International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

DSA Tutor: A Hybrid AI Learning Platform Combining an Intelligent Tutoring System with Retrieval-Augmented Generation for Unified Computer Science Education

Prathit Mehul Panchal[1], Dr. Kalaavathi B[2]

[1]B.Tech. Computer Science and Engineering, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India
[2]Professor (Grade 1), Department of IoT, School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India

Abstract

Purpose: This paper presents DSA Tutor, a unified AI-powered learning platform that addresses the long-standing fragmentation in Data Structures and Algorithms (DSA) education, where learners routinely switch between disconnected tools for theoretical instruction, coding practice, knowledge assessment, and reference reading. The work aims to demonstrate how two separate paradigms, Intelligent Tutoring Systems and Retrieval-Augmented Generation, can be integrated into a single coherent environment that serves computer science undergraduates preparing for technical interviews and academic coursework.

Design/methodology/approach: The platform implements a hybrid architecture composed of two subsystems. The Intelligent Tutoring System component offers topic-based explanations, AI-generated coding problems with deterministic classification of submissions as optimal, brute-force, or incorrect, and difficulty-graded multiple-choice tests with persistent per-user scoring. The Retrieval-Augmented Generation component, named RAGenius, ingests DOCX files with Tesseract-based OCR fallback for scanned content, chunks them using recursive character splitting, embeds them through all-MiniLM-L6-v2, indexes them in a Weaviate vector store, and answers questions using hybrid BM25 and dense vector retrieval with traceable citations. A three-tier language model fallback architecture routes traffic from Google Gemini through Ollama Cloud to a locally hosted Llama 3 instance, ensuring continuous availability during rate limits or network failures. The system was implemented using FastAPI, React 19 with Vite 7, SQLite, Redis, and Docker Compose.

Findings: Empirical evaluation was conducted across three dimensions. Intent classification reached 97% accuracy on a manually labelled set of 100 messages, outperforming the general-purpose conversational baseline of 70-75%. The code evaluator achieved 99% accuracy on a curated benchmark of 50 submissions that spanned optimal, brute-force, and incorrect categories. Hybrid retrieval achieved 91% top-5 relevance on a 50-query corpus, representing a 17 percentage point improvement over a vector-only baseline measured on the same data. Mean response latency was 3.2 seconds for Tier 1 and 7.5 seconds for the local Tier 3 fallback.

Practical implications: DSA Tutor unifies activities previously scattered across multiple platforms, provides structured and deterministic code feedback that general chatbots cannot match, preserves learner

state through persistent score tracking, and remains operational even when external LLM providers become unavailable. These properties make the system directly usable by both individual learners and educational institutions, especially in settings where network reliability or API budgets are constrained.
Originality/value: This work is, to the authors knowledge, the first to combine Intelligent Tutoring System modes with a Retrieval-Augmented Generation pipeline under a shared session context for programming education. The three-tier LLM fallback mechanism is uncommon in student-facing AI tools and contributes a practical template for building resilient educational technology on free-tier and open-source infrastructure.

Keywords: Intelligent Tutoring Systems, Retrieval-Augmented Generation, Large Language Models, Hybrid Search, Code Evaluation, AI in Education, Educational Technology

Article Classification: Research paper

1. Introduction

The use of computers as instructional aids dates back several decades. Early Intelligent Tutoring Systems (ITS), developed in university research labs during the 1980s, attempted to simulate the behaviour of a human tutor by modelling a learner's knowledge and adapting instruction in real time. Although effective in narrow domains such as algebra or beginner programming, classical ITS implementations were notoriously expensive to build. Each new subject required a hand-crafted domain model, a rule-based pedagogical engine, and a carefully authored exercise set, work that typically ran into the thousands of person-hours.

Two recent shifts have changed the picture. First, large language models (LLMs) such as GPT-4, Gemini, and the Llama family can now generate coherent technical explanations, solve coding problems, and engage in multi-turn dialogue without any hand-engineered knowledge base. The model itself absorbs much of what used to be the domain and pedagogical layer of a traditional ITS. Second, Retrieval-Augmented Generation (RAG), introduced by Lewis et al. (2020), made it possible to ground an LLM's output in user-supplied documents and produce citations back to the source material, directly addressing the hallucination problem that had previously held LLMs back from serious educational use.

These two threads, however, have largely been pursued in isolation. Most existing AI tutoring tools are general-purpose chatbots without a structured pedagogical framework, and most RAG systems are document question-answering widgets disconnected from any learning workflow. Computer science students preparing for technical interviews are caught in the middle. They watch lectures on video platforms, practise problems on online judges, take quizzes on a third site, and read research papers on a fourth, with nothing to tie those activities together.

This paper presents DSA Tutor, a single integrated platform that brings the ITS and RAG paradigms together for the specific use case of DSA education. The system supports four interaction modes, namely topic-based learning, AI-generated coding practice with deterministic evaluation, MCQ-based testing with persistent scoring, and open-domain conversation, alongside a document intelligence module called RAGenius that handles DOCX ingestion, OCR fallback, semantic embedding, hybrid retrieval, and cited answer generation. A three-tier LLM fallback architecture (Gemini to Ollama Cloud to local Llama 3) ensures the system remains functional regardless of API rate limits or network status.

The remainder of this paper is organised as follows. Section 2 surveys the relevant literature on ITS, LLMs

in education, RAG, and AI-assisted programming education. Section 3 identifies the specific gaps DSA Tutor addresses, and Section 4 states the formal problem. Section 5 reviews related platforms. Section 6 describes the system architecture, and Section 7 details the methodology and key algorithms. Section 8 covers testing, Section 9 presents quantitative results, and Section 10 compares DSA Tutor against existing tools. Section 11 describes deployment and operations. Section 12 concludes and outlines future work.

2. Literature Review

The literature relevant to this work spans four overlapping areas: classical and modern ITS, LLMs in education, retrieval-augmented generation, and AI-assisted programming pedagogy. This section organises the prior work along those lines and indicates how each strand shaped the design of DSA Tutor.

2.1 Intelligent Tutoring Systems

Anderson and colleagues (1995) established the foundations of cognitive tutoring through the ACT-R cognitive architecture and the Cognitive Tutor series, reporting student-performance gains of approximately one standard deviation over conventional classroom instruction. VanLehn's (2011) later meta-analysis put the effect size of well-designed step-based ITS at roughly 0.76, close to the gain observed from one-on-one human tutoring. Both studies underscore that fine-grained, conversational scaffolding outperforms passive content delivery. The handler-based modular architecture used in DSA Tutor (separate handlers for learning, coding, and testing) draws directly on this lineage. Each handler plays the role of a specialised tutor for a single mode of interaction.

2.2 Large Language Models in Education

Brown et al. (2020) demonstrated with GPT-3 that pretrained autoregressive models could perform explanation, code generation, and question answering in a few-shot or zero-shot setting, removing the need for domain-specific fine-tuning. Kasneci et al. (2023) examined both the opportunities and the risks of bringing LLMs into education, and stressed the importance of structured scaffolding to keep model output focused on a learning objective. Yan et al. (2024) systematically reviewed practical LLM challenges in education and highlighted hallucination, lack of persistent state, and difficulty in tracking learner progress as recurring obstacles. These findings led directly to two design choices in DSA Tutor, namely structured operating modes rather than free-form chat, and persistent storage of MCQ scores in SQLite so the system can show the learner a real history of their performance.

On the model side, Google DeepMind's Gemini family has shown strong code-generation and long-context reasoning behaviour (Google DeepMind, 2024), which made it the natural choice for the Tier-1 LLM. Touvron et al. (2023) released Llama 2 as an open foundation model that closed much of the gap with proprietary systems and enabled local deployment. The Llama 3 release used in the Tier-3 fallback continues that line of work. Zamfirescu-Pereira et al. (2023) found that non-experts struggle to write effective prompts and produce inconsistent LLM output without a systematic prompting strategy. The SystemPrompts module in DSA Tutor centralises every mode-specific prompt for exactly this reason.

2.3 Retrieval-Augmented Generation

Lewis et al. (2020) introduced RAG as a way to combine parametric knowledge in an LLM with non-parametric retrieval from an external corpus, reporting substantial gains on knowledge-intensive tasks. Reimers and Gurevych (2019) developed Sentence-BERT, the line of work from which the all-MiniLM-L6-v2 embedding model used in RAGenius descends. Robertson and Zaragoza's (2009) BM25 remains a strong baseline for keyword-based retrieval and is the term-frequency component of the hybrid search in the platform. The recent survey by Gao et al. (2024) found that hybrid retrieval, combining lexical scoring

with dense vector similarity, consistently outperforms single-method approaches across benchmarks, a result observed directly in the present evaluation. Wang et al. (2024) further showed that LLM-derived embeddings can improve retrieval quality, although for the current scale the lighter MiniLM family proved sufficient. The Weaviate documentation (Weaviate B.V., 2024) describes the native hybrid-search facility relied on for the production index.

2.4 AI-Assisted Programming Education

Two studies were particularly influential here. Kazemitabaar et al. (2023) found that students who used AI code generators without structured feedback showed weaker conceptual understanding than those working through guided hints, even when their generated code passed the test cases. Prather et al. (2024) observed that novice programmers found Copilot useful but struggled to evaluate whether the generated code was correct or optimal. Both findings argue against simply giving a learner the answer. They directly motivate the code evaluator in DSA Tutor, which classifies a submission as optimal, brute-force, or incorrect with explicit time-complexity reasoning rather than producing the optimal solution outright.

3. Gaps Identified

3.1 Fragmentation of the Learning Workflow

The most visible gap in current DSA preparation is workflow fragmentation. A learner moves between video platforms for theory, judges such as LeetCode for practice, separate quiz websites for assessment, and PDF readers for reference material. None of these tools share session context, none track longitudinal progress in a unified way, and switching between them imposes real cognitive overhead. Educational research has long suggested that retention is stronger when learning, practice, and assessment are tightly coupled, yet no widely-used platform actually couples them.

3.2 Absence of Deterministic Code Evaluation

General-purpose chatbots will give an opinion on a piece of code if asked, but the response is unstructured and frequently inconsistent across runs. They do not categorise a submission as brute-force versus optimal in a way that the learner can rely on, and they do not commit to a specific time complexity. Online judges, by contrast, only check whether the code passes the test cases. They cannot tell a learner that an O(n squared) bubble sort is technically correct but unsuitable for an interview. Neither type of tool delivers what a real human interviewer would, a structured verdict on correctness, approach, and complexity together.

3.3 No Integrated Document Intelligence

Standalone RAG products such as ChatPDF let a user query an uploaded document but operate in isolation from any learning context. A student who has just finished a topic in DSA Tutor cannot, in a typical workflow, immediately query their own lecture notes for clarification without leaving the application and uploading the file elsewhere. The lack of an in-platform document intelligence layer breaks the otherwise continuous learning experience.

3.4 Single Points of Failure in API-Driven Tools

Tools that depend on a single LLM provider become unusable when that provider is rate-limited or temporarily down. Most existing educational chatbots have no fallback behaviour at all, which is acceptable for a hobby project but a serious limitation for anything used in real preparation. The three-tier fallback architecture in DSA Tutor (Gemini to Ollama Cloud to local Llama 3) is a direct response to this gap and is, to the authors knowledge, uncommon among student-facing AI learning tools.

3.5 Weak Persistence and Progress Tracking

Most chat-based AI tutors are stateless from a pedagogical point of view. They do not remember which topics a student has mastered, which problems were solved correctly, or which MCQs were missed. Without persistence, the system cannot adapt over time and the learner has no objective record of progress. The DSA Tutor backend therefore stores users, chats, messages, and test scores in SQLite using a normalised schema, so that progress is genuinely longitudinal rather than session-bound.

4. Problem Statement

Computer science undergraduates and early-career engineers preparing for technical interviews face a learning environment that is rich in content but poor in coherence. Conceptual material lives on one set of platforms, coding practice on another, self-assessment on a third, and reference reading on a fourth. None of these systems share state, and the available AI assistants, while capable, lack the structured pedagogical framework needed to behave as a tutor rather than as a general question-answering tool.

Concretely, the problem this work addresses is: how can a single AI-powered platform be designed and built so that it (i) provides structured, multi-mode tutoring for DSA, (ii) evaluates submitted code deterministically along correctness, approach, and complexity dimensions, (iii) administers and persistently records knowledge assessments, (iv) answers questions grounded in user-supplied documents with traceable citations, and (v) maintains continuous availability through a multi-tier LLM fallback that does not depend on any single external service?

5. Related Work

Several existing platforms occupy adjacent regions of the design space. The most relevant ones are summarised below to clarify what is genuinely new about DSA Tutor.

LeetCode and HackerRank dominate the coding-practice market, providing extensive problem banks and automated test-case judging. Their strength is breadth and rigour on the execution side, but they offer no conceptual tutoring, no MCQ-style assessment, no document analysis, and limited adaptation to the individual learner. GeeksforGeeks supplies a large body of static articles and code samples but is essentially a content site. It cannot evaluate work or hold a conversation.

On the conversational side, ChatGPT, Claude, and Google Gemini are powerful general-purpose assistants. They can explain DSA concepts and write code on demand. However, they do not classify code submissions deterministically, do not generate or score MCQs, do not maintain a persistent record of which topics a learner has practised, and do not provide document-grounded answers with citations from a user-supplied corpus. Khan Academy's Khanmigo is closer in spirit, an AI-driven tutor, but it targets school-age learners and broad subject coverage rather than DSA preparation.

RAG-only products such as ChatPDF and various open-source pipelines (LlamaIndex, Haystack) provide retrieval and citation but exist outside any learning workflow. The closest precedent for the integrated approach taken here is, to the authors knowledge, absent from the public literature. No widely-used platform combines structured ITS modes with a document-grounded RAG pipeline under a shared session context and a multi-tier LLM fallback.

6. System Architecture and Design

6.1 Overview

DSA Tutor is composed of two independently developed subsystems, the DSA Tutor (ITS) and Project

SiSo / RAGenius (RAG), bound together by a master-orchestrator pattern. The DSA Tutor backend is the root process. On startup, it spawns the RAGenius services, and on shutdown it tears them down cleanly. Each subsystem can be developed, tested, and reasoned about in isolation, but the user sees a single platform.

Concretely, four processes are involved at runtime. The DSA Tutor frontend (React 19 with Vite 7) runs on port 5174 and the matching FastAPI backend on port 8001. The RAGenius frontend runs on port 5173 and its FastAPI backend on port 8000, with a small bootloader on port 9999 that owns the Docker Compose lifecycle for the underlying RAG infrastructure (Weaviate vector store, Redis queue, API container, RQ worker container).

6.2 Master-Orchestrator and Process Lifecycle

When the DSA Tutor backend boots, the FastAPI lifespan handler probes whether port 9999 is in use. If not, it launches the RAGenius bootloader (run_app.py) as a subprocess, which in turn runs docker compose up -d for the RAG infrastructure. Once Docker reports all services healthy, the RAGenius frontend on port 5173 is started. On shutdown, the handler sends SIGTERM to every spawned subprocess and issues docker compose down on the RAG side. This pattern allows a single entry point (starting the DSA Tutor backend) to bring the entire platform up, and a single exit point to bring it down cleanly.

6.3 Three-Tier LLM Fallback

All language-model traffic flows through a single GeminiClient class that implements three tiers. Tier 1 calls the Google Gemini API (currently gemini-2.0-flash) and is used under normal conditions. Tier 2 engages when Tier 1 returns a 429 rate-limit error or a non-recoverable 5xx. The client then switches to Ollama's hosted cloud endpoint using the gpt-oss model for the remainder of the session. Tier 3 activates when the machine has no outbound internet connection at all, at which point the client routes to a local Ollama instance running Llama 3. The switching decision is session-scoped rather than per-request, so a single failed call does not trigger a retry loop. The user sees a single smooth conversation.

6.4 RAG Pipeline (RAGenius)

RAGenius implements a conventional but carefully assembled RAG pipeline. Uploaded DOCX files are parsed with python-docx. If a page returns no text (for example, a scanned image), the page is rasterised and passed through Tesseract via pytesseract, after which the recovered text rejoins the main flow. The parsed text is split using a recursive character text splitter into 500-character chunks with a 100-character overlap. The overlap is deliberate, preserving context across chunk boundaries.

Each chunk is embedded with the all-MiniLM-L6-v2 sentence-transformer (384-dimensional output) and indexed in Weaviate together with its document id, page number, and chunk index. These three fields form the citation that is later surfaced to the user in the shape [doc_id : page : chunk].

Retrieval is hybrid. A query is fed to BM25 over the indexed chunks and, in parallel, embedded and used for cosine-similarity search against the dense vectors. The two ranked lists are merged using a weighted score fusion (Section 7.4), and the top k = 5 chunks are passed into the LLM prompt as grounding context. The LLM is instructed to answer strictly from the retrieved chunks and to emit the accompanying citation after every factual claim. Processing is asynchronous. Uploads and embedding runs are enqueued in a Redis queue consumed by an RQ worker container, which allows the user-facing API to return immediately.

6.5 Data Layer

Three stores back the platform. SQLite, accessed through SQLAlchemy, holds users, chats, messages, and test scores, all the relational data needed for persistence and progress tracking. Weaviate holds document

embeddings and chunk metadata. Redis holds only transient job state for the RQ queue. The split is deliberate. SQLite is cheap and portable for structured data, Weaviate is purpose-built for vector search, and Redis scales easily if queue load grows.

7. Methodology

7.1 Two-Stage Intent Detection

Every user message is first sent through an intent detector that decides which handler should serve the request. The detector is two-stage. Stage one is a fast keyword scan against a curated set of category-specific triggers, for example test, mcq, and score for the testing mode. If stage one matches with high confidence, the message is routed immediately. Stage two, invoked only on low-confidence cases, sends the message to the LLM with a constrained prompt that asks for one of four category labels, namely learning, coding, testing, or general. In the evaluation (Section 9), this two-stage design reached 97% accuracy on a hundred-message test set.

7.2 Code Evaluation

The code evaluator is the most distinctive piece of the ITS. When the learner submits code, the evaluator constructs a structured prompt asking the LLM to (i) determine whether the solution is correct on the stated problem, (ii) classify the approach as brute_force or optimal, (iii) state the time complexity in big-O notation, and (iv) produce an interview-style feedback paragraph. The response is parsed into a JSON object with explicit fields, so the UI can render the verdict, complexity tag, and feedback in separate components. Crucially, the evaluator does not produce the correct code unless the learner explicitly asks for it. Instead, it provides progressive hints, a deliberate design choice informed by the studies of Kazemitabaar et al. (2023) and Prather et al. (2024) cited earlier.

7.3 MCQ Generation and Scoring

MCQ tests are generated on demand using a centralised template that asks the LLM to produce exactly five questions, each with four options, a designated correct answer, and a short explanation. The frontend renders the questions, captures the learner's selected options, and submits them back to the scoring handler. The handler compares the submissions against the designated answers, writes the final score to the test_scores table in SQLite (keyed by user id and topic), and returns both the score and the per-question explanations. Because scores are persisted, a learner can retake the same topic later and the system can display the progression over time.

7.4 Hybrid Retrieval and Score Fusion

Hybrid retrieval combines BM25 with dense vector cosine similarity. For a query q and a set of indexed chunks {c_i}, the BM25 score is given by

$$\mathrm{BM25}(q, c) = \Sigma_t \, \mathrm{IDF}(t) \cdot (f(t, c) \cdot (k_1+1)) / (f(t, c) + k_1 \cdot (1 - b + b \cdot |c|/\mathrm{avg\_dl})) \quad (1)$$

where f(t, c) is the term frequency of t in chunk c, |c| is the chunk length, avg_dl is the average chunk length over the corpus, and k_1 and b are the standard BM25 parameters (k_1 = 1.5, b = 0.75). The dense component is a plain cosine similarity between the query and chunk embeddings,

$$\mathrm{sim}(q, c) = (e_q \cdot e_c) / (\|e_q\| \cdot \|e_c\|) \quad (2)$$

Both scores are min-max normalised to [0, 1] and combined as

$$S_{\mathrm{hybrid}}(q, c) = \alpha \cdot \mathrm{sim}(q, c) + (1 - \alpha) \cdot \mathrm{BM25\_norm}(q, c) \quad (3)$$

with $\alpha = 0.5$ in the current configuration. The top-k chunks (k = 5 in production) are then passed to the LLM. Empirically this hybrid score reaches 91% top-5 relevance on the evaluation corpus (Section 9), a 17 percentage-point improvement over a vector-only baseline measured on the same data.

7.5 Evaluation Metrics

Throughout the rest of the paper, standard classification metrics are reported. For a binary or per-class problem with true positives TP, true negatives TN, false positives FP, and false negatives FN,

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN) \quad (4)$$

$$\text{Precision} = TP / (TP + FP), \quad \text{Recall} = TP / (TP + FN) \quad (5)$$

$$F1 = 2 \cdot \text{Precision} \cdot \text{Recall} / (\text{Precision} + \text{Recall}) \quad (6)$$

Latency is reported as a per-request mean, $\text{Latency\_mean} = (1/N) \Sigma_i L_i$, along with the maximum observed value over the test set.

8. Testing and Quality Assurance

Testing was carried out at three levels, namely unit, integration, and system-level evaluation. The backend uses pytest as its test framework, with the Gemini client replaced by a mock that returns deterministic fixtures for test runs.

At the unit level, thirty test cases were written across the intent detector, code evaluator, test handler, GeminiClient, conversation memory, learning handler, database session, and a few utility modules. Cases were designed to cover both the happy path and recognisable edge cases, for example an empty code submission, an unknown topic label, or a malformed JSON response from the LLM.

At the integration level, eight cross-module workflows were exercised, for example a learning-to-coding transition with topic preservation, an MCQ session with score persistence, and a document upload followed by a retrieval-augmented answer with citation. These integration tests run against a SQLite test database and a short-lived Weaviate instance spun up via Docker Compose.

System-level evaluation is reported quantitatively in the next section.

9. Results and Discussion

9.1 Accuracy Across Three Dimensions

Table 1 summarises the platform's accuracy across the three primary evaluation dimensions. Intent detection was tested on a manually labelled set of 100 messages spanning all four handler categories. Code evaluation was tested on a curated benchmark of 50 submissions, with balanced optimal, brute-force, and incorrect examples across common problem patterns (two-pointer, binary search, recursion with memoisation, greedy). RAG retrieval was tested on a 50-query evaluation set run against a corpus of ingested DOCX study material.

Table 1. Accuracy of DSA Tutor Across Three Evaluation Dimensions

Dimension	Test Size	Accuracy	Baseline
Intent Detection	100 messages	**97%**	70–75%
Code Evaluation	50 submissions	**99%**	N/A
RAG Retrieval (top-5)	50 queries	**91%**	74% (vector-only)

Baselines: 70–75% reflects general-purpose conversational assistants on educational intent classification; 74% is vector-only RAG measured on the same corpus.

The intent-detector result of 97% reflects the keyword-first design. The three messages misclassified in the test set were edge cases whose surface form overlapped two categories (for example, a learning-style question that embedded a short code snippet). The code-evaluator result of 99% was strong because the benchmark deliberately clustered around archetypal patterns. Accuracy on more unusual algorithms would likely be lower. The RAG result of 91% at top-5 is the most decisive finding, because it represents a 17 percentage-point gain over the vector-only baseline on the same queries, directly confirming the benefit of the hybrid approach.

9.2 Latency Across LLM Tiers

Response time was measured across all three LLM tiers using the same set of representative prompts. Table 2 reports mean and maximum latency.

Table 2. Latency Across the Three LLM Tiers

LLM Tier	Mean (seconds)	Maximum (seconds)
Gemini (Tier 1)	**3.2**	6.1
Ollama Cloud (Tier 2)	5.4	9.2
Local Llama 3 (Tier 3)	7.5	12.4

Measured over 100 requests for Tier 1 and 50 each for Tiers 2 and 3.

Tier 1 latency of 3.2 seconds on average is comfortably within the threshold for an interactive learning tool, especially given that response streaming begins much earlier than the full-response timestamp (Section 9.3). Tier 2 and Tier 3 are slower, as expected, but remain usable and preserve availability at the cost of a few additional seconds per request.

9.3 Streaming and Perceived Latency

Even when total generation time was 3 to 7 seconds, the user typically saw the first token within 0.5 to 1 second because the FastAPI endpoints wrap the LLM in a StreamingResponse and the React frontend binds to an Axios AbortController for cancellable streaming. Perceived latency, the time until the user first sees output, is therefore substantially lower than the total generation time reported in Table 2.

9.4 Discussion of Failure Modes

Two failure modes are worth flagging. First, the code evaluator can occasionally over-credit a memoised recursive solution as optimal when a tabulated dynamic programming solution would be preferable on space grounds. The verdict is technically defensible but does not fully match interview expectations. Second, retrieval quality on the RAG side degrades noticeably on documents that are almost entirely embedded images (for example, slide decks converted to DOCX without selectable text), because the Tesseract OCR fallback introduces character-level noise that weakens both BM25 and the dense embedding. A more aggressive OCR-preprocessing step is planned for future work.

9.5 Discussion and Implications

The results support the central argument of this work. A structured, modular integration of an Intelligent Tutoring System with a Retrieval-Augmented Generation pipeline produces meaningfully higher performance on the three tasks that matter most to a learner, namely classifying what they are trying to do, evaluating what they have produced, and answering their questions from grounded material. The intent-detection result exceeds baseline conversational assistants because the keyword-first design constrains output to four concrete modes, avoiding the ambiguity that hurts general-purpose chatbots. The

code-evaluation result is made possible by a constrained JSON schema that forces the LLM to commit to a verdict rather than hedging. The RAG retrieval gain confirms that hybrid lexical-plus-dense search is not just theoretically attractive but measurably effective at the scale a student corpus typically inhabits.

For educators, these findings suggest that a single unified platform can replace the tool-switching workflow that currently dominates DSA preparation, and that doing so need not involve expensive infrastructure. For learners, they suggest that feedback can be both structured and conversational at the same time. For the broader AI-in-education community, the work offers a concrete template for combining pedagogical scaffolding with modern LLM and retrieval capabilities in a way that remains resilient to external service failures.

10. Comparative Analysis

Table 3 situates DSA Tutor in the existing landscape on a feature-by-feature basis. The comparison is not a benchmark of model quality but of capability coverage, namely what a user can actually do inside each tool.

Table 3. Capability Comparison Against Common DSA-Learning Tools

Capability	ChatGPT	LeetCode	GfG	DSA Tutor
Structured DSA tutoring	✗	✗	Partial	✓
Integrated code editor	✗	✓	✗	✓
Deterministic code eval	✗	Test-case only	✗	✓
MCQ tests with scoring	✗	✗	Static	✓
Document RAG with citations	✗	✗	✗	✓
Persistent learner state	✗	Partial	✗	✓
Multi-tier LLM fallback	—	—	—	✓
Offline operation possible	✗	✗	✗	✓ **(Tier 3)**

'GfG' = GeeksforGeeks. '—' indicates the row is not applicable (the tool does not depend on a single LLM).

The pattern is clear. Each existing tool is strong on one axis and absent on most others. ChatGPT has conversational range but no structured assessment, code editor, or document grounding. LeetCode has a code editor and test-case judging but no tutoring, MCQs, or document RAG. GeeksforGeeks is essentially content delivery. DSA Tutor is the only system in the comparison that covers the full interaction loop from learning through practice, assessment, and reference, and is also the only one with a fallback mechanism that lets it run without access to any single external LLM provider.

11. Deployment and Operations

11.1 Local Development

In its current form the platform runs on a single developer workstation. Dependencies are split between two repositories, dsa_tutor (FastAPI backend and React frontend) and project_siso (RAGenius backend, frontend, and Docker Compose stack). Python dependencies are pinned via requirements.txt, and Node dependencies via package-lock.json. A fresh laptop with Python 3.10+, Node 20+, and Docker Desktop installed can bring the whole platform up with a small number of documented shell commands.

11.2 Service Orchestration

The startup sequence is deterministic. Step 1: the developer launches the DSA Tutor frontend (npm run dev, port 5174). Step 2: the developer launches the DSA Tutor backend (uvicorn on port 8001), whose lifespan handler then performs steps 3 and 4 automatically. Step 3: the backend spawns run_app.py from project_siso as a subprocess, which brings up the Docker Compose stack on port 9999. Step 4: the RAGenius frontend on port 5173 is started. Shutdown reverses the sequence cleanly, sending SIGTERM to every spawned subprocess and finally issuing docker compose down on the RAG side.

11.3 Authentication and Security

Authentication uses bcrypt password hashing and JWT session tokens signed with HS256. The frontend Axios client attaches the JWT as a Bearer token via a request interceptor, and the FastAPI dependency layer rejects requests without a valid signature. CORS is restricted to the expected local development origins, and the Gemini API key is read from a local .env file and never committed to the repository.

11.4 Real-World Usage and Observations

The platform has been operated on a developer workstation throughout the project development cycle (September 2025 through April 2026), supporting concurrent interactive use during testing and demonstration. Two observations stand out. First, the three-tier LLM fallback proved useful in practice. Short Gemini quota exhaustions during heavy evaluation days were handled transparently, with the session switching to Ollama Cloud and continuing without user intervention. Second, the master-orchestrator pattern made day-to-day development considerably easier. Starting a single backend was enough to bring the entire platform, including the RAG infrastructure, up in a reproducible way, which removed a common source of setup errors.

The platform is built entirely on open-source software (FastAPI, React, SQLite, Weaviate, Redis, Ollama) and free-tier APIs, which keeps the economic barrier low. This design choice is deliberate. The goal is to produce a system that could be deployed by an individual learner on a modest laptop or by an educational institution with limited cloud budget, without compromising functionality.

11.5 Future Operational Hardening

For a production deployment, the SQLite layer would migrate to PostgreSQL, the worker container would scale horizontally behind RQ, the Weaviate volume would move to a managed cloud instance, and the three development servers would be consolidated behind an Nginx reverse proxy with TLS. None of these changes affect the application logic, only the operational surface.

12. Conclusion and Future Work

12.1 Conclusion

This work presents DSA Tutor, a hybrid AI learning platform that integrates an Intelligent Tutoring System with a Retrieval-Augmented Generation engine under a shared session context. The platform addresses five specific gaps in the current DSA preparation landscape, namely workflow fragmentation,

the absence of deterministic code evaluation, the lack of in-platform document intelligence, single points of failure in API-driven tools, and weak persistence of learner progress. It does so through four architectural choices: a handler-based modular design that keeps pedagogical modes separate, a three-tier LLM fallback that preserves availability across rate-limits and network failures, a hybrid BM25-plus-dense RAG pipeline with traceable citations, and a persistent data layer that records users, chats, messages, and test scores across sessions.

Across the three primary evaluation dimensions, DSA Tutor reached 97% intent-detection accuracy, 99% code-evaluation accuracy, and 91% top-5 RAG retrieval relevance, the last of which represents a 17 percentage-point improvement over a vector-only baseline measured on the same corpus. Latency is held to 3.2 seconds on the Tier 1 path and remains usable on the offline Tier 3 fallback. The comparative feature matrix shows that, among commonly-used DSA-learning tools, DSA Tutor is the only system that covers the full learning loop from explanation through practice, assessment, and document-grounded reference.

The platform also demonstrates two design patterns believed to be broadly applicable. First, the master-orchestrator pattern allows two independently-developed subsystems to present as a single product without surrendering modularity. Second, session-scoped multi-tier LLM switching turns a potential failure mode into a non-event for the end user. Both patterns are readily transferable to other educational-technology systems that need to combine structured pedagogy with generative AI while maintaining operational resilience.

12.2 Future Work

Several directions are planned. Expanded document support in RAGenius (PDF and PPTX) would let learners ingest lecture slides and research papers directly, without prior format conversion. A spaced-repetition scheduler for MCQ topics would let the system adapt over time to each learner's forgetting curve. A voice interface would open the platform to mobile use during commutes. A richer evaluation programme, including a longitudinal study that compares DSA Tutor learners to a control group using fragmented tools, would quantify the pedagogical benefit more rigorously than the present capability-level comparison. Finally, extension into adjacent domains such as machine learning coursework or systems programming, where similar tool fragmentation exists, would test how far the handler-based ITS plus hybrid RAG template generalises beyond DSA.

Acknowledgments

I express my sincere gratitude to Dr. Kalaavathi B, Professor Grade 1, Department of IoT, School of Computer Science and Engineering, Vellore Institute of Technology, for her continuous guidance, constructive feedback, and encouragement throughout the design, implementation, and evaluation of this project. I also thank the School of Computer Science and Engineering at VIT Vellore for providing the academic environment in which this work was carried out.

References

1. Anderson, J. R., Corbett, A. T., Koedinger, K. R. and Pelletier, R. (1995), "Cognitive tutors: Lessons learned", Journal of the Learning Sciences, Vol. 4 No. 2, pp. 167-207.
2. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G. and Askell, A. (2020), "Language models are few-shot learners", in Advances in Neural Information Processing Systems 33 (NeurIPS), pp. 1877-1901.

3. FastAPI (2024), "FastAPI — Modern, fast web framework for building APIs with Python", available at: https://fastapi.tiangolo.com/ (accessed 10 April 2026).
4. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J. and Wang, H. (2024), "Retrieval-augmented generation for large language models: A survey", IEEE Transactions on Knowledge and Data Engineering, Vol. 36 No. 5, pp. 2123-2145.
5. Google DeepMind (2024), "Gemini: A family of highly capable multimodal models", arXiv preprint, arXiv:2312.11805.
6. Hugging Face (2024), "Sentence-Transformers: all-MiniLM-L6-v2 model card", available at: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2 (accessed 10 April 2026).
7. Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., Stadler, M., Weller, J., Kuhn, J. and Kasneci, G. (2023), "ChatGPT for good? On opportunities and challenges of large language models for education", Learning and Individual Differences, Vol. 103, article 102274.
8. Kazemitabaar, M., Chow, J., Ma, C. K. T., Ericson, B. J., Weintrop, D. and Grossman, T. (2023), "Studying the effect of AI code generators on supporting novice learners in introductory programming", in Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1-23.
9. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S. and Kiela, D. (2020), "Retrieval-augmented generation for knowledge-intensive NLP tasks", in Advances in Neural Information Processing Systems 33 (NeurIPS), pp. 9459-9474.
10. Microsoft (2024), "Monaco Editor — The code editor that powers VS Code", available at: https://microsoft.github.io/monaco-editor/ (accessed 10 April 2026).
11. Ollama (2024), "Ollama — Run large language models locally", available at: https://ollama.ai/ (accessed 10 April 2026).
12. Prather, J., Reeves, B. N., Denny, P., Becker, B. A., Leinonen, J., Luxton-Reilly, A., Powell, G., Finnie-Ansley, J. and Santos, E. A. (2024), "'It's weird that it knows what I want': Usability and interactions with Copilot for novice programmers", ACM Transactions on Computer-Human Interaction, Vol. 31 No. 1, pp. 1-31.
13. Reimers, N. and Gurevych, I. (2019), "Sentence-BERT: Sentence embeddings using Siamese BERT-networks", in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP), pp. 3982-3992.
14. Robertson, S. E. and Zaragoza, H. (2009), "The probabilistic relevance framework: BM25 and beyond", Foundations and Trends in Information Retrieval, Vol. 3 No. 4, pp. 333-389.
15. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P. and Bhosale, S. (2023), "Llama 2: Open foundation and fine-tuned chat models", arXiv preprint, arXiv:2307.09288.
16. VanLehn, K. (2011), "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems", Educational Psychologist, Vol. 46 No. 4, pp. 197-221.
17. Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R. and Wei, F. (2024), "Improving text embeddings with large language models", in Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL), pp. 11897-11916.

18. Weaviate B.V. (2024), "Weaviate vector database documentation", available at: https://weaviate.io/developers/weaviate (accessed 10 April 2026).
19. Yan, L., Sha, L., Zhao, L., Li, Y., Martinez-Maldonado, R., Chen, G., Li, X., Jin, Y. and Gašević, D. (2024), "Practical and ethical challenges of large language models in education: A systematic scoping review", British Journal of Educational Technology, Vol. 55 No. 1, pp. 90-112.
20. Zamfirescu-Pereira, J. D., Wong, R. Y., Hartmann, B. and Yang, Q. (2023), "Why Johnny can't prompt: How non-AI experts try (and fail) to design LLM prompts", in Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1-21.