International Journal of Engineering Research and Science & Technology

ISSN 2319-5991 www.ijerst.com
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

Research Paper

MY TALKIE - AI CHATBOT WITH VOICE ASSISTANCE

Reddy Joshma Devi
PG MCA Scholar, Department of Computer Science
Sir C R Reddy College, Eluru, Andhra Pradesh, India - 534005
joshreddy408@gmail.com

Banisetti Jyothi Satya Bhavani
Professor, Department of Computer Science
Sir C R Reddy College, Eluru, Andhra Pradesh, India - 534005
jyothiganesh006@gmail.com

Musunuru Ratnakar
HOD, Department of Computer Science
Sir C R Reddy College, Eluru, Andhra Pradesh, India - 534005

***Abstract*— Educational institutions repeatedly answer questions about admissions, examinations, attendance, fees, scholarships, placements, library services, and campus facilities. This paper presents My Talkie, a web-based AI chatbot with voice assistance and document-analysis support for student queries. The application uses Python and Flask for request processing, HTML, CSS, and JavaScript for the interface, and the Groq API with the Llama-3.1-8B-Instant model for context-aware response generation. Students can enter questions as text or use browser speech recognition. Responses are displayed in the chat window and can be read aloud through speech synthesis. The system also accepts image, PDF, and text-file uploads: Tesseract OCR extracts text from images, pdfplumber reads PDFs, and text files are processed directly before the recovered content is supplied to the language model. The project architecture separates the user interface, Flask backend, file-processing services, AI service, and voice services. The documented executions demonstrate text-based question answering, voice interaction, image OCR, PDF explanation, text-file analysis, and college-oriented responses through a compact browser interface. The resulting prototype provides a practical foundation for continuous student assistance and can be extended with authentication, college ERP connectivity, multilingual interaction, chat history, cloud deployment, and personalized academic services.**

Keywords*— *Educational chatbot, Voice assistance, Groq API, Llama 3.1, Flask, Speech recognition, Text-to-speech, Tesseract OCR, PDF processing.

I. INTRODUCTION

Colleges communicate a large volume of information to students through offices, websites, notices, circulars, PDFs, images, and faculty interactions. Questions about admission procedures, course offerings, examination schedules, attendance requirements, fee payments, scholarships, placements, library rules, and campus services are frequently repeated. A manual support process depends on office hours and staff availability, and students may have to search several sources before obtaining a clear answer. Conversational software offers a direct interface in which the student states a question naturally and receives an immediate response.

Chatbots have evolved from pattern-based programs such as ELIZA [2] to systems that use natural-language processing and large language models. Reviews of educational chatbot research show applications in information delivery, learning support, mentoring, assessment, and administrative assistance [5], [6]. For institutional support, the important characteristics are availability, ease of use, natural-language interaction, and the ability to present information through channels that students already understand. Voice interaction further improves accessibility by allowing hands-free input and spoken output.

My Talkie is designed as a browser-based student assistant. The frontend offers a text field, microphone control, upload button, send button, clear control, and response panel. Flask receives questions and files, prepares the prompt, communicates with the Groq service, and returns a JSON response. The deployed model identified in the project is Llama-3.1-8B-Instant, a member of the Llama 3 family of transformer language models [13]. Because the system uses an external language-model service rather than fixed keyword rules, differently phrased questions can be handled through contextual generation.

The project extends ordinary question answering with multimodal document input. An image can contain a notice or assignment; Tesseract OCR converts visible text into machine-readable form [15]. A PDF can be read with pdfplumber, while a text file is decoded directly. The extracted material is combined with the user request and supplied to the AI model for explanation or summarization. The browser Web Speech API supports speech recognition and speech synthesis, enabling the same answer to be presented as both text and audio [21].

The contributions of this work are: (1) an integrated student-support chatbot using Flask and a hosted large language model; (2) text and voice interaction within one interface; (3) image, PDF, and text-file analysis; (4) a modular client-server architecture; and (5) documented end-to-end outputs covering general AI questions, uploaded documents, and college-specific inquiries. The paper follows the structure of the supplied conference sample and reports only the functions and outputs represented in the project document.

II. RELATED WORK

Turing proposed the imitation game as an operational discussion of machine intelligence [1], while Weizenbaum demonstrated early natural-language conversation through ELIZA [2]. Subsequent chatbot research examined whether conversational agents are useful interfaces and distinguished rule-based, retrieval-based, and generative approaches [3], [4]. Rule-based agents are predictable for narrow frequently asked questions but require manually maintained patterns. Generative systems can respond to varied wording and broader topics, although their outputs require appropriate prompting, source control, and user awareness.

Education is a major chatbot application domain. Okonkwo and Ade-Ibijola reviewed chatbot use across educational settings and identified benefits related to rapid service, personalization, and support [5]. Wollny et al. analyzed pedagogical roles and the use of chatbots for learning and mentoring [6]. Earlier systems included language-learning bots [7], conversational negotiation of learner models [8], and the CSIEC English-learning chatbot [9]. These studies show that conversational interfaces can support both teaching and institutional communication, but practical value depends on integration with the user's immediate task.

Modern response generation is based on transformer architectures. Self-attention allows a model to represent relationships across a sequence [10], while pretrained language models such as BERT demonstrate the value of large-scale language representation [11]. Autoregressive models can generate responses from instructions and examples [12]. The Llama 3 family extends open foundation models for multilinguality, reasoning, tool use, and long-context processing [13]. My Talkie accesses an instruction-following model through the Groq API, so the web application is responsible for constructing concise prompts and presenting the returned answer.

Document understanding is relevant when students upload notices and academic material. Tesseract provides an established OCR engine for printed text [15]. Transformer OCR and OCR-free document understanding approaches such as TrOCR and Donut illustrate later developments for visually rich documents [16], [17]. The current prototype uses Tesseract for images and pdfplumber for PDF text extraction, which is appropriate for a lightweight academic system. More complex layouts, tables, handwriting, or degraded scans could be handled by future layout-aware models.

Voice interfaces combine automatic speech recognition and text-to-speech. Connectionist temporal classification supports sequence labeling without frame-level alignment [19], while large-scale weakly supervised systems such as Whisper show the potential of robust multilingual speech recognition [18]. Text-to-speech systems convert linguistic representations into audible output [20]. In the current implementation, the browser supplies speech recognition and synthesis services, reducing backend complexity and providing immediate interaction through standard web APIs [21].

III. MATERIALS AND METHODS

The proposed method follows a client-server workflow. A student opens the browser interface and enters a query, activates the microphone, or attaches a supported file. JavaScript prepares the request and sends it to the Flask backend. The backend validates the input, extracts document text when required, combines the recovered material with the student's instruction, and sends the prompt to the Groq service. The generated reply is returned to the browser, appended to the conversation area, and optionally spoken through speech synthesis. The modules remain independent so that the AI provider, OCR engine, PDF reader, and interface can be replaced without redesigning the complete application.

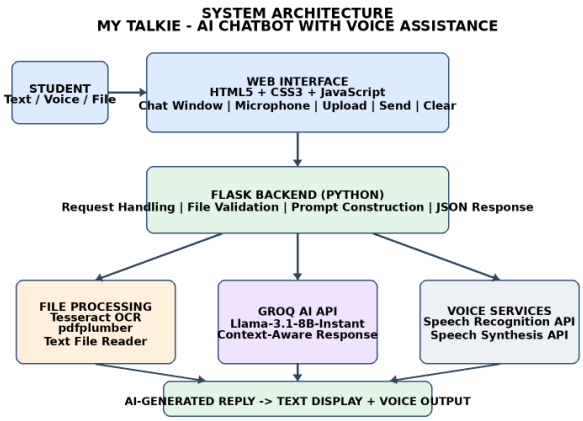


Fig. 1. System architecture reconstructed from the project document.

Figure 1 preserves the architecture specified in the report. The student interacts with an HTML/CSS/JavaScript user interface. Flask performs server-side request handling. File inputs are routed to Tesseract OCR, pdfplumber, or a text reader, while ordinary text questions proceed directly to the Groq API. The Llama-3.1-8B-Instant model generates the answer. The final response is shown as text and can be delivered as speech. This separation provides a clear path for adding authentication, data storage, ERP services, and multilingual components.

G) Prompt and Response Workflow

The implementation sequence begins with a student request and ends with synchronized text and speech output. A normal text query is forwarded directly after validation. When the microphone is used, the browser first converts speech to editable text. When a file is attached, the backend selects the extraction path according to the file type. All successful paths produce a text prompt that can be submitted through one Groq client call.

The prompt should preserve two clearly separated elements: the student instruction and the recovered document content. Delimiters prevent the uploaded text from being confused with application instructions, while a concise role description encourages the model to respond as a college-support assistant. For a long PDF, the server can split the content into manageable sections, obtain section summaries, and create a final combined explanation. This avoids truncating the beginning or end of the document without warning.

The response path is equally important. Flask returns a small JSON object containing the generated answer and any processing status needed by the interface. JavaScript appends the answer to the response panel and enables speech playback only after the text has been displayed. If extraction fails, the system should return the failed stage rather than sending an empty prompt. If AI generation fails, the extracted text can remain available so that the student can retry without uploading the document again.

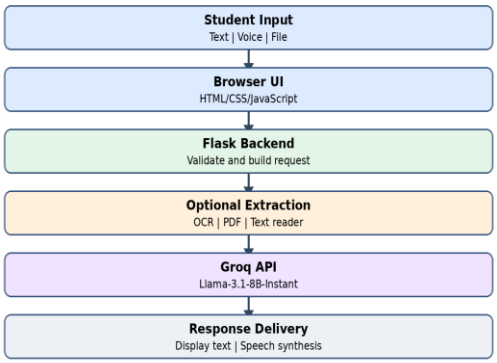


Fig. 2. End-to-end request, extraction, generation, and response workflow.

A) Dataset and Input Collection

The operational input collection used by the project consists of student questions and representative uploaded documents. Text queries cover general concepts and college-oriented information, including admissions, course availability, examinations, attendance, fees, library services, placements, and scholarships. Uploaded content includes screenshots or notice images, PDF documents, and plain-text files. These inputs exercise the application as a conversational and document-assistance system rather than as a classifier.

Each input modality enters the same response pipeline after conversion to text. A typed or recognized question is available directly. An image is opened through Pillow and processed with Tesseract OCR. A PDF is read page by page using pdfplumber, and a text file is decoded from its byte content. The recovered text is combined with an instruction such as explain, summarize, or answer the question. This unified representation allows one language-model endpoint to process several document types.

Figure 2 shows the small set of input modalities represented in the project results. The first example submits an image for analysis, the second submits a PDF, and the third submits a text file. The displayed executions illustrate the practical material supplied to the chatbot and the corresponding answer area used to present the AI response.

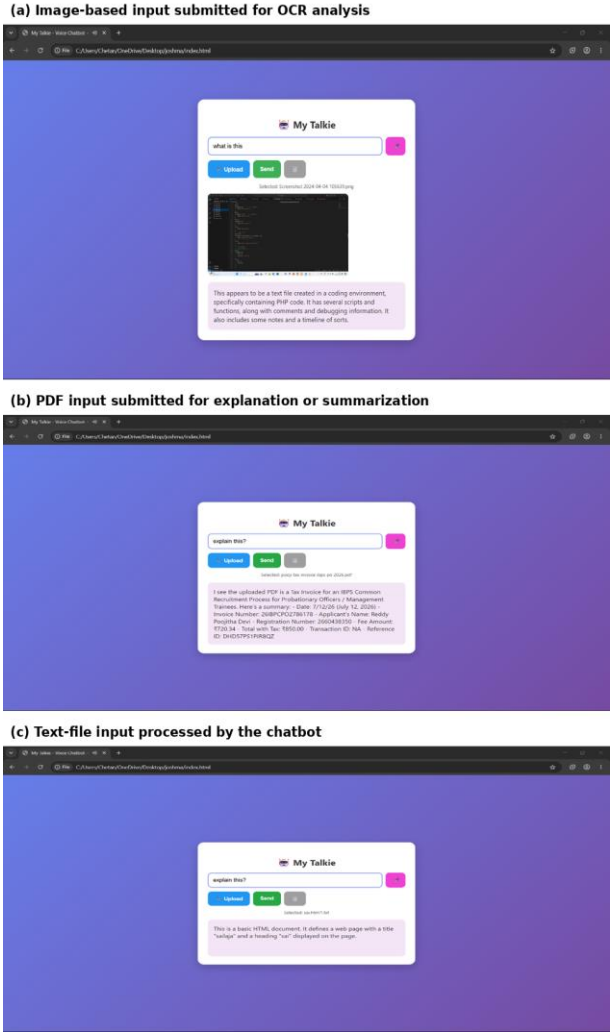


Fig. 3. Representative image, PDF, and text-file inputs from the project results.

B) User Interface and Request Construction

The interface is implemented in HTML5, CSS3, and JavaScript. It contains the My Talkie title, a single-line question field, microphone button, upload control, send button, clear button, file-selection status, optional preview, and a response panel. A gradient background and compact card layout keep the primary controls visible without requiring navigation to multiple pages. The same interface is used for ordinary questions and file-assisted requests.

JavaScript collects the message and selected file in a form request. When the user presses Send, the interface displays the request state, sends the data to the Flask route, and updates the response panel using the returned JSON. Clear removes the visible conversation state. When a file is selected, its name or preview helps the user confirm that the correct document is being processed. Input validation prevents a blank request from being submitted without a question or file.

C) Flask Backend and Groq Integration

The backend uses Python and Flask. The home route returns the chatbot page, while the ask route accepts POST requests containing a message and an optional file. Flask-CORS supports frontend-backend communication when the components are served from different origins. The application loads the Groq API key from a .env file through python-dotenv, keeping the credential outside the main source code. The Groq client then sends the prepared conversation to the selected model and returns the generated content.

The prompt combines the student's explicit question with text extracted from an uploaded document. This structure preserves the user's intent: a notice can be summarized, an assignment can be explained, or a general college question can be answered. The backend limits the response to data that can be serialized safely and returns a concise error message when the file is unsupported, content cannot be extracted, or the external service fails. A production version should also enforce file-size limits, MIME validation, request timeouts, rate limiting, and logging that excludes sensitive document content.

D) OCR, PDF, and Text-File Processing

For image input, Pillow opens the uploaded bytes and provides an image object to Tesseract. The OCR output becomes plain text that can be placed in the AI prompt. This workflow is suitable for notices, screenshots, printed assignments, and clear scanned pages. Recognition quality depends on image resolution, contrast, orientation, font size, and layout. Preprocessing steps such as grayscale conversion, thresholding, denoising, and deskewing can improve difficult inputs.

For PDF input, pdfplumber opens the in-memory document and extracts available text from each page. The page contents are concatenated before the AI request. Text-based PDFs can be processed directly, while image-only PDFs require an OCR fallback. Plain-text files are decoded and submitted without recognition. The modular implementation allows each path to return a common text representation to the response-generation stage.

E) Voice Assistance

Voice input uses the browser SpeechRecognition interface or its vendor-prefixed equivalent. After the microphone control is activated, the browser converts the student's speech to text and inserts the transcript into the message field. The student can review or edit the recognized question before sending it. This review is useful for names, course titles, and abbreviations that may be transcribed incorrectly.

Voice output uses SpeechSynthesis. After the server reply is displayed, JavaScript creates a speech utterance and requests the browser to read it aloud. Text remains visible during playback, so voice assistance complements rather than replaces the visual response. The browser-based approach avoids storing audio on the server and provides an accessible hands-free mode for students who prefer listening.

F) Security, Reliability, and Deployment

The project stores the API key in an environment file and separates interface code from backend processing. Uploaded files are processed for the current request. For deployment, HTTPS, file-type verification, safe filename handling, content-size restrictions, malware scanning, and deletion policies should be applied. Prompts created from documents must be treated as untrusted input because uploaded text may contain instructions intended to manipulate the language model. System instructions, schema checks, and content boundaries can reduce prompt-injection risk.

Reliability depends on both the local application and the hosted AI service. The backend should handle network timeouts and service errors without freezing the interface. Long documents may require truncation or chunking before submission. College-specific answers become more dependable when the chatbot is connected to an approved institutional knowledge base or ERP service instead of relying only on general model knowledge. Access control and audit trails are required before personalized attendance, marks, fee status, or student records are exposed.

IV. EXPERIMENTAL RESULTS

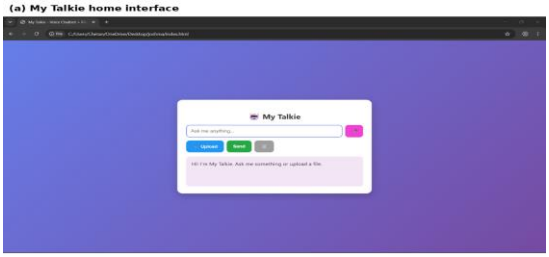
The project results demonstrate the successful execution of the main modules through browser screenshots and sample user interactions. The home page loads correctly with the My Talkie interface, which includes controls for entering a question, recording voice input through the microphone, uploading a file, sending the request, and clearing the displayed content. In the text-based interaction, a user enters a question about artificial intelligence, and the system generates a clear and concise explanation in the response area. This confirms the complete communication flow, starting from JavaScript-based input collection, continuing through the Flask backend and Groq API, and finally returning the generated response to the browser.

The file-assisted interactions further verify the system's ability to process different document formats along with user queries. After a file is selected, its name is displayed in the interface and the document is processed together with the entered request. In the image-upload example, the system extracts text using OCR and provides an appropriate explanation based on the recovered content. Similarly, the PDF example generates a response describing the information extracted from the uploaded document, while the text-file example directly reads and explains the available text. A college-related query also returns relevant information about courses, showing the usefulness of the chatbot in an institutional-support environment. In addition, voice input and voice output operate through the same interface and response-processing pipeline, providing a more accessible and interactive user experience.

Table 1. Functional Outcomes Documented in the Project

Module	Documented outcome
Text interaction	Typed college query receives an AI-generated reply.
Voice input	Browser speech recognition converts spoken input to text.
Voice output	Speech synthesis reads the generated reply aloud.
Image analysis	Tesseract OCR extracts text before AI explanation.
PDF analysis	pdfplumber extracts content for summary or explanation.
Text-file analysis	Uploaded text is read and processed by the AI model.

(a) My Talkie home interface



(b) Text query and AI-generated response

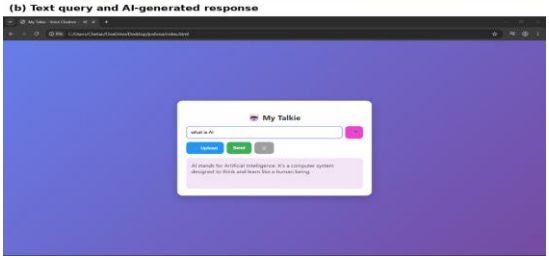


Fig. 4. Home page and text-based AI response documented in the project.

Figure 4(a) shows the initial interface, while Fig. 4(b) presents a completed text interaction. The response is displayed inside the same card, avoiding a page change and preserving a direct conversational workflow. The compact design makes the main functions available from one screen.

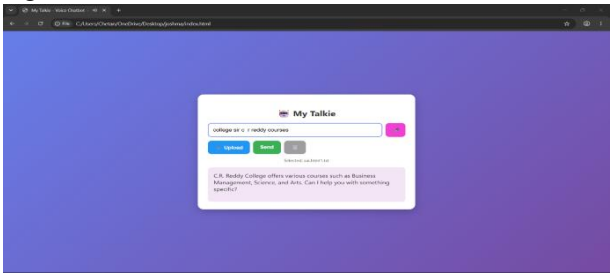


Fig. 5. College-oriented query and generated response.

Figure 5 illustrates a college-oriented question submitted through My Talkie. The displayed output demonstrates how the chatbot can be used as a first point of contact for general institutional information. Personalized or time-sensitive data should later be supplied through verified college services.

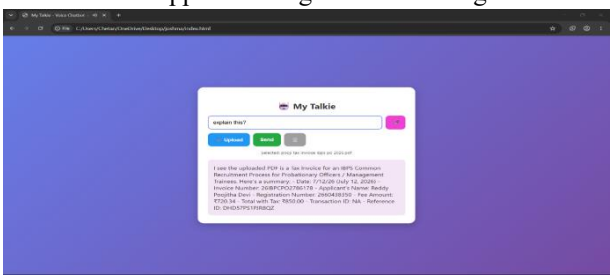


Fig. 6. PDF-assisted explanation generated through the document-processing pipeline.

Figure 6 presents the PDF-processing output recorded in the project. The selected document is read by the backend, its text is supplied to the AI service, and the generated explanation is returned in the same response panel used for ordinary questions. This keeps document assistance consistent with the normal chat workflow.

A) Discussion of Functional Outcomes

The combined outputs show that a single web interface can coordinate ordinary question answering, voice assistance, OCR, PDF reading, text-file processing, and AI response generation. The architecture avoids separate applications for each modality. A student can begin with a text question and then upload supporting material without changing the interaction model. This is useful for notices, assignments, circulars, schedules, and short study documents.

The use of a hosted language model reduces the need to construct and maintain a large set of keyword rules. However, generated answers are not automatically authoritative. College policies, dates, fees, examinations, and eligibility rules can change. An institutional deployment should retrieve such facts from approved sources and display source links or document references. The current prototype is most appropriate as a general assistance layer and document-explanation tool.

Voice support broadens interaction but depends on browser capabilities, microphone permission, environmental noise, pronunciation, and language support. OCR similarly depends on document quality. The interface should preserve transcripts and extracted text for user review before they influence the final request. Clear status indicators can distinguish speech recognition, document extraction, AI generation, and audio playback so that users understand which stage is active.

B) Applications and Future Enhancement

The system can assist prospective students with admission and course questions, enrolled students with academic schedules and services, and staff by reducing repeated general inquiries. File analysis is useful for interpreting notices, assignment descriptions, examination instructions, and text-based learning material. Voice output can support users who prefer auditory interaction or need hands-free access.

The project identifies student authentication, ERP integration, attendance checking, examination-result display, fee-status retrieval, multilingual support, mobile applications, WhatsApp integration, email notifications, chat-history storage, an administrative dashboard, cloud deployment, and advanced security as future enhancements. These features fit naturally around the existing Flask architecture. A retrieval component can index approved college documents and attach evidence to each answer, while an admin dashboard can update information and monitor unresolved query categories.

Future evaluation should focus on practical service quality. A de-identified set of representative student questions can be reviewed by college staff for correctness, completeness, relevance, and clarity. Document processing can be assessed with notices and PDFs of varied quality, while voice interaction can be checked across accents and noise conditions. Response latency, failed requests, correction frequency, and user satisfaction can be recorded during actual trials. Such measurements should be produced by executed experiments and institutional review.

C) Institutional Knowledge and Answer Governance

A college chatbot becomes more useful when it can distinguish general knowledge from institution-specific facts. General explanations, study guidance, and document summaries can be generated from the language model. Official facts should be retrieved from a controlled source maintained by the institution. The response layer can combine retrieved passages with the student question and instruct the model to answer only from the supplied evidence. This retrieval-augmented pattern reduces unsupported statements and allows the interface to show the source circular, webpage, or policy document.

Information ownership should be assigned before deployment. Admissions staff can maintain entry requirements and application dates; the examination branch can maintain schedules and regulations; the accounts section can maintain approved fee information; and the placement cell can maintain recruitment notices. Each content item should include an effective date and an expiry or review date. The chatbot should decline to invent details when the current source does not contain the requested information and should direct the student to the responsible office.

Personalized services require stronger controls than public frequently asked questions. Attendance, marks, fee status, scholarship records, and individual timetables should be returned only after authentication and authorization. The backend should not place these records in a general prompt longer than necessary. Session expiry, role checks, audit events, and minimal disclosure protect students while preserving the conversational interface. These controls can be added around the existing Flask route without changing the image, PDF, or voice modules.

An administrator dashboard can support content approval, unanswered-query review, model configuration, service status, and usage reporting. Review should focus on categories of questions and response quality rather than exposing complete student conversations. Repeated unanswered questions can indicate missing official content. Staff corrections can be added to the knowledge base and tested before publication, creating a controlled improvement cycle instead of relying on unrestricted model adaptation.

D) Limitations and Operational Considerations

The prototype depends on the availability of the Groq service and an internet connection. A network interruption, expired credential, provider-side usage limit, or model change can affect response generation even when the local Flask application is operating correctly. The interface should therefore distinguish local input errors from external-service failures and allow the student to retry without losing the entered question or selected file.

Generated responses can be fluent without being institutionally correct. Questions involving current examination dates, fee amounts, admission deadlines, attendance percentages, scholarships, or placement announcements should be answered from controlled college data. Retrieval from approved web pages, circulars, ERP records, or an administrator-maintained knowledge base can ground the model response. The answer should identify its source and the date of the underlying information whenever the topic is time-sensitive.

Uploaded documents may contain personal information, examination material, or internal notices. A production deployment should define upload retention, access control, encryption, deletion, and audit policies. The backend should reject executable content and unsupported formats, limit document size and page count, neutralize unsafe filenames, and avoid placing raw document text in application logs. Students should be informed when a file is transmitted to an external AI provider.

Voice recognition quality varies with microphone quality, background noise, accent, speaking rate, and browser implementation. OCR is affected by image blur, skew, compression, handwriting, and complex page structure. For both modalities, the recognized or extracted text should remain visible so that the student can correct it before generation. This human review stage is important for names, dates, register numbers, course codes, and other high-impact details.

The current interface demonstrates a single-turn request-response pattern. Conversation memory can improve follow-up questions, but retaining history also introduces privacy and context-management concerns. A future design should allow the user to start a new topic, delete stored conversations, and understand which earlier messages are being supplied to the model. Administrative monitoring should focus on service quality and unresolved topics rather than exposing unnecessary personal content.

V. CONCLUSION

This paper presented My Talkie, an AI chatbot with voice assistance for answering student queries and analyzing uploaded documents. The system uses an HTML/CSS/JavaScript frontend, a Python Flask backend, the Groq API with Llama-3.1-8B-Instant, Tesseract OCR, pdfplumber, and browser speech services. Its architecture supports typed questions, spoken questions, text and spoken answers, image analysis, PDF processing, and text-file processing through a unified interface.

The documented outputs show the complete interaction flow: the home page accepts input, a text query receives an AI response, uploaded image, PDF, and text content are processed, and a college-oriented question is answered in the browser. The modular design keeps file extraction, AI communication, and voice services separate, making the application understandable and extendable for academic development.

Further work can connect the chatbot to authenticated college systems, approved institutional documents, multilingual speech and text services, mobile and messaging platforms, cloud hosting, chat history, notifications, and administrative controls. With verified data sources, privacy safeguards, and structured evaluation, My Talkie can evolve into a dependable digital support channel that improves access to routine academic and administrative information.

REFERENCES

[1] A. M. Turing, “Computing machinery and intelligence,” Mind, vol. 59, no. 236, pp. 433-460, 1950.

[2] J. Weizenbaum, “ELIZA - A computer program for the study of natural language communication between man and machine,” Commun. ACM, vol. 9, no. 1, pp. 36-45, 1966.

[3] B. A. Shawar and E. Atwell, “Chatbots: Are they really useful?” LDV Forum, vol. 22, no. 1, pp. 29-49, 2007.

[4] E. Adamopoulou and L. Moussiades, “An overview of chatbot technology,” in Artificial Intelligence Applications and Innovations, IFIP AICT 584, Springer, 2020, pp. 373-383.

[5] C. W. Okonkwo and A. Ade-Ibijola, “Chatbots applications in education: A systematic review,” Computers and Education: Artificial Intelligence, vol. 2, Art. no. 100033, 2021.

[6] S. Wollny, J. Schneider, D. Di Mitri, J. Weidlich, M. Rittberger, and H. Drachsler, “Are we there yet? A systematic literature review on chatbots in education,” Frontiers in Artificial Intelligence, vol. 4, Art. no. 654924, 2021.

[7] L. K. Fryer and R. Carpenter, “Bots as language learning tools,” Language Learning & Technology, vol. 10, no. 3, pp. 8-14, 2006.

[8] A. Kerly, P. Hall, and S. Bull, “Bringing chatbots into education: Towards natural language negotiation of open learner models,” in Knowledge-Based Intelligent Information and Engineering Systems, LNCS 4692, Springer, 2007, pp. 179-192.

[9] J. Jia, “CSIEC: A computer assisted English learning chatbot based on textual knowledge and reasoning,” Knowledge-Based Systems, vol. 22, no. 4, pp. 249-255, 2009.

[10] A. Vaswani et al., “Attention is all you need,” in Advances in Neural Information Processing Systems, vol. 30, 2017, pp. 5998-6008.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. NAACL-HLT, 2019, pp. 4171-4186.

[12] T. B. Brown et al., “Language models are few-shot learners,” in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 1877-1901.

[13] A. Dubey et al., “The Llama 3 herd of models,” arXiv:2407.21783, 2024.

[14] D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. draft. Stanford, CA, USA, 2024.
[15] R. Smith, "An overview of the Tesseract OCR engine," in Proc. 9th Int. Conf. Document Analysis and Recognition, 2007, pp. 629-633.
[16] M. Li et al., "TrOCR: Transformer-based optical character recognition with pre-trained models," arXiv:2109.10282, 2021.
[17] G. Kim et al., "OCR-free document understanding transformer," in Computer Vision - ECCV 2022, Springer, 2022, pp. 498-517.
[18] A. Radford et al., "Robust speech recognition via large-scale weak supervision," in Proc. 40th Int. Conf. Machine Learning, PMLR 202, 2023, pp. 28492-28518.
[19] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in Proc. ICML, 2006, pp. 369-376.
[20] P. Taylor, Text-to-Speech Synthesis. Cambridge, U.K.: Cambridge University Press, 2009.
[21] Mozilla, "Web Speech API," MDN Web Docs, accessed Jul. 2026.
[22] Pallets Projects, "Flask documentation," accessed Jul. 2026.
[23] Python Software Foundation, "Python 3 documentation," accessed Jul. 2026.
[24] Groq, "Groq API documentation," accessed Jul. 2026.
[25] J. Vine, "pdfplumber: Plumb a PDF for detailed information," project documentation, accessed Jul. 2026.
[26] Tesseract OCR Project, "Tesseract user manual," accessed Jul. 2026.
[27] Pillow Contributors, "Pillow imaging library documentation," accessed Jul. 2026.
[28] OWASP Foundation, "OWASP API Security Top 10 - 2023," 2023.
[29] National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, 2023.
[30] M. Grinberg, Flask Web Development, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.