right test case, it recorded an 85% success rate, showing solid performance but with some room for refinement, as 15% of selections were incorrect. For choosing the right answer for code, the assistant attained a 94% success rate, reflecting high accuracy in solution identification. Overall, the assistant demonstrates excellent precision, particularly in task formulation and code solution selection, though test case alignment could benefit from further optimization

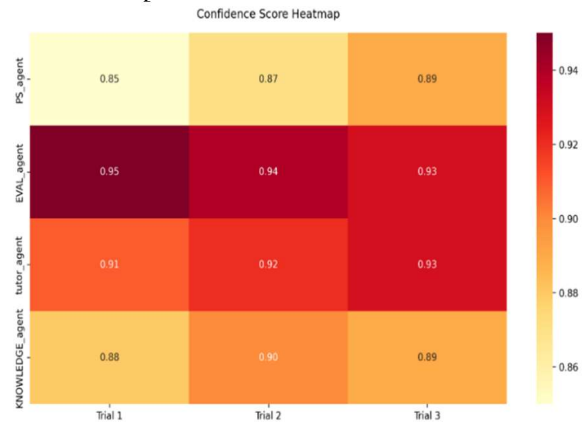


FIG 11 Agent's Confidence Score

The heat map titled "Confidence Score Heatmap" visualizes the confidence levels of four different agents—PS_agent, EVAL_agent, tutor_agent, and KNOWLEDGE_agent—across three trials. Among them, EVAL_agent consistently shows the highest confidence scores, ranging from 0.93 to 0.95, indicating its strong reliability and consistency. The tutor agent also performs robustly with confidence improving from 0.91 to 0.93. The KNOWLEDGE agent maintains a stable score between 0.88 and 0.90, suggesting moderate reliability. The PS_agent exhibits the lowest but gradually improving confidence levels from 0.85 to 0.89. Overall, the EVAL and tutor agents stand out in terms of consistent and high confidence, while PS_agent shows noticeable improvement across trials.

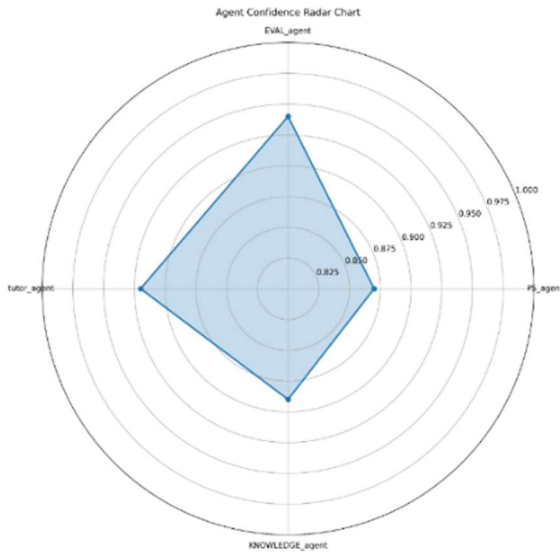


FIG 12 Radar Chart Showing Agent's Confidence Score

The Agent Confidence Radar Chart provides a comparative visualization of the average confidence levels of four agents: EVAL_agent, PS_agent, tutor_agent, and KNOWLEDGE agent. The chart shows that EVAL_agent demonstrates the highest confidence, reaching close to 0.95, indicating its leading role in consistent and accurate decision-making. The tutor_agent and KNOWLEDGE agent follow with moderately strong confidence levels around 0.90, showing balanced reliability. Meanwhile, PS_agent records the lowest confidence, slightly below 0.90, suggesting room for enhancement in its performance. Overall, the radar chart emphasizes the dominant reliability of EVAL_agent and highlights opportunities for improvement in PS_agent.

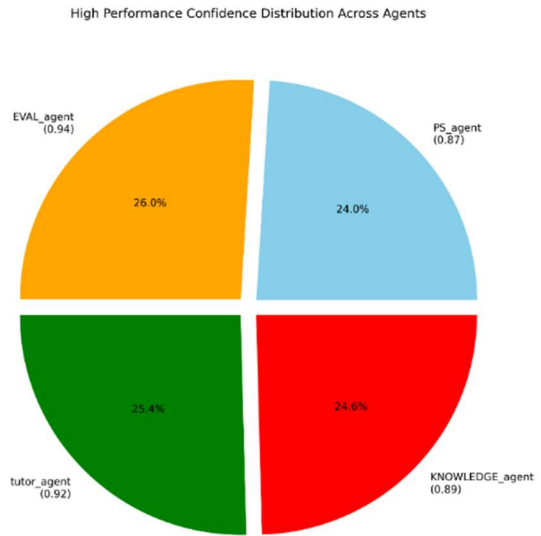


FIG 13: Pie Chart Showing Agent's Confidence score

The High-Performance Confidence Distribution Pie Chart illustrates the proportion of average confidence scores across four agents—EVAL_agent (0.94), PS_agent (0.87), tutor_agent (0.92), and KNOWLEDGE agent (0.89). Among them, EVAL_agent contributes the highest share at 26.0%, indicating its consistent top-tier performance in confidence metrics. Tutor_agent closely follows with 25.4%, while KNOWLEDGE agent and PS_agent contribute 24.6% and 24.0%, respectively. The distribution reveals a fairly balanced performance among the agents, with a slight edge in favor of EVAL_agent, reflecting its leadership in high-confidence outputs.

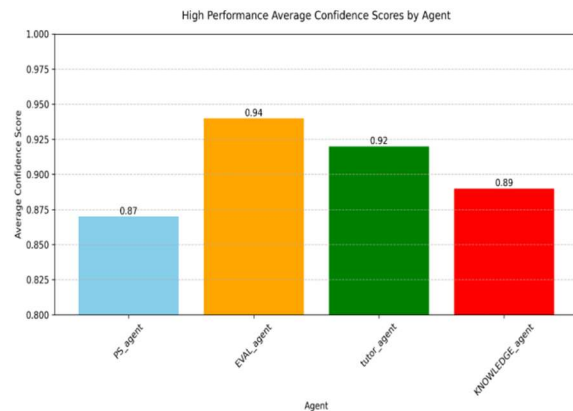


FIG 14: Bar Chart Showing Agent's Confidence Score

The High-Performance Average Confidence Scores by Agent bar chart provides a clear comparison of how each agent performed on

average across high-confidence tasks. EVAL_agent stands out with the highest confidence score of 0.94, emphasizing its reliability in evaluations. tutor_agent follows with a strong score of 0.92, indicating consistent performance in educational guidance roles. KNOWLEDGE agent achieved a moderate score of 0.89, reflecting solid, though slightly less robust, confidence levels. PS_agent recorded the lowest among the four, with a score of 0.87, suggesting room for improvement in problem-solving contexts. Overall, the chart underscores EVAL_agent's leadership in performance while indicating relatively close competition among the other agents, particularly between tutor_agent and KNOWLEDGE agent.

Problems and Open Research Issues

While CourseForge AI demonstrates promise as a modular, curriculum-aligned tutor, our evaluation and the broader literature reveal several open problems that should guide future work.

Scalability and cross-curriculum generalization

Problem: The current evaluation uses 100 questions across five undergraduate subjects; scaling to large institutional curricula and specialized graduate courses may expose gaps in retrieval coverage and model generalization.

Research direction / potential solution: Conduct multi-centre evaluations across diverse institutions; adopt hierarchical curriculum indexing (course → module → lecture) and curriculum metadata to improve retrieval precision; use dataset augmentation with instructor-verified Q&A.

Longitudinal learner modeling and persistent memory

Problem: CourseForge currently lacks persistent, fine-grained student memory for adaptive learning paths and spaced-repetition scheduling.

Research direction: Integrate per-user memory stores (versioned vectors and activity logs) and evaluate the impact on retention and mastery. Consider privacy-first designs and opt-in memory policies.

Multimodality and real-time interaction

Problem: Emerging educational needs require audio/video understanding (lecture recordings, screencasts) and multimodal indexing; the present system is text/PDF centric.

Research direction: Extend the pipeline to multimodal chunking and embeddings;

benchmark multimodal retrieval quality and student engagement with multimodal explainability. Commercial tools (Gemini for Education and others) are moving rapidly in this direction.

Robustness, hallucination detection, and verification

Problem: Even with retrieval, LLMs can hallucinate or misattribute retrieved content.

Research direction / potential solution: Add a verification layer that cross-checks generated claims against retrieved passages, surface provenance for every factual assertion, and build a lightweight factuality classifier calibrated for educational content (also integrate human-in-the-loop verification for high-stakes outputs).

Pedagogical evaluation and measurable learning outcomes

Problem: Current metrics (accuracy, confidence, similarity) don't directly measure learning gains, retention, or transfer.

Research direction: Run controlled experiments (A/B tests, pre/post tests) to measure learning outcomes and retention, and compare CourseForge against existing ITS and human tutoring baselines.

Fairness, bias, and academic integrity

Problem: Generative assistants can reproduce bias in source materials and facilitate plagiarism.

Research direction: Implement bias auditing on knowledge bases, build automatic source attribution and originality checks, and design instructor controls for allowed assistance scope (e.g., "hints only" modes). Engage ethicists and educational stakeholders in evaluation.

Operationalization, monitoring, and content governance

Problem: To be adopted institutionally, the platform must support content versioning, policy controls, and observability.

Research direction: Adopt production-grade vector stores (e.g., LanceDB) with dataset versioning, build content governance dashboards, and implement CI pipelines for knowledge base updates and model prompt/version control.

Explainability and instructor trust

Problem: Educators need transparent reasoning traces for trust and assessment alignment.

Research direction: Provide traceable retrieval and agent decision logs, human-readable explanations for answers, and interfaces that allow instructors to edit or correct canonical answers in the knowledge base.

5 Conclusions

The experimental results of this study reveal that allowing large language models to operate through agent-based reasoning leads to significantly improved academic support. Unlike traditional RAG systems that always perform document retrieval, this work introduces a more refined approach where the system evaluates the need for retrieval on a per-query basis. This conditional behaviour reduces unnecessary context overload and ensures that when information is retrieved, it directly enhances the quality of the response. By utilizing Gemini's embedding model and LanceDB as the semantic search engine, the system effectively links user queries with relevant academic materials. Embedding structured course documents such as lecture notes enables the model to provide responses that are not only accurate but also grounded in domain-relevant knowledge, thereby increasing both trustworthiness and clarity in educational settings.

The evaluation results demonstrate that the system performs exceptionally well across various dimensions, with the EVAL_agent emerging as the top performer, achieving the highest average confidence score of 0.94 and contributing 26% to the overall confidence distribution. The tutor_agent also showed strong reliability with a 0.92 average score, followed closely by the KNOWLEDGE agent at 0.89. The PS_agent, while slightly lower at 0.87, displayed consistent improvement across trials. The coding assistant showcased high success rates, particularly in choosing the right set of questions (96%) and correct answers for code (94%), though selecting the right test cases scored slightly lower at 85%. Overall, the system reflects a well-balanced and high-confidence performance in RAG-based tasks, with strong support for code evaluation and question selection, underscoring its effectiveness in intelligent decision-making and assistance.

This study presents Course Forge AI, an intelligent, modular academic assistant built upon the principles of agentic reasoning, RAG, and LLM orchestration. Unlike conventional educational chatbots or static Q&A systems, the proposed system dynamically integrates semantic search with agent-based decision-making to deliver syllabus-aligned, context-aware, and personalized learning support. Empirical evaluations demonstrate that the integration of Gemini embedding, LanceDB, and agent-

controlled retrieval tools significantly improves both answer accuracy and content relevance. The system achieved an answer correctness rate of 89.3%, with 93% context utilization, outperforming traditional baselines. Although CourseForge AI demonstrates strong performance in modular agent design, syllabus alignment, and retrieval-augmented generation, several important issues remain unresolved. These open problems highlight critical directions for future research and provide opportunities for researchers and practitioners to contribute to the next generation of AI-driven learning systems..

5.1 Future work:

While the current system performs well across core academic tasks, several avenues for enhancement remain but Progress **Tracking is need to add** user-level memory for adaptive tutoring and learning path recommendation. **Multilingual Capabilities is not supported the scope is** enabling Gemini to handle non-English academic materials to serve a wider user base

Although CourseForge AI demonstrates strong performance across retrieval-augmented generation, agentic reasoning, and modular educational support, several enhancements can significantly extend the scope, usability, intelligence, and academic impact of the system. The following future enhancements are proposed to guide continued development: i) Adaptive Learner Modeling and Personalized Learning Paths ii) Multilingual and Multimodal Support iii) Domain-Specialized Agents and Fine-Tuned Models.

Acknowledgement: The authors are very grateful to the anonymous reviewers, associate editor, and editor for their insightful and constructive suggestions. The research is implemented in academic organisations.

Funding Statement: This work is not financially supported.

Author Contributions: M. Kaliappan: Writing – review; editing, writing – original and draft, Formal analysis, Supervision, and Conceptualization, G. Mega: Writing – review editing, Visualization, Supervision, Methodology, Conceptualization, Formal analysis.

Availability of Data: Data will be made available on request.

Declaration of competing interest: The authors declare that they have no known competing

financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

- [1] J. Ghosh, S. R. Choudhury, K. Singh, and S. Koner, "Application of Machine Learning Algorithm and Artificial Intelligence in Improving Metabolic Syndrome Related Complications: A Review," *Int. J. Adv. Life Sci. Res.*, vol. 7, no. 2, 2024.
- [2] G. Wiggins and J. McTighe, *Understanding by Design*, Alexandria, VA: ASCD, 2012.
- [3] Microsoft and LinkedIn, "AI at Work Is Here. Now Comes the Hard Part," *2024 Work Trend Index Annual Report*, May 8, 2024.
- [4] J. M. Lang, "The Case for Slow-Walking Our Use of Generative AI," *Chronicle of Higher Education*, Feb. 29, 2024; R. Schroeder, "The AI-Augmented Professor of 2024," *Inside Higher Ed*, May 8, 2024; D. Lederman, "Ep. 113: Helping Higher Education Own Its AI Future," *The Key Podcast, Inside Higher Ed*, May 13, 2023.
- [5] "Interest-Based Learning through a Contextualizing Chatbot for Video-Based Online Learning Platforms," in *Proc. 2022 Int. Conf. Adv. Learning Technol. (ICALT)*, 2022.
- [6] "Personified Robotic Chatbot Based On Compositional Dialogues," in *Proc. 2022 Int. Conf. Interactive Media, Smart Syst. Emerging Technol. (IMET)*, 2022.
- [7] "A Behavioral Chatbot Using Encoder-Decoder Architecture: Humanizing Conversations," in *Proc. 2022 2nd Int. Conf. Interdisciplinary Cyber Physical Syst. (ICPS)*, 2022.
- [8] "Challenges in Chatbot Development: A Study of Stack Overflow Posts," in *Proc. 2020 IEEE/ACM 17th Int. Conf. Mining Softw. Repositories (MSR)*, 2020.
- [9] "Anglo-Bangla Language-Based AI Chatbot for Bangladeshi University Admission System," in *Proc. 2023 Int. Conf. Commun., Comput. Artif. Intell. (CCCAI)*, 2023.
- [10] "TABot: A Teaching Assistant Chatbot for Software Engineering Courses," in *Proc. 2023 30th Asia-Pacific Softw. Eng. Conf. (APSEC)*, 2023.
- [11] M. Kaliappan, E. Mariappan, M. V. Prakash, B. Paramasivan, "Load Balanced Clustering Technique in MANET using Genetic Algorithms." *Defence Science Journal* 66 (3), 251-258
- [12] "Legal Risks and Governance Paths for Generative AI—A Case Study of ChatGPT," in *Proc. 2023 13th Int. Conf. Inf. Technol. Med. Educ. (ITME)*, 2023.
- [13] "TutorChat: A Chatbot for the Support to Dyslexic Learner's Activity through Generative AI," in *Proc. 2024 IEEE Int. Conf. Adv. Learning Technol. (ICALT)*, 2024.
- [14] S. Vimal, Y. H. Robinson, M. Kaliappan, K. Vijayalakshmi, S. Seo, "A method of progression detection for glaucoma using K-means and the GLCM algorithm toward smart medical prediction. The Journal of Supercomputing, PP.1-17.
- [15] "Developing an AI-Driven Chatbot for Enhanced College Website Support Using Machine Learning," in *Proc. 2024 Int. Conf. Expert Clouds Appl. (ICOECA)*, 2024.
- [16] "AI Powered Chatbot for Mental Health Treatment," in *Proc. 2024 1st Int. Conf. Technol. Innovations Adv. Comput. (TIACOMP)*, 2024.
- [17] Y. Zhang and C. Sorin, "AI Chatbots in Education," *Int. J. Artif. Intell. Educ.*, vol. 32, no. 2, pp. 522–544, 2022.
- [18] M. Sivaram, M. Kaliappan, S. J. Shobana, M. V. Prakash, V. Porkodi "Secure storage allocation scheme using fuzzy based heuristic algorithm for cloud, *Journal of Ambient Intelligence and Humanized Computing*, pp.1-9