

Campus Ai: A Chatbot Using Natural Language Processing for College Assistance

Athul Krishna S¹, Chandrika K², Sanjay G³, Sanjay S⁴

Dept. Of Artificial Intelligence And Data Science Sengunthar Engineering College (Autonomous) Tiruchengode, Tamil Nadu, India

Abstract—Students and faculty at engineering colleges often search several portals, notices and documents to find routine academic information. This paper presents Campus AI, a web-based chatbot prototype that lets users ask college-related questions in natural language. The system is implemented as a Next.js and React application. A server-side chat endpoint applies per-IP sliding-window rate limiting and input sanitization, assembles a prompt from an institutional system prompt, the latest eight conversation messages and, when available, the authenticated student's profile, and then queries the Llama 3.1 70B model hosted on the Groq platform in JSON mode. The reply is returned as a structured object containing the answer text, an optional generated visual and source labels, and is rendered in a conversational interface alongside role-specific dashboards for students, faculty and administrators. We describe the architecture, request workflow and implementation, and we separate the features that are implemented in the prototype from those that are only planned. Functional behaviour is demonstrated through screenshots of the working prototype. No quantitative evaluation of answer accuracy, response time or user satisfaction has been carried out; such an evaluation is identified as future work.

Index Terms—chatbot, natural language processing, large language model, college information system, conversational agent, educational technology, role-based access

I. INTRODUCTION

Academic and administrative information in a college is spread over student portals, examination notices, circulars, handbooks and departmental channels. A student who wants to know an attendance rule or an examination regulation must first decide where to look and then read through lengthy documents. Faculty and office staff, in turn, answer the same routine questions repeatedly. Chatbots have been studied widely as a way of giving learners and staff immediate, conversational access to such information [1]–[3].

This paper describes Campus AI, a chatbot prototype developed in the Department of Artificial Intelligence and Data Science of Sengunthar Engineering College. Users type questions in everyday language; a large language model (LLM) interprets the question and composes the answer, and the interface presents it together with an optional explanatory diagram and source labels.

The same web application also provides dashboards for students, faculty and administrators. The objectives of the project are to (i) provide a single conversational entry point to college information, (ii) personalise answers using the profile of the authenticated student, (iii) protect the service against abusive request rates and malformed input, and (iv)

provide a modular base for later extensions. The contributions of this paper are:

- a description of the Campus AI architecture and request workflow as implemented in the supplied source code;
- a structured-response design in which the model returns an answer, an optional visual request and source labels as a single JSON object, with a fallback when parsing fails;
- a candid account of what the prototype demonstrates, what it does not, and which parts of the original project report remain proposals.

The remainder of the paper is organised as follows. Section II reviews related work, Section III analyses existing practice and the research gap, Sections IV–VII present the proposed system, its architecture, workflow and implementation, Section VIII discusses the observed results, Sections IX–X cover limitations and future work, and Section XI concludes.

II. RELATED WORK

Systematic reviews of educational chatbots report applications such as answering frequently asked questions, supporting administrative services and assisting learning, and they note that most deployed systems were rule-based or retrieval-based and limited to predefined topics [1]–[3]. Broader reviews of artificial intelligence in education highlight opportunities in personalised support and

administration, together with open challenges in ethics, evaluation and teacher readiness [4], [5].

The arrival of generative models has shifted the discussion toward open-ended conversational assistants. Reviews of generative AI in education describe benefits for access to information and personalised feedback, but also concerns about accuracy and over-reliance [6], [7]. An exploratory study of a general-purpose chatbot on science-education tasks likewise found that answers need careful checking by the user [8].

Technically, current LLMs build on the Transformer architecture [9], and openly available model families such as Llama 3 provide strong general language capability [10]. Retrieval-augmented generation (RAG) combines a generator with an external document store so that answers can be grounded in a controlled corpus [11], whereas LLMs used without grounding may produce fluent but unsupported statements, commonly called hallucinations [12]. Campus AI sits between these positions: it uses a hosted LLM guided by an institutional system prompt and student context, and it identifies document retrieval as an extension (Section X).

III. EXISTING SYSTEM AND RESEARCH GAP

The project report characterises current practice at the institution in the following way.

- Scattered sources. Examination, attendance and regulation information is held in different portals, notices and documents, so users must first identify the right source.
- Manual document search. Handbooks, circulars and regulations are mostly PDF files that must be opened and searched by hand.
- Rule-based chatbots. Where chatbots are used, they depend on predefined questions and keywords and handle differently worded queries poorly.
- Separate data and documents. Structured records (attendance, marks, timetables) and unstructured documents are managed independently.
- Repetitive enquiries. Staff spend time answering identical routine questions.

The gap addressed by Campus AI is the lack of a single conversational interface that tolerates free-form phrasing, knows who the user is, and presents the answer in a readable form.

We do not claim that the prototype closes this gap fully: integration with institutional documents is not part of the supplied implementation (Table I).

IV. PROPOSED SYSTEM

Campus AI is a web application with three user roles (student, faculty, administrator). Students can chat with the “SEC AI Assistant”, view an academic summary (attendance, CGPA, backlogs, placements, events) and reach modules such as marks, leave, timetable and materials. Faculty screens include department students, attendance roll call, marks and CGPA entry, and leave approvals. An administrator view summarises students, staff, announcements and placements. Because the original report mixes implemented work with proposals (for example IoT classroom automation, complaint tickets and document indexing), Table I records the status of each capability according to the evidence available in the supplied report, source code and screenshots.

Table I. Implementation Status Of Campus Ai Capabilities

Capability	Status
Natural-language chat through an LLM	Implemented
Per-IP sliding-window rate limiting	Implemented
Input sanitization and length cap	Implemented
Student-context personalisation	Implemented
Structured answer with visual and sources	Implemented
Student, faculty and admin screens	Implemented (screenshots)
Upload and indexing of college documents	Described; no code supplied
Database schema and queries	Described; no code supplied

Capability	Status
IoT monitoring and classroom automation	Described; no code supplied
Complaint and notification modules	Described; no code supplied
Voice, Tamil/English, mobile app, MFA	Future work

V. SYSTEM ARCHITECTURE

Fig. 1 shows the three tiers of the implemented system. The client tier is a React application written in TypeScript. It contains the chat page, a message renderer that displays text, a zoomable visual card and source badges, and the role-based

dashboards. The server tier is a single POST route (/api/chat) that performs rate limiting, validation, prompt assembly, response parsing and visual resolution. The inference tier is the Groq API serving Llama 3.1 70B.

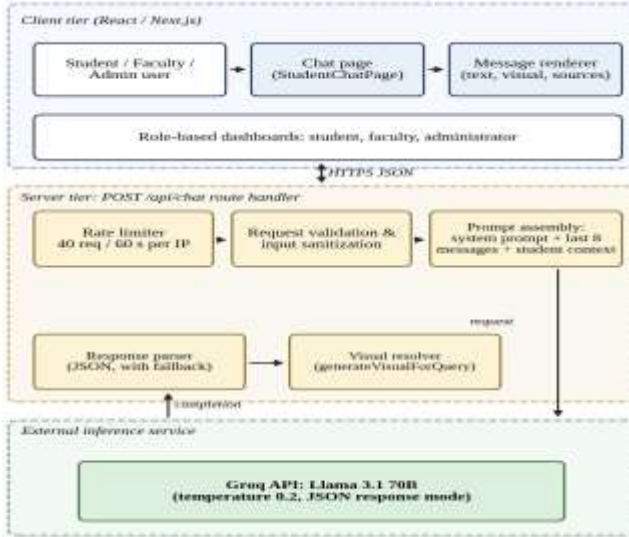


Fig. 1. Implemented architecture of Campus AI.

Student data used for personalisation (name, register number, department and year) is read on the client through application hooks and sent with each chat request as studentContext. The storage technology behind these hooks is not included in the supplied excerpts and is therefore not described here.

VI. METHODOLOGY AND WORKFLOW

The request workflow is summarised in Fig. 2 and proceeds as follows.

1. **Submission.** The chat page appends the user's message to the conversation and posts the message list and optional student context to /api/chat.

1. **Rate limiting.** The client IP is taken from the x-forwarded-for header. A sliding-window counter keeps the timestamps of recent requests per IP. With window length $W = 60$ s and limit $L = 40$, a request is accepted when the number of timestamps inside the window satisfies (1). Otherwise the server returns HTTP 429 with Retry-After: 60.

$$n = |\{ t_i \in T : t_i > t_{now} - W \}|, \text{ accept iff } n < L \quad (1)$$

1. **Validation and sanitization.** A missing or empty message array yields HTTP 400. The last message is trimmed, script

tags, HTML tags and javascript: prefixes are removed, and the text is truncated to 1000 characters.

2. **Prompt assembly.** The prompt consists of a system prompt that defines the institutional assistant persona, the most recent eight messages, and, if present, an additional system message carrying the authenticated student's name, department, year and register number.

3. **Generation.** The prompt is sent to Llama 3.1 70B through the Groq SDK with temperature 0.2, a limit of 2200 output tokens and JSON object response format.

4. **Parsing.** The reply is parsed as JSON containing answer, visual_required, visual_type, visual_prompt and sources. If parsing fails, the raw text is used as the answer with no visual.

5. **Visual resolution.** If a visual is requested, a helper service returns a visual URL and a source label.

6. **Response and rendering.** The structured object is returned to the client and rendered. If the request fails, the interface shows a connection-error message and a status indicator turns red.

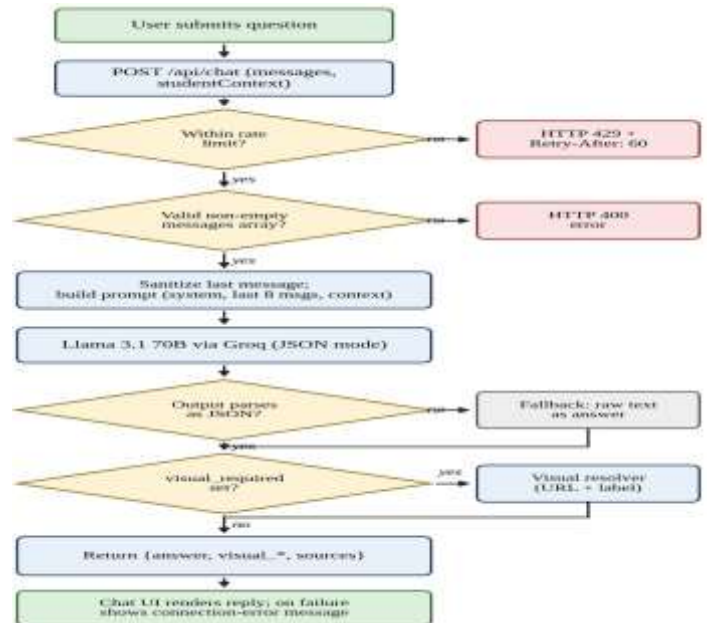


Fig. 2. Request workflow of the /api/chat endpoint.

VII. SYSTEM IMPLEMENTATION AND TECHNOLOGIES

Table II lists the technologies identified in the source code. The project report also mentions Flask or Django, MySQL or PostgreSQL, NLTK and spaCy as candidate technologies; these do not appear in the supplied implementation and are not treated as part of it.

Table II. Technologies Identified In The Source Code

Component	Technology / setting
Web framework	Next.js route handler (NextRequest, NextResponse)
Front end	React hooks, TypeScript, utility-class styling, lucide-react icons
LLM	Llama 3.1 70B (llama-3.1-70b-versatile) via groq-sdk
Generation settings	Temperature 0.2; 2200 max tokens; JSON mode
Context	Last 8 messages plus student profile
Protection	Sliding-window limiter (40 req/60 s per IP); sanitizer
Identity	Client-side auth context and student-profile hook

- **Chat interface.** The chat page keeps the message list, typing indicator and error state in React state, scrolls to the newest message, allows the conversation to be cleared, and shows a welcome message. The message renderer displays plain text with preserved line breaks, a visual card with zoom and open-in-new-tab actions, and a row of source badges.
- **Error handling.** The server distinguishes rate-limit (429), bad-request (400) and server (500) errors, and falls back to raw text when the model output is not valid JSON. The client shows a friendly message when the call fails.
- **Security measures.** The rate limiter and sanitizer are implemented as small TypeScript functions. The report additionally states design intentions such as password hashing and role-based access control; the corresponding enforcement code is not part of the supplied excerpts.

VIII. RESULTS AND DISCUSSION

1. Functional Demonstration

Fig. 3(a) shows the chat page answering a question about the department's data-science architecture. The reply contains a labelled diagram card followed by structured text, and the header shows the connection state and the model in use. Fig. 3(b) shows the student dashboard with an attendance indicator, CGPA, backlog, placement and event counters, and quick-access tiles including the chatbot. The records appear to form a small prototype dataset.



Fig. 3. (a) Chat interface with generated visual. (b) Student dashboard.

Fig. 4(a) shows the faculty options screen with links to department students, attendance roll call, marks and CGPA, and leave approvals. Fig. 4(b) shows a longer assistant answer containing an ordered workflow description.



Fig. 4. (a) Faculty options. (b) Long assistant response.

Discussion

The screenshots demonstrate that the end-to-end path (interface, endpoint, model, structured reply, rendering) works and that role-specific screens exist. They do not demonstrate correctness. The example queries shown are general questions about AI and data-science pipelines rather than institution-specific questions, so they do not show that the assistant answers college rules correctly.

Fig. 4(b) also reveals a rendering limitation: a table generated by the model appears with raw pipe characters and literal `<br>` tags, because the renderer displays text with preserved whitespace rather than parsing Markdown. Either a

Markdown renderer or an instruction in the system prompt to avoid tables would remove this problem.

The report describes module-by-module functional testing, but it provides no test cases, counts or outcomes. No measurements of answer accuracy, response latency, throughput or user satisfaction were supplied. Consequently, this paper makes no claim of improved accuracy, reduced response time or superior performance over existing systems; quantitative evaluation remains future work.

IX. ADVANTAGES AND LIMITATIONS

Advantages

- Free-form questions are accepted, so users need no menus or exact keywords.
- Answers can be personalised with the logged-in student's profile.
- The structured JSON reply separates text, visuals and sources, which simplifies rendering and later extension.
- Basic abuse protection (rate limit, input cleaning) and explicit error paths are in place.
- The modular Next.js design allows additional roles, modules and data sources to be added.

Limitations

- Grounding. The supplied code contains no document retrieval; answers depend on the model and the system prompt. A low temperature reduces randomness but does not remove hallucination [12], and the default source label in the fallback path is not evidence that a handbook was consulted.
- Sanitization scope. In the listed route handler the sanitized text is used for the visual-resolver path, while the messages forwarded to the model are passed unchanged. Prompt-injection defences are not described.
- Trust in client data. `studentContext` is read from the request body; the excerpt shows no server-side verification of identity, so personalisation should not be treated as access control.
- Rate limiter. Counters are held in process memory and keyed by a client-supplied header, so they are not shared across server instances and can be bypassed by spoofing.
- External dependency. The service depends on a third-party hosted model and on the availability of the named model version.
- Evaluation. No quantitative evaluation has been done.

X. FUTURE ENHANCEMENTS

- Grounded answers. Index authorised documents (regulations, circulars, notices) and add retrieval-augmented generation [11] with cited passages.
- Evaluation. Build a labelled question set from real student queries and measure answer correctness, latency and user-reported usability.
- Security. Verify sessions on the server, derive student context from the session, use a shared rate-limit store, apply sanitization to model inputs, and add audit logs and multi-factor authentication.
- Interaction. Support Tamil and English, voice input, and a mobile application; render Markdown responses correctly.
- Integration. Connect to existing ERP, examination and attendance systems and add personalised notifications.
- Smart-campus modules. Implement the IoT monitoring, classroom automation and complaint modules described in the report as separate, evaluated components.

XI. CONCLUSION

Campus AI is a working web prototype that offers a conversational interface to college information, personalised with the signed-in student's profile and backed by Llama 3.1 70B on Groq. Its server pipeline adds rate limiting, input cleaning, structured JSON replies and error fallbacks, and the application provides student, faculty and administrator screens. The evidence supplied supports the claim that the pipeline functions; it does not yet support claims about accuracy, speed or user satisfaction. The most valuable next steps are grounding answers in institutional documents, hardening server-side security, and running a systematic evaluation with real users.

REFERENCES

1. S. Okonkwo and A. Ade-Ibijola, "Chatbots applications in education: A systematic review," *Computers and Education: Artificial Intelligence*, vol. 2, Art. no. 100033, 2021.
2. L. Labadze, M. Grigolia, and L. Machaidze, "Role of AI chatbots in education: systematic literature review," *Int. J. Educ. Technol. High. Educ.*, vol. 20, Art. no. 56, 2023.
3. D. Ramandanis and S. Xinogalos, "Designing a chatbot for contemporary education: A systematic literature review," *Information*, vol. 14, no. 9, Art. no. 503, 2023.
4. S. Wang, F. Wang, Z. Zhu, J. Wang, T. Tran, and Z. Du, "Artificial intelligence in education: A systematic literature review," *Expert Syst. Appl.*, vol. 252, Art. no. 124167, 2024.

5. T. K. F. Chiu, Q. Xia, X. Zhou, C. S. Chai, and M. Cheng, "Systematic literature review on opportunities, challenges, and future research recommendations of artificial intelligence in education," *Computers and Education: Artificial Intelligence*, vol. 4, Art. no. 100118, 2023.
6. H. Yu and Y. Guo, "Generative artificial intelligence empowers educational reform: current status, issues, and prospects," *Front. Educ.*, vol. 8, Art. no. 1183162, 2023.
7. C. K. Lo, "What is the impact of ChatGPT on education? A rapid review of the literature," *Educ. Sci.*, vol. 13, no. 4, Art. no. 410, 2023.
8. G. Cooper, "Examining science education in ChatGPT: An exploratory study of generative artificial intelligence," *J. Sci. Educ. Technol.*, vol. 32, pp. 444–452, 2023.
9. A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems* 30, 2017.
10. A. Dubey et al., "The Llama 3 herd of models," *arXiv:2407.21783*, 2024.
11. P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems* 33, 2020.
12. Z. Ji et al., "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, Art. no. 248, 2023.