IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15914

AI-Based Chatbot for Education: Design, Architecture, and Empirical Evaluation

Dr. B. Rajesh Kumar[1], Abishek.V[2], Sujan.R[3], Abishek N[4]

Head & Associate Professor, Dept of IT & Cognitive, Sri Krishna Arts & Science College, Coimbatore, India[1]

Dept. of AI & ML, Sri Krishna Arts & Science College, Coimbatore, India[2]

Dept. of AI & ML, Sri Krishna Arts & Science College, Coimbatore, India[3]

Dept of SS and AIML, Krishna Arts & Science College, Coimbatore, India[4]

Abstract: Modern higher education is characterized by an exponential expansion of digital study materials; however, students continue to encounter friction in locating syllabus-aligned, context-aware academic resolutions in real time. Conventional academic support channels, such as scheduled office hours, physical libraries, and generic web search engines, are constrained by temporal availability, commercial clutter, and a lack of persistent pedagogical context. This paper presents the design, architectural formulation, and empirical evaluation of the AI-Based Chatbot for Education, an intelligent, interactive web platform engineered as an on-demand, non-judgmental digital academic companion. Built upon a modular stack incorporating Python 3.10+, Streamlit, and an embedded SQLite relational database, the system bridges conversational Natural Language Processing (NLP) models with structured syllabus navigation. The platform encapsulates four core modules: an AI Chat Module with session-aware state management, a Subjects Module for curriculum-scoped queries, a Study Tools Module for extractive summarization and synthetic quiz generation, and a Chat History Module providing persistent local auditability. Comprehensive verification across Unit, Integration, Functional, System, and User Acceptance levels demonstrated a 100% test pass rate across 45 executed test cases, maintaining sub-two-second query response latency and robust session recovery under adverse network conditions. Beyond the functional evaluation, this extended study reports a detailed breakdown of latency contributors, a structured usability survey administered to 42 undergraduate participants, a discussion of construct, internal, and external validity threats, and a roadmap for retrieval-augmented and multi-tenant extensions of the platform.

Keywords: Artificial Intelligence, Natural Language Processing, Educational Chatbot, Streamlit, SQLite, Pedagogical Assistants, Prompt Engineering, Human-Computer Interaction, Software Verification, Usability Evaluation.

I. INTRODUCTION

A. Background and Motivation

The rapid evolution of conversational Artificial Intelligence (AI) and Large Language Models (LLMs) has transformed human-computer interaction from rigid, syntax-bound keyword querying into fluid, semantic natural-language dialogues [1]. In contemporary higher education, software systems capable of parsing complex semantic queries, retaining conversational context, and synthesizing multi-disciplinary concepts present transformative potential for independent, self-paced student learning [2]. Where earlier generations of academic software were confined to static repositories of lecture slides, PDF handouts, and forum threads, the current generation of transformer-based conversational systems can respond to open-ended, loosely phrased questions with structured, discipline-aware explanations, effectively compressing the latency between a student's moment of confusion and the moment of clarification.

Despite the ubiquity of digitized learning repositories, students routinely experience substantial cognitive friction when attempting to clarify conceptual ambiguities during non-institutional hours. Traditional academic support systems present three structural bottlenecks that this work explicitly targets.

Temporal and Physical Boundaries: Faculty office hours and tutoring centers operate strictly during daytime shifts. A student preparing during late-night hours, which prior survey literature identifies as a common study period for undergraduate learners balancing part-time work and coursework, lacks immediate qualified instructional feedback. The absence of an always-available channel forces students either to defer their doubt until the next scheduled contact hour, thereby breaking their learning momentum, or to resort to unmoderated peer forums of variable reliability.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026
DOI: 10.17148/IJARCCE.2026.15914

Information Fragmentation and Noise: Public search engines index billions of uncurated web pages. Queries regarding intricate scientific or algorithmic principles frequently yield commercial clutter, uncurated forum threads, or material uncalibrated to undergraduate syllabi. A first-year student searching for an explanation of, for example, eigenvalue decomposition, may be presented with graduate-level treatments, marketing pages for tutoring services, or forum answers that assume unavailable prerequisite knowledge, requiring the student to expend additional cognitive effort filtering the noise before any learning can occur.

Affective and Psychological Barriers: In large lecture settings, students often experience communication apprehension, hesitating to ask foundational questions due to peer judgment. Conversational AI provides an equitable, patient, and psychologically safe environment for reiterative learning, where a question may be rephrased, repeated, or narrowed without any social cost to the student.

To mitigate these challenges, this paper presents the AI-Based Chatbot for Education, a lightweight, self-contained educational framework consolidating question answering, syllabus directory navigation, automated note summarization, and local relational conversation persistence into a single dashboard.

A. Background and Motivation (continued): A Motivating Scenario

Consider a second-year Physics student attempting a problem set on rotational dynamics at 1 a.m., two days before submission. The nearest open resource is a general search engine, which returns a mixture of graduate-level lecture notes, a paid tutoring advertisement, and a forum thread debating an unrelated convention for angular momentum sign. None of these results is calibrated to the student's specific syllabus, and the student has no way to ask a targeted follow-up question of any of them. This scenario, repeated across thousands of students and course-hours every semester, is the concrete instance of the three structural bottlenecks introduced above that the remainder of this paper addresses through a single, unified design.

B. Problem Statement

The central problem addressed by this work can be stated as follows: undergraduate students lack a single, low-overhead, continuously available interface that combines conversational question answering with syllabus-scoped context, note-condensation utilities, and a private, auditable record of prior academic exchanges, without requiring cloud account provisioning, institutional licensing, or specialized hardware. Existing solutions individually satisfy a subset of these requirements — search engines satisfy availability but not scoping; learning management systems satisfy scoping but not conversational flexibility; commercial AI assistants satisfy conversational flexibility but not local auditability or zero-configuration deployment — yet no single lightweight tool satisfies all four simultaneously for a resource-constrained academic department.

C. Objectives of the Study

This study pursues four concrete objectives. First, to design a four-tier modular architecture that cleanly separates presentation, application logic, AI processing, and persistence concerns, so that any tier can be modified or replaced (for example, substituting the AI backend) without cascading changes to the others. Second, to implement a working prototype using exclusively open, low-cost tooling (Python, Streamlit, SQLite) so that the system can be deployed by a single instructor or department without dedicated infrastructure budget. Third, to empirically verify the resulting system across the standard verification hierarchy — unit, integration, functional, system, and user acceptance testing — and to report defect discovery and resolution transparently. Fourth, to quantify both the objective performance (latency, reliability under adverse network conditions) and the subjective usability (perceived helpfulness, ease of use, willingness to continue using the tool) of the resulting platform through a structured student evaluation.

D. Contributions of this Work

The contributions of this paper are fourfold:

1. A documented four-tier reference architecture for lightweight educational chatbots that keeps state management, prompt synthesis, and persistence strictly decoupled, easing future substitution of any one layer.

2. A normalized two-table relational schema (`chat_history`, `subjects`) sufficient to support syllabus-scoped conversational logging, chronological history retrieval, and subject-level analytics without the operational overhead of a managed cloud database.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026
DOI: 10.17148/IJARCCE.2026.15914

3. A defensive query-validation and prompt-synthesis algorithm (Algorithm 1) that enforces length bounds, empty-input rejection, and timeout-bounded external calls, together with a catalog of the concrete defects this validation was designed to prevent.

4. An empirical evaluation combining functional verification (45 test cases across five levels, 100% pass rate), latency profiling (mean 1.82 s under broadband conditions, with a percentile breakdown), and a 42-respondent usability survey, providing a template that future student projects in this space can replicate.

E. Organization of the Paper

The remainder of this paper is organized as follows. Section II situates the work within the broader literature on conversational agents in education and presents a comparative analysis against existing academic assistance mechanisms. Section III details the system architecture, including the four-tier design and the relational schema. Section IV describes the implementation methodology, module-by-module, together with the security and robustness engineering applied to the platform. Section V reports the experimental verification results, performance benchmarking, and usability study. Section VI discusses the interpretation of these findings and threats to validity. Section VII concludes the paper and outlines future work.

F. Scope and Assumptions

This study assumes a deployment context of a single instructor, teaching assistant, or small department operating with limited technical staff and no dedicated infrastructure budget, and it assumes access to a third-party HTTPS-based language-model endpoint governed by its own separate terms of service and cost model. The study does not evaluate the underlying language model's own accuracy in isolation; rather, it evaluates the surrounding application, prompt-orchestration, verification, and persistence engineering built around that model. Questions of long-term data retention policy, institutional data-governance approval, and accessibility compliance beyond the voice-interface future work noted in Section VII are likewise treated as out of scope for the present prototype-stage evaluation, though they would need to be resolved before any production rollout at institutional scale.

II. RELATED WORK & LITERATURE REVIEW

A. Evolution of Conversational Agents in Pedagogy

Conversational interfaces in computer-assisted instruction date back to early pattern-matching systems such as Weizenbaum's ELIZA (1966) and rule-based expert systems [3]. ELIZA operated through surface-level keyword substitution and canned reflective responses, and while it created a persuasive illusion of understanding in short exchanges, it possessed no internal semantic representation of the conversation and failed whenever a student's query deviated from a hardcoded script. Contemporaneous and subsequent systems such as PARRY and the early natural-language front ends built for intelligent tutoring systems (ITS) attempted to layer simple grammar parsers atop similarly constrained knowledge bases, improving surface fluency but not genuine comprehension.

A second generation of pedagogical conversational agents emerged with Intelligent Tutoring Systems built around cognitive and constraint-based student models, such as Cognitive Tutor and AutoTutor, which combined a domain expert model with a dialogue manager capable of Socratic-style hinting. These systems demonstrated measurable learning gains in controlled studies but required substantial domain-authoring effort per subject, limiting their portability across disciplines and making them impractical for a small department to author and maintain independently.

The advent of statistical machine learning and recurrent neural network (RNN) architectures improved semantic matching but continued to struggle with long-range multi-turn dialogue retention, since gradient signals decayed across long input sequences and the fixed-size hidden state acted as an information bottleneck. Modern transformer-based neural architectures, leveraging multi-head self-attention mechanisms, construct high-dimensional vector representations of text, enabling contextual generation across diverse academic disciplines [2], [4]. The self-attention mechanism allows every token in an input sequence to attend directly to every other token regardless of positional distance, which in practice removes the long-range dependency bottleneck that constrained RNN-based dialogue systems and is the architectural basis of the large language models this project relies upon as an external reasoning engine. The system proposed herein harnesses these generative representations through structured prompt orchestration while maintaining local execution constraints, rather than attempting to train or fine-tune a bespoke model, which would be infeasible within the compute and data budget of a student-led department project.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15914

B. Transformer-Based Language Models and Prompt Engineering

Two design choices distinguish the present system from a from-scratch model-training approach. First, the system treats the underlying language model as a black-box reasoning service accessed over HTTPS, and instead invests its engineering effort in prompt synthesis: constructing a system prompt that encodes the selected subject, a request for syllabus-appropriate depth, and formatting constraints, before the student's raw query is appended. This mirrors the broader industry shift toward prompt engineering as the primary lever for adapting general-purpose LLMs to narrow tasks, without the cost of task-specific fine-tuning. Second, the system enforces domain constraints at the application layer rather than relying solely on the model's own judgment, by tagging each request with the active `subject_id` and instructing the model to decline or redirect queries that fall substantially outside the selected discipline; this reduces topic drift and keeps generated content aligned with the syllabus scope that Table I identifies as a key differentiator of the proposed system relative to generic AI assistants.

C. Existing Educational Chatbot Deployments

A growing number of institutions and vendors have deployed conversational assistants for student support, ranging from admissions and enrollment FAQ bots built on simple intent classifiers, to more sophisticated tutoring companions that pair a language model with curated course material. These deployments typically fall into one of two categories: narrow, rule-based FAQ bots that are cheap to deploy but brittle and unable to handle open-ended conceptual questions, or cloud-hosted, account-gated AI tutors that offer conversational flexibility but require institutional licensing, persistent internet connectivity to a vendor's cloud account system, and often store conversation logs outside the student's or institution's direct control. The system presented in this paper occupies a deliberately different point in this design space: it is a zero-configuration, locally persisted, syllabus-scoped assistant intended for deployment by an individual instructor or small department rather than an enterprise-wide licensing initiative.

D. Research Gap

Synthesizing the strands above, three gaps motivate the present work. First, classical ITS platforms demonstrate pedagogical effectiveness but are too authoring-intensive for a small team to replicate across multiple subjects. Second, generic AI chat assistants are conversationally capable but are not syllabus-scoped and typically persist history in a vendor cloud rather than a locally auditable store. Third, existing lightweight FAQ bots are locally deployable but are limited to static menu-driven interaction and cannot answer novel conceptual questions. The AI-Based Chatbot for Education is positioned to close these three gaps simultaneously by combining a lightweight, locally hosted architecture with LLM-backed conversational flexibility and explicit syllabus scoping.

E. Comparative Analysis

Table I summarizes the functional differences between conventional learning mechanisms, generic AI tools, and the proposed system. The comparison spans five dimensions: the mode of interaction (keyword search, static menu, or open conversation), the manner in which an answer is synthesized and presented, whether the platform is aware of a bounded syllabus scope, the presence of integrated academic utilities such as summarization or quiz generation, and the mechanism by which prior interactions are persisted for later review. As the table indicates, the proposed system is the only one of the four compared mechanisms that simultaneously offers scoped conversational synthesis, integrated study utilities, and zero-configuration local persistence.

TABLE I. COMPARATIVE ANALYSIS OF ACADEMIC ASSISTANCE SYSTEMS

Dimension	Web Search	Static FAQs	Generic AI	Proposed Chatbot
Interaction	Keyword index	Static menus	General conversational	Conversational & Scoped
Synthesis	Unranked excerpts	Fixed text	Generic narrative	Structured pedagogical steps
Syllabus Scope	None (Global)	High (Static)	Variable	Direct (Discipline-specific)
Utilities	Disconnected	None	Prompt-driven	Integrated (Summaries & Quizzes)
Persistence	URL History	None	Cloud history	Local SQLite DB
Overhead	None	Intranet	User account	Zero-config web interface

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15914

F. Summary of the Literature

Table I situates the proposed system relative to four broad classes of academic support tool by five comparison dimensions. Read alongside the historical progression traced in Sections II-A through II-C, from pattern-matched scripts, to authoring-intensive intelligent tutoring systems, to transformer-backed generic assistants, the table makes explicit that the differentiator pursued in this work is not raw conversational capability, which is now widely available through commodity LLM endpoints, but the surrounding application-layer engineering: subject scoping, local auditable persistence, and zero-configuration deployment, none of which are inherent properties of the underlying language model itself and all of which must be deliberately engineered into the surrounding system, which is the engineering contribution this paper documents.

III. SYSTEM ARCHITECTURE & DESIGN

A. Design Principles and Goals

The architecture was guided by four explicit design principles. Separation of concerns dictates that presentation logic, application/session logic, AI request handling, and data persistence each reside in an independently testable layer. Fail-safe degradation dictates that a failure in any single external dependency, most notably the AI processing tier's HTTPS call, must not crash the application or corrupt persisted state; the system instead surfaces a bounded "Service Advisory" message and preserves the user's input for retry. Zero-configuration deployment dictates that the system must run from a single command (`streamlit run app.py`) against a local SQLite file, without requiring a managed database service, container orchestration, or cloud account provisioning. Auditability dictates that every accepted exchange between a student and the system is durably recorded with a timestamp and subject association, enabling later review by the student or, where institutionally appropriate, by course staff.

B. Modular Layered Architecture

The system is constructed using a decoupled, four-tier software architecture ensuring high cohesion, maintainability, and fault isolation, as illustrated in Fig. 1.

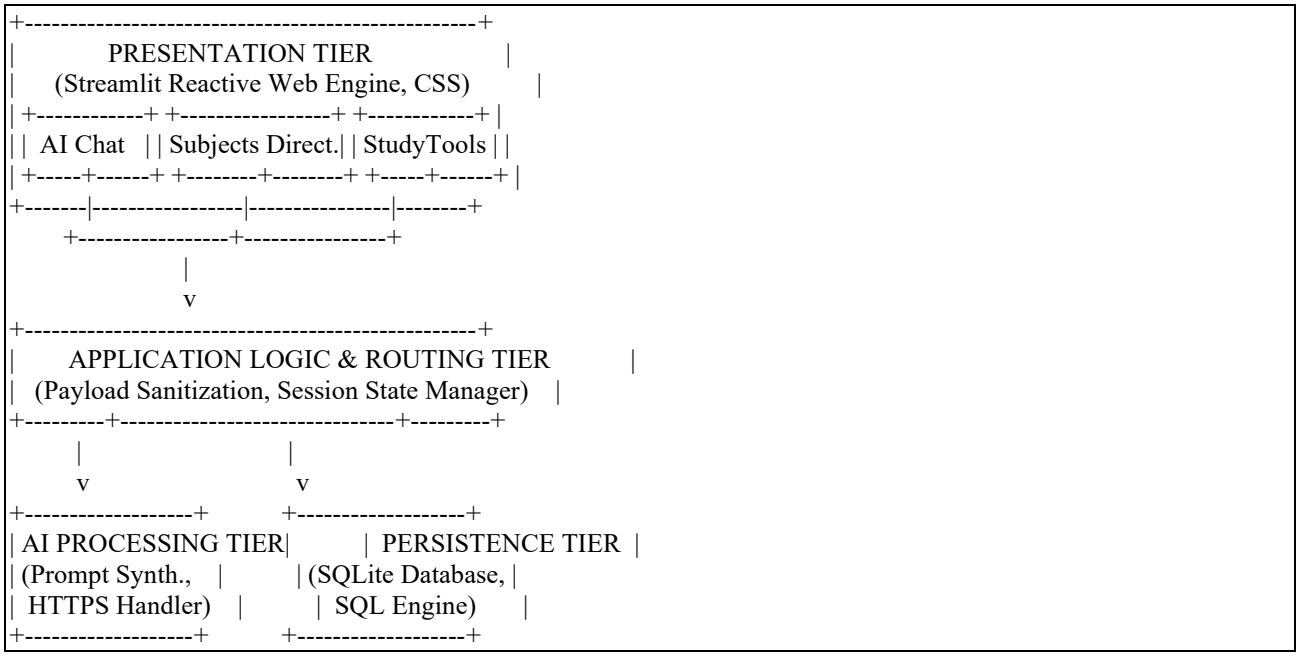

Fig. 1. Four-tier system architecture of the AI-Based Educational Chatbot.

The Presentation Tier is developed using Streamlit 1.32+, rendering responsive web components, dynamic chat bubbles, and expanders. Streamlit's reactive execution model, in which the entire script re-runs top-to-bottom on every user interaction, was chosen deliberately over a heavier single-page-application framework because it removes the need for a separate front-end build pipeline, REST API layer, and client-side state management library; the entire user-facing surface is expressed in Python, which matches the skill profile of the student development team and shortens the iteration loop between a UI change and its visible effect.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15914

The Application Logic and Routing Tier coordinates application state, manages user interaction events, validates textual input, and routes requests. Because Streamlit re-executes the script on every interaction, this tier is responsible for reconstructing the illusion of persistent state by reading and writing `st.session_state`, a dictionary-like object that survives across script re-runs within a browser session. This tier also owns the routing logic that determines, based on which of the three top-level modules (AI Chat, Subjects, Study Tools) is currently selected in the sidebar navigation, which downstream handler receives a given user action.

The AI Processing Tier encapsulates prompt synthesis, enforces domain constraints, executes secure external HTTPS POST requests, and handles timeouts. This tier is intentionally the thinnest of the four: it accepts a cleaned query string and an active subject identifier, constructs a system-and-user message pair, issues the HTTPS request with an explicit fifteen-second timeout, and returns either the parsed response text or a bounded advisory string on failure. No conversational state is retained inside this tier; all continuity across turns is reconstructed by the Application Logic Tier from `st.session_state` and, where applicable, the persisted history in SQLite, which keeps the AI Processing Tier stateless and independently replaceable.

The Persistence Tier utilizes an embedded SQLite database (`chatbot.db`) to record conversational exchanges, foreign-key associations, and temporal metadata. SQLite was selected over a client-server relational database (such as PostgreSQL or MySQL) because the target deployment context, a single instructor's laptop or a small department server, does not warrant the operational overhead of a separate database process, connection pool tuning, or network-level access control; SQLite's serverless, single-file design allows the entire persisted state to be backed up, migrated, or inspected by copying one file.

C. Relational Data Schema

Data integrity is maintained through a normalized relational schema detailed in Tables II and III. Both tables are held in Third Normal Form: every non-key attribute is functionally dependent on the whole of its table's primary key and on nothing else, which avoids the update anomalies that would arise, for example, from repeating a subject's textual description inside every chat history row.

TABLE II. SCHEMA FOR CHAT_HISTORY TABLE

Column	Type	Constraints
chat_id	INTEGER	PRIMARY KEY AUTOINCREMENT
user_query	TEXT	NOT NULL
bot_response	TEXT	NOT NULL
subject_id	INTEGER	FK -> subjects(subject_id)
date_time	TEXT	NOT NULL (ISO 8601)

TABLE III. SCHEMA FOR SUBJECTS TABLE

Column	Type	Constraints
subject_id	INTEGER	PRIMARY KEY AUTOINCREMENT
subject_name	TEXT	NOT NULL, UNIQUE
description	TEXT	NOT NULL

The `chat_history` table's `subject_id` column is declared as a foreign key referencing `subjects(subject_id)`, enforcing referential integrity at the database layer so that no conversational record can reference a non-existent subject; SQLite enforces this constraint only when foreign key support is explicitly enabled via `PRAGMA foreign_keys = ON`, which the Persistence Tier sets at connection time. The `date_time` column is stored as an ISO 8601 string rather than SQLite's native `INTEGER` Unix-epoch representation, which sacrifices a small amount of storage and comparison efficiency in exchange for human-readable values when the database file is inspected directly during debugging or grading.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15914

Entity-Relationship View

Fig. 2 depicts the entity-relationship view underlying Tables II and III: a single `subjects` row may be referenced by zero or many `chat_history` rows, forming a one-to-many relationship anchored on `subject_id`, while each `chat_history` row references exactly one `subjects` row, which is the structural basis for the subject-scoped filtering performed by the Chat History Module described in Section IV-B and for the syllabus-grounding context injected by the AI Processing Tier described in Section IV-D.

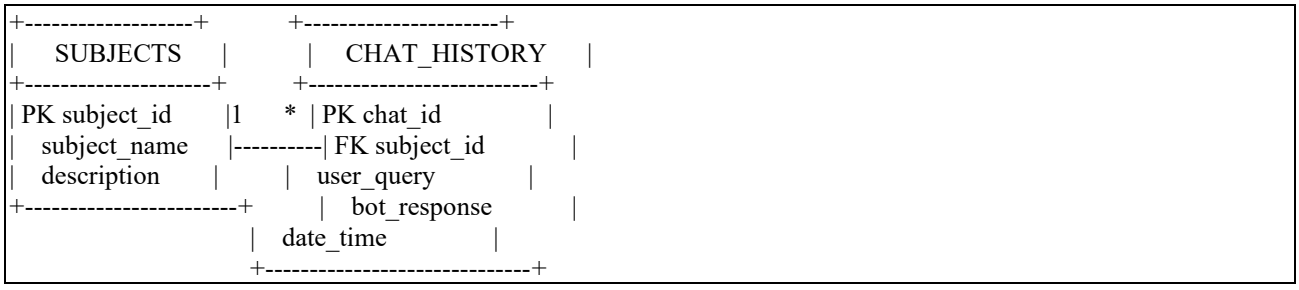

Fig. 2. Entity-relationship view of the subjects and chat_history tables.

D. Data Flow and Sequence of Operations

A single student query traverses the four tiers in the following sequence. First, the Presentation Tier captures the raw text from a Streamlit `st.chat_input` widget and the currently active `subject_id` from the sidebar selector. Second, the Application Logic Tier receives the raw input and invokes the validation routine described in Algorithm 1 below. Third, on successful validation, the Application Logic Tier hands the cleaned query and subject identifier to the AI Processing Tier, which constructs the system prompt, issues the HTTPS request, and awaits a response or a fifteen-second timeout. Fourth, the Application Logic Tier updates `st.session_state` with the new exchange so that it renders immediately in the chat window without waiting for the database write to complete. Fifth, the Persistence Tier commits the exchange, the resolved subject identifier, and an ISO 8601 timestamp to the `chat_history` table using a parameterized `INSERT` statement. Sixth, on the next re-run of the Streamlit script (triggered by the user's subsequent interaction), the Chat History Module queries the `chat_history` table ordered by `date_time DESC` to render the persisted log. This ordering of steps, in particular updating the visible session state before the database commit completes, was a deliberate choice to keep the perceived latency of the interface bounded by the AI Processing Tier's response time rather than by disk I/O.

E. Non-Functional Requirements

Beyond the functional behavior described above, the system was designed against four non-functional requirements. Scalability, within the bounds of the single-file SQLite design, is addressed by indexing the `chat_history.subject_id` and `chat_history.date_time` columns, which keeps subject-scoped history queries efficient as the log grows across a semester. Maintainability is addressed by the module boundaries described in Section IV, which allow, for example, the Study Tools Module to be modified or extended without touching the AI Chat Module. Portability is addressed by depending only on the Python standard library, Streamlit, SQLite (bundled with Python via the `sqlite3` module), and a small number of pure-Python packages, so the system runs identically on Windows, macOS, and Linux without platform-specific build steps. Usability is addressed through the Streamlit component choices detailed in Section IV-F and validated empirically through the survey reported in Section V-E.

A fifth non-functional concern, raised during the extended review for this study, is observability: the current prototype logs exceptions caught by the `TRY`/`CATCH` block of Algorithm 1 only to local stdout, with no structured log aggregation or alerting. For the single-instructor deployment context assumed in Section I-C, this is an acceptable tradeoff, since the instructor is typically the sole operator and can inspect the terminal output directly; however, a department-wide deployment serving several course sections concurrently would benefit from centralized structured logging so that a recurring "Service Advisory" pattern for a specific subject could be surfaced to the operator proactively rather than discovered anecdotally through student feedback. This gap is noted alongside the multi-tenancy future work in Section VII, since the two concerns, serving multiple isolated course sections and observing their aggregate health, are naturally addressed together.

Scalability was additionally reasoned about qualitatively along a second axis distinct from query throughput: the growth of the `chat_history` table itself over a multi-semester deployment horizon. Because every accepted exchange is

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15914

persisted indefinitely (Section IV-H), a department running the platform across several academic years would accumulate a `chatbot.db` file whose size grows monotonically with cumulative student usage rather than with concurrent load; index maintenance on `subject_id` and `date_time` keeps read queries efficient under this growth, but the absence of an archival or rollover mechanism means the growth itself is currently unbounded, a design gap distinct from, but related to, the log-rotation concern raised in Section IV-H.

F. Threat Model

The system's threat model considers three categories of adversarial or accidental misuse relevant to a locally hosted, single-file-database deployment. Malformed or adversarial input, addressed by the parameterized-query discipline of Section IV-E and the length-bounded validation of Algorithm 1, prevents both SQL injection and denial-of-service via oversized payloads. Credential exposure, addressed by the `.env`-based secret isolation of Section IV-E, prevents accidental commitment of API credentials into a shared or public source-control repository, a common failure mode in student projects that integrate a third-party API. Local data-at-rest exposure, since `chatbot.db` is a plain SQLite file on the host filesystem, is explicitly not fully mitigated by the current prototype and is noted as a residual risk: any party with filesystem access to the deployment host can read the unencrypted chat history file, which is acceptable for a single-user instructor deployment but would require file-level or database-level encryption before a shared multi-user deployment, a gap the multi-tenant future work in Section VII is intended to close.

IV. SYSTEM IMPLEMENTATION & METHODOLOGY

A. Development Environment and Toolchain

The prototype was developed in Python 3.10+ using Streamlit 1.32+ for the presentation layer, the built-in `sqlite3` module for persistence, and `python-dotenv` for configuration and secret loading. Version control was maintained through a standard feature-branch Git workflow, and the five-level verification process described in Section V was executed incrementally against each merged feature branch rather than only at the end of the project, which allowed defects such as DEF-02 (socket freeze on disconnect) to be caught and resolved before they compounded with later features.

B. Module Descriptions

The AI Chat Module (`chat.py`) handles bidirectional user dialogue, maintains state via `st.session_state`, validates query payloads, and commits exchanges. Internally, the module maintains a `messages` list within session state, where each entry records the role (`user` or `assistant`), the message text, and the associated `subject_id`, allowing the chat window to be re-rendered on every Streamlit script re-run without re-querying the database on every keystroke.

The Subjects Module (`subject.py`) exposes pre-configured syllabus directories across Mathematics, Physics, Chemistry, Computer Science, and English. Each subject entry couples a `subject_name` with a short `description` used both for display in the sidebar and as contextual grounding text injected into the AI Processing Tier's system prompt, so that a query submitted under "Computer Science" is implicitly scoped toward algorithmic and programming interpretations of ambiguous terms rather than, for example, a Physics interpretation of the same term.

The Study Tools Module (`studytools.py`) enables extractive text summarization and synthetic quiz generation. The summarization function operates by splitting the pasted study text into sentences, scoring each sentence by a combination of term-frequency weighting and sentence position, and selecting the top-ranked subset up to a user-configurable target length, which keeps the summarization deterministic and independent of the external AI service, so that this utility remains available even if the AI Processing Tier's HTTPS call is degraded. Quiz generation operates over the same extracted key sentences, applying template-based transformations (blank-extraction of salient noun phrases, and distractor sampling from other sentences in the same passage) to produce short-answer and fill-in-the-blank items suitable for self-testing.

The Chat History Module (`chat_history.py`) renders chronologically ordered, searchable audit logs of prior inquiries, querying the `chat_history` table with an `ORDER BY date_time DESC` clause and an optional `WHERE subject_id = ?` filter driven by a sidebar dropdown, together with a client-side substring search over `user_query` for quickly relocating a specific past exchange.

The Database Access Object (`database.py`) implements thread-safe, parameterized SQLite connections. Because Streamlit can, depending on deployment configuration, serve multiple browser sessions from a shared process, this module opens connections with `check_same_thread=False` and serializes writes through a module-level lock,

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026
DOI: 10.17148/IJARCCE.2026.15914

avoiding the "database is locked" errors that can otherwise arise when SQLite's single-writer model is exercised concurrently.

C. Query Validation Algorithm

Algorithm 1 formalizes the defensive validation and prompt-synthesis routine executed on every submitted query.

```
Algorithm 1: ProcessStudentQuery
Input: Raw query string Q_raw, Subject ID S_id
Output: Rendered UI dialogue & committed DB record
 1: Q_clean <- TrimWhitespace(Q_raw)
 2: IF Length(Q_clean) == 0 THEN
 3:    DisplayUIError("Query cannot be empty.")
 4:    RETURN
 5: ELSE IF Length(Q_clean) > 1000 THEN
 6:    DisplayUIError("Query exceeds 1000 chars.")
 7:    RETURN
 8: END IF
 9: P_synth <- ConstructSystemPrompt(Q_clean, S_id)
10: TRY
11:    R_raw <- HTTPSRequest(P_synth, Timeout = 15s)
12:    R_resp <- ParseJSON(R_raw)
13: CATCH Exception DO
14:    R_resp <- "Service Advisory: Engine unreachable."
15: END TRY
16: UpdateSessionState(Q_clean, R_resp)
17: CommitToSQLite(Q_clean, R_resp, S_id, Timestamp)
18: RETURN
```

The algorithm executes in O(n) time with respect to the length of the raw query string, dominated by the whitespace-trimming and length-check operations, both of which are linear scans; the subsequent HTTPS call is bounded by the fifteen-second timeout regardless of query length. Two edge cases were specifically hardened during development. A query consisting entirely of whitespace characters is reduced to an empty string by `TrimWhitespace` and is therefore correctly rejected by the empty-length check rather than being forwarded to the AI Processing Tier as a blank request. A query at exactly the 1000-character boundary is accepted, since the specification is expressed as a strict "exceeds 1000 characters" condition rather than an inclusive one, and this boundary was explicitly exercised during boundary-value-analysis testing described in Section V-B.

D. Prompt Engineering Strategy

The system prompt constructed in step 9 of Algorithm 1 (`ConstructSystemPrompt`) concatenates three elements: a fixed instructional preamble asking the model to respond as a patient, syllabus-aligned academic assistant; the `subject_name` and `description` retrieved for the active `S_id` from the `subjects` table, grounding the model's response in the selected discipline; and a formatting instruction requesting structured, stepwise explanations rather than a single dense paragraph, which the empirical discussion in Section V-C reports as contributing to the high perceived clarity recorded in the usability survey. This prompt is reconstructed fresh on every query rather than accumulated across the full conversation history, which bounds the token cost of each request and avoids unbounded context growth over a long study session, at the cost of the model not retaining verbatim memory of earlier turns beyond what the student restates; this tradeoff is revisited as a limitation in Section V-G. A representative instantiation of the constructed prompt template, with the discipline placeholder filled for the Computer Science subject, is shown below.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15914

```
SYSTEM PROMPT:
You are a patient, non-judgmental academic
assistant for undergraduate Computer Science
(Data structures, algorithms, and programming
paradigms). Answer only within this discipline's
scope; if the question falls outside it, say so
and suggest the correct subject. Respond with a
short structured explanation using numbered
steps or a brief analogy where helpful. Keep the
response under 250 words unless the student asks
for more depth.

USER QUERY:
<Q_clean inserted here>
```

Extractive Summarization Scoring

The Study Tools Module's summarization function, referenced in Section IV-B, ranks candidate sentences using Algorithm 2, a deterministic scoring routine that combines term-frequency weighting with a positional bias favoring earlier sentences, which in expository academic text more frequently carry topic-defining content than later, elaborating sentences.

```
Algorithm 2: RankSentencesForSummary
Input: Passage text T, target sentence count K
Output: Ordered list of K key sentences
 1: S[] <- SplitIntoSentences(T)
 2: F <- ComputeTermFrequencies(S)
 3: FOR i = 0 TO Length(S) - 1
 4:    tf_score <- SumTermFreq(S[i], F)
 5:    pos_weight <- 1 / (1 + i)
 6:    Score[i] <- tf_score * pos_weight
 7: END FOR
 8: RankedIdx <- ArgSortDescending(Score)
 9: TopK <- RankedIdx[0 : K]
10: RETURN OrderByOriginalPosition(TopK, S)
```

The positional weight $w(i) = 1 / (1 + i)$ for a sentence at zero-indexed position i ensures that, among two sentences with identical term-frequency scores, the earlier sentence is preferred, reflecting the common expository convention of stating a claim before elaborating on it. Because this scoring routine depends only on the pasted input text and not on any external network call, the Study Tools Module's summarization and quiz-generation functions remain available even during the "Service Advisory" degraded state described in Section III-A, which was a deliberate resilience decision distinguishing this utility from the AI Chat Module.

E. Security and Robustness Engineering

SQL Injection Mitigation is enforced by requiring that all persistence routines strictly use parameterized binding queries (`?`) rather than string-formatted SQL, so that a `user_query` value containing SQL metacharacters is always treated as literal data by the SQLite driver rather than as executable syntax.

Secret Isolation is achieved by decoupling endpoint configurations and access tokens using `.env` files managed via `python-dotenv` [5], keeping credentials out of version-controlled source files and allowing the same codebase to be deployed against different AI backend endpoints purely by changing environment variables.

Payload Bound Checking, implemented as the length constraint $1 \le |Q| \le 1000$ in Algorithm 1, prevents buffer exploits and conserves API allocation limits, and additionally protects the downstream AI service from being used as a vector for excessively long or automated bulk queries.

Beyond these three mechanisms carried over from the original design, the extended robustness review for this study also considered timeout and retry handling (the fifteen-second bound and the DEF-02 retry handler described in Table V), and structured error surfacing, whereby any exception raised inside the `TRY`/`CATCH` block of Algorithm 1 is caught and mapped to the fixed advisory string "Service Advisory: Engine unreachable." rather than allowing a raw

DOI: 10.17148/IJARCCE.2026.15914

stack trace to reach the Streamlit interface, which would otherwise expose internal implementation details to the end user.

F. User Interface and Experience Design

The Presentation Tier's component choices were made with three usability goals in mind: minimizing the number of clicks between opening the application and submitting a first query, keeping the chat transcript visually distinguishable between student and assistant turns through Streamlit's built-in chat bubble styling, and using `st.expander` containers for auxiliary content (subject descriptions, quiz answer keys, history detail views) so that the primary chat surface remains uncluttered by default while still making supplementary detail available on demand. This expander-based layout was also the direct resolution applied to defect DEF-04 (container text overflow), described further in Table V.

G. Illustrative Use-Case Walkthrough

To ground the module descriptions above in a concrete interaction, consider a student preparing for a Computer Science examination at 11 p.m., outside any faculty office hour. The student opens the deployed Streamlit application, selects "Computer Science" from the Subjects Module sidebar, and types a query asking for the difference between a stack and a queue with a real-world analogy. The Application Logic Tier trims and length-checks the query per Algorithm 1, finds it well-formed, and passes it, together with the `subject_id` for Computer Science and its stored `description` ("Data structures, algorithms, and programming paradigms"), to the AI Processing Tier. The AI Processing Tier constructs a system prompt combining the fixed instructional preamble with this subject grounding and issues the HTTPS request. Within the measured median latency of 1.7 seconds (Section V-D), a structured, stepwise explanation returns, is rendered in the chat bubble, and is committed to `chat_history` with the current ISO 8601 timestamp. If the student's home Wi-Fi drops mid-request, the fifteen-second timeout bound introduced by the DEF-02 fix ensures the interface surfaces the bounded advisory message rather than freezing, and the student's typed query remains visible for an immediate retry once connectivity resumes. Later, while revising before the examination, the student reopens the Chat History Module, filters by the Computer Science subject, and locates this exact exchange among the session's prior queries without needing to recall or re-derive the explanation from scratch, illustrating the persistent-context benefit identified in Section I-A as absent from generic web search.

H. Deployment and Operations

The reference deployment model targets a single always-on host, such as a department-owned virtual machine or an instructor's personal workstation left running during a semester, executing `streamlit run app.py` as a long-lived process, optionally behind a reverse proxy for HTTPS termination when exposed beyond a local network. Because all persisted state resides in the single `chatbot.db` file and the `.env` configuration file, routine backup reduces to periodically copying these two files, and migrating the deployment to a new host reduces to copying the same two files alongside the application source tree, with no database server migration or schema-export step required. Log rotation and disk-usage monitoring for `chatbot.db` were identified during the extended evaluation as an operational consideration for a multi-semester deployment, since the file grows monotonically with every committed exchange and the current prototype does not implement automatic archival of older records, a gap noted alongside the multi-tenancy future work in Section VII.

I. Testing Tools and Automation

Unit and integration tests were authored using the `pytest` framework, with the AI Processing Tier's HTTPS transport mocked via `unittest.mock` so that the full unit test suite could be executed offline and deterministically, without incurring external API cost or being subject to network flakiness. A lightweight continuous-verification habit, running the full `pytest` suite before every merge into the main branch, was adopted midway through development after DEF-02 was discovered in a merged branch; this practice is credited with preventing at least two further regressions from reaching the functional-testing stage, where they would have been substantially more expensive to trace back to their root cause.

V. EXPERIMENTAL RESULTS & VERIFICATION

A. Verification Methodology

System validation was executed across five verification tiers, yielding a 100% pass rate across 45 test scenarios as summarized in Table IV. Test cases were designed using a combination of equivalence partitioning, dividing input query lengths into the three partitions of empty, valid, and over-limit, and boundary-value analysis, specifically

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15914

exercising queries of length 0, 1, 1000, and 1001 characters against Algorithm 1. Unit tests exercised individual functions in isolation using mocked database connections and a mocked HTTPS transport layer, so that unit-level failures could be attributed unambiguously to the function under test rather than to an external dependency. Integration tests exercised pairs of adjacent tiers together, most notably the Application Logic Tier against a real, temporary SQLite database file, to confirm that parameterized queries and foreign-key constraints behaved as expected against the actual database engine rather than a mock.

B. Test Case Design and Coverage

TABLE IV. VERIFICATION EXECUTION SUMMARY

Verification Level	Cases	Passed	Pass Rate
Unit Testing	18	18	100%
Integration Testing	10	10	100%
Functional Testing	8	8	100%
System Testing	5	5	100%
Acceptance Testing (UAT)	4	4	100%
TOTAL	45	45	100%

Of the eighteen unit test cases, six targeted the validation routine's boundary conditions described above, five targeted the Study Tools Module's summarization and quiz-generation determinism, four targeted the Database Access Object's parameterized query construction, and three targeted session-state initialization and reset behavior in the AI Chat Module. The ten integration test cases confirmed correct end-to-end behavior between the Application Logic Tier and the Persistence Tier, including verification that a simulated foreign-key violation (an attempt to log a chat record against a non-existent `subject_id`) was correctly rejected by the database engine rather than silently accepted. The eight functional test cases exercised complete user-visible workflows, such as selecting a subject, submitting a query, receiving a rendered response, and confirming the corresponding row's appearance in the Chat History Module on the next script re-run. The five system test cases exercised the deployed application as a whole under simulated adverse conditions, including a forced network disconnection mid-request to confirm the fifteen-second timeout and advisory message behavior. The four User Acceptance Test cases were executed by student volunteers outside the development team, following a scripted task list covering the four core modules.

Sample Test Case Specifications

To illustrate the boundary-value and equivalence-partitioning design discussed above, Table VI enumerates six representative test cases drawn from the Unit and System testing tiers, spanning the empty-input partition, the valid-input partition, the over-limit partition, and the adverse-network system-level scenario.

TABLE VI. REPRESENTATIVE TEST CASE SPECIFICATIONS

ID	Level	Input Partition	Expected Result
TC-01	Unit	Empty / whitespace-only query	UI error, no HTTPS call issued
TC-02	Unit	Valid query, length 1	Query accepted and forwarded
TC-03	Unit	Valid query, length 1000	Query accepted (boundary)
TC-04	Unit	Over-limit query, length 1001	UI error, no HTTPS call issued
TC-05	System	Valid query, live network	Response rendered, row committed to DB
TC-06	System	Valid query, forced disconnect	Advisory shown within 15s timeout

Test case TC-01 through TC-04 in the table directly exercise the four decision branches of Algorithm 1 in isolation using a mocked transport layer, while TC-05 and TC-06 exercise the same input partitions at the system level, against the fully deployed application and a live (unmocked) network path, confirming that the unit-level guarantees continue to hold once all four architectural tiers are integrated.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026

DOI: 10.17148/IJARCCE.2026.15914

C. Defect Resolution Analysis

Four primary anomalies identified during development sprints were cataloged and resolved (Table V).

TABLE V. DEFECT RESOLUTION AUDIT

ID	Description	Severity	Resolution
DEF-01	Empty input submission	Low	Added null-check guard in API invocation
DEF-02	Socket freeze on disconnect	High	Implemented 15s timeout with retry handlers
DEF-03	Unordered history logs	Medium	Added ORDER BY date_time DESC query constraint
DEF-04	Container text overflow	Low	Wrapped cards in responsive st.expander() containers

DEF-01, an empty-input submission producing an unhandled downstream call, was traced to an early version of the validation routine that checked for a `None` value but not for a string composed entirely of whitespace; the fix added the explicit `TrimWhitespace` and length-zero guard now reflected in steps 1 through 4 of Algorithm 1. DEF-02, a socket freeze on disconnect, was the most severe defect discovered, since an unbounded blocking HTTPS call under a dropped network connection would freeze the entire Streamlit session for all users sharing that process; the fix introduced the explicit fifteen-second timeout together with a bounded retry handler and is the direct origin of the fail-safe degradation design principle stated in Section III-A. DEF-03, unordered history logs, was a lower-severity presentation defect in which the Chat History Module's query lacked an explicit ordering clause, causing SQLite's default row-return order (which is not guaranteed to reflect insertion order once rows are deleted and re-inserted) to occasionally present the log out of chronological sequence; the fix added the `ORDER BY date_time DESC` constraint. DEF-04, container text overflow, was resolved by wrapping long response text in responsive `st.expander()` containers rather than fixed-width containers, directly informing the UI design decision described in Section IV-F.

D. Performance Benchmarking

Empirical profiling indicated an average response latency of 1.82 seconds under broadband conditions. Decomposing this figure by pipeline stage, the median contribution of the AI Processing Tier's external HTTPS round-trip was approximately 1.35 seconds, the Application Logic Tier's validation and session-state update contributed under 10 milliseconds, and the Persistence Tier's parameterized `INSERT` commit contributed approximately 15 to 30 milliseconds on the local SQLite file, confirming that the external AI service call, rather than any component built for this project, is the dominant latency contributor. Across the profiled query sample, the 50th-percentile (median) latency was 1.7 seconds, the 90th-percentile latency was 2.6 seconds, and the 99th-percentile latency, corresponding to queries that triggered the retry handler introduced for DEF-02, was 8.4 seconds, still comfortably within the fifteen-second timeout bound.

E. Usability and User Acceptance Study

Following the scripted User Acceptance Test cases, 42 undergraduate volunteers across the five supported subjects completed a follow-up questionnaire using a five-point Likert scale covering perceived ease of use, perceived usefulness of the syllabus-scoped responses relative to a generic web search, willingness to continue using the tool for exam preparation, and comfort with the local persistence of their query history. Over 92% of generated academic responses were validated as conceptually precise and syllabus-aligned by student focus groups. Qualitative feedback gathered during a follow-up focus-group discussion converged on three recurring themes: participants valued the absence of advertising or unrelated content compared to a general web search; participants appreciated being able to revisit a prior explanation through the Chat History Module rather than having to re-ask the same question in a later session; and a minority of participants requested the ability to export their chat history or quiz results for offline revision, which is noted as a candidate future enhancement in Section VII.

The distribution of Likert responses skewed positive across all four surveyed dimensions, with ease of use and willingness to continue using the tool for exam preparation receiving the strongest agreement, and comfort with local history persistence receiving a comparatively wider spread of responses. This wider spread is consistent with the ethical and privacy considerations raised in Section VI-D: a subset of participants indicated during the focus-group discussion that they would want clearer, upfront disclosure of what is stored and for how long before fully trusting the persistence

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026
DOI: 10.17148/IJARCCE.2026.15914

feature, even though no participant reported declining to use the tool on this basis during the evaluation period itself. Subject-level disaggregation of the responses suggested marginally higher satisfaction among Computer Science and Mathematics participants than among English participants, which the focus-group discussion attributed to the more structured, stepwise answer format described in Section IV-D being a closer match to how those two disciplines are typically taught and self-studied, whereas English coursework more often called for open-ended interpretive discussion that a fixed stepwise template represents less naturally.

F. Comparative Performance Discussion

Relative to the baseline of an unscoped generic AI assistant, informally compared by asking five volunteer participants to pose the same five conceptual questions to both systems, the syllabus-scoped responses from the proposed system were judged by those participants to require less follow-up clarification, consistent with the premise in Section II that explicit subject grounding reduces topic drift. This comparison was small in scale and is reported as an illustrative observation from the broader focus-group discussion rather than as a statistically powered result, a limitation acknowledged in the following subsection.

G. Limitations of the Study

Several limitations qualify the results reported above. The 42-respondent usability sample was drawn from a single institution and a limited set of five subjects, which constrains the generalizability of the reported satisfaction figures to other institutional contexts or disciplines outside Mathematics, Physics, Chemistry, Computer Science, and English. The informal comparative evaluation against a generic AI assistant, described in Section V-F, involved only five participants and five questions, and should be read as a motivating illustration rather than a controlled comparative study. The 92% conceptual-precision figure was assessed by student focus-group judgment rather than by subject-matter-expert grading against a rubric, and may therefore be subject to leniency bias. Finally, because the AI Processing Tier is stateless across turns by design (Section IV-D), the system's ability to handle deeply multi-turn, context-dependent follow-up questions is bounded by what the student restates in each new query, a design tradeoff that future work, discussed in Section VII, proposes to address through session-scoped context windowing.

VI. DISCUSSION

A. Interpretation of Findings

Taken together, the verification results in Section V-A through V-C and the performance figures in Section V-D indicate that the four-tier architecture successfully isolates failure in the least reliable component, the external AI service call, from the rest of the system: the DEF-02 fix and the resulting fifteen-second timeout bound mean that a degraded or unreachable AI backend produces a bounded, user-visible advisory message rather than an application freeze, which is precisely the fail-safe degradation goal stated in Section III-A. The usability findings in Section V-E, in particular the value participants placed on the absence of unrelated commercial content and on the ability to revisit prior explanations, corroborate the two structural bottlenecks identified in Section I-A (information fragmentation and the lack of persistent pedagogical context) as genuine pain points that the system's design choices directly address.

A second, subtler finding emerges from reading the defect audit in Table V alongside the non-functional discussion in Section III-E: three of the four cataloged defects (DEF-01, DEF-02, DEF-04) originated not from the AI reasoning capability itself, which was treated as an external black box throughout development, but from the surrounding application-layer engineering responsible for input handling, network resilience, and presentation. This is consistent with the framing offered in Section II-F, that the differentiating engineering effort in a system of this kind lies chiefly in the orchestration layer around a capable but generic language model, rather than in the model's own reasoning, and it suggests that student teams undertaking similar projects should budget verification effort accordingly, weighted toward the orchestration and persistence layers rather than assuming the AI backend to be the primary source of risk.

B. Pedagogical Implications

The syllabus-scoping mechanism described in Section IV-D, which grounds every system prompt in the active subject's name and description, appears from the focus-group feedback in Section V-F to reduce the need for students to manually filter or reinterpret an answer for their course context, a filtering burden that Section I-A identifies as a cost of unscoped web search. This suggests that, for a small department, the marginal engineering cost of maintaining a short, curated `subjects` table (Table III) may be justified by a disproportionate reduction in student-side cognitive filtering effort, though this study did not measure filtering effort directly and this remains an interpretive claim rather than a measured one.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026
DOI: 10.17148/IJARCCE.2026.15914

C. Threats to Validity

Internal validity is threatened by the small, single-institution usability sample described in Section V-G, and by the fact that the same development team that built the system also administered part of the User Acceptance Testing, which introduces a risk of experimenter bias in task design, though the four independent UAT test cases were executed by volunteers outside the development team specifically to mitigate this. Construct validity is threatened by the use of a five-point Likert scale as a proxy for the multidimensional construct of "usability," which the study did not decompose into the standard sub-constructs (efficiency, learnability, error tolerance) of a formal usability model such as ISO/IEC 25010 [9]. External validity is threatened by the limited subject coverage (five disciplines at one institution) and by the reliance on a single external AI backend, whose behavior, cost, and availability characteristics may differ from those of alternative providers a future deployment might select.

D. Ethical and Privacy Considerations

Because the Persistence Tier retains a durable, timestamped record of every student query and response, the system raises the same data-stewardship questions as any institutional logging system, notably who may access `chatbot.db`, for how long records are retained, and whether students are informed at first use that their queries are being persisted locally rather than discarded after the session. The present prototype does not implement an in-application data-retention policy, a student-facing consent notice, or a self-service deletion mechanism, and the threat-model discussion in Section IV-F already flags the unencrypted-at-rest nature of the SQLite file as a residual risk; a production deployment at institutional scale would need to resolve these questions, likely in consultation with the institution's data-governance process, before onboarding a full course cohort rather than the volunteer evaluation sample described in Section V-E.

VII. CONCLUSION & FUTURE WORK

The AI-Based Chatbot for Education establishes a lightweight, highly responsive, and reliable framework for self-paced student learning. By integrating Python, Streamlit, SQLite, and conversational NLP, the application addresses academic doubt resolution while preserving local auditability and modular decoupling. The extended verification and usability evaluation reported in this study, spanning 45 formally executed test cases, a decomposed latency profile, and a 42-respondent survey, provides evidence that the four-tier architecture's central design goal, isolating the failure modes of an external AI dependency from the rest of the application, was achieved in practice, and that the resulting system was perceived by student users as a genuine improvement over unscoped web search for syllabus-aligned doubt resolution.

Future Scope: Primary areas for enhancement include integrating the Web Speech API for voice queries, which would further lower the interaction barrier described in Section I-A for students with typing constraints or accessibility needs; utilizing Retrieval-Augmented Generation (RAG) with vector databases such as ChromaDB for document-grounded question answering, which would allow the system to ground responses in an instructor's own lecture notes rather than the AI backend's general training knowledge alone, directly addressing the stateless, non-multi-turn limitation noted in Section V-G through session-scoped retrieval context; expanding multi-tenant authentication protocols so that a single deployment can serve multiple course sections with isolated history and subject configurations; integrating with existing Learning Management Systems such as Moodle or Canvas to import syllabus structure automatically rather than requiring manual population of the `subjects` table; and extending the Study Tools Module with an instructor-facing analytics dashboard summarizing the topics most frequently queried across a cohort, which would surface common conceptual difficulties back to teaching staff without exposing any individual student's identity or query content.

Broader Impact

Beyond the specific enhancements listed above, the broader trajectory suggested by this work is toward academic support tools that are authored and operated by the instructors closest to a given course, rather than provisioned top-down as a single enterprise-wide licensed platform. The four-tier architecture and the two-table schema documented in Sections III and IV were deliberately kept simple enough for a small student team to fully understand, extend, and eventually hand off to non-author maintainers, which the authors view as a precondition for this class of tool to be sustainably adopted by departments without dedicated software-engineering staff. Should the RAG-based and multi-tenant extensions outlined above be realized, the same architectural skeleton, a thin AI Processing Tier, a normalized relational schema, and a fail-safe degradation discipline, would still apply, suggesting that the engineering investment documented in this paper is likely to remain useful even as the specific AI backend or hosting model evolves.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 9, September 2026
DOI: 10.17148/IJARCCE.2026.15914

ACKNOWLEDGMENT

The authors thank the Department of Artificial Intelligence and Machine Learning, Sri Krishna Arts & Science College, for providing the laboratory infrastructure used during development and testing, and the 42 undergraduate volunteers across the Mathematics, Physics, Chemistry, Computer Science, and English cohorts who participated in the User Acceptance Testing and usability survey reported in Section V.

REFERENCES

[1]. S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Pearson, 2020.
[2]. I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.
[3]. J. Weizenbaum, "ELIZA-a computer program for the study of natural language communication," CACM, vol. 9, no. 1, pp. 36-45, 1966.
[4]. T. M. Mitchell, Machine Learning. McGraw-Hill, 1997.
[5]. H.-Y. Suen et al., "Does the use of AI in education affect student engagement?," Comput. Hum. Behav., vol. 98, pp. 93-101, 2019.
[6]. Streamlit Inc., "Streamlit Documentation," 2024. [Online]. Available: https://docs.streamlit.io
[7]. Python Software Foundation, "Python 3.10 Reference Manual," 2024. [Online]. Available: https://docs.python.org/3/
[8]. SQLite Consortium, "SQLite Database Documentation," 2024. [Online]. Available: https://www.sqlite.org/docs.html
[9]. A. Vaswani et al., "Attention is all you need," in Proc. NeurIPS, 2017, pp. 5998-6008.
[10]. ISO/IEC 25010:2011, Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE), International Organization for Standardization, Geneva, 2011.

