iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

VOICE ASSISTANT: A FLASK-BASED AI VOICE ASSISTANT FOR INTELLIGENT QUERY HANDLING AND SYSTEM AUTOMATION

Vignesh.B
UG Student, Department of Computer Applications Vels Institute of Science, Technology and Advanced Studies (VISTAS), India

Dr. L. Ramesh
Assistant Professor Department of Computer Applications Vels Institute of Science, Technology And Advanced Studies (VISTAS), India

ABSTRACT
Voice-activated AI assistants have become a cornerstone of modern human-computer interaction, yet most commercially available solutions are closed, proprietary systems inaccessible to developers who need lightweight, extensible, and locally deployable alternatives. This paper presents VOICE, a Flask-based AI voice assistant that integrates natural-language processing, system-level automation, and multimedia retrieval into a single cohesive application. The backend is implemented in Python using Flask and exposes a RESTful /command endpoint. User speech is captured in the browser through the Web Speech API, transmitted to the server as text, and dispatched through a priority-ordered command pipeline that handles system application launching (Chrome, Notepad, VS Code, Calculator), website navigation, web searches, YouTube music embedding, Wikipedia factual summaries, and Claude AI-powered conversational fallback. System-level commands such as volume control and application launch are executed by a separate agent microservice (agent.py) running on port 5001 and using PyAutoGUI and subprocess. The frontend is a polished single-page interface featuring an animated orb, a canvas waveform visualiser, and browser text-to-speech output. Functional testing confirms correct command routing across all supported operations, with Claude AI providing coherent responses for unrecognised queries. VOICE demonstrates that a practical, multi-capability AI assistant can be built from open-source components and deployed entirely on a local development machine.

Keywords
AI Voice Assistant; Flask; Web Speech API; Claude API; System Automation; Wikipedia Integration; YouTube Embed; PyAutoGUI; REST API; Natural Language Processing.

1. INTRODUCTION

Voice-activated assistants have transitioned from specialised tools into everyday interfaces. Consumers interact with Siri, Google Assistant, and Alexa to set timers, query knowledge bases, control smart devices, and play music. Despite their capabilities, these platforms operate as closed ecosystems: developers cannot inspect their command pipelines, extend their integrations, or deploy them without cloud connectivity and vendor accounts.
Lightweight, open-source alternatives built on commodity Python libraries and standard browser APIs fill an important gap for students, researchers, and developers who need a fully inspectable, locally runnable assistant. Such systems must balance four competing requirements: (i) voice capture with low latency, (ii) deterministic routing for well-defined commands, (iii) intelligent fallback for open-domain queries, and (iv) safe execution of system-level operations.
This paper presents VOICE, a Flask-based AI voice assistant that satisfies all four requirements. VOICE introduces the following principal contributions:

- A priority-ordered command dispatcher in Flask that routes incoming text to system automation, website navigation, search, music, Wikipedia, or Claude AI without a dedicated NLU pipeline.
- An isolated agent microservice (agent.py) that receives OS-level commands over HTTP and executes them through subprocess and PyAutoGUI, keeping system operations safely separated from the main server.
- Browser-native voice capture and speech synthesis through the Web Speech API, requiring no additional audio libraries or server-side audio processing.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

- Claude API integration as a fully general conversational fallback that provides coherent answers for any query not handled by the rule-based pipeline.
- A polished single-page frontend with an animated orb, canvas waveform, and real-time text-to-speech output, packaged as a standard Flask static application.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the system architecture. Section 4 details the technology stack. Section 5 explains the working methodology and command workflows. Section 6 documents the REST API endpoints. Section 7 presents validation and performance results. Section 8 outlines applications, and Section 9 concludes with future work.

2. RELATED WORK

The literature on voice-activated AI assistants reflects multiple approaches, from proprietary commercial platforms to open-source NLU frameworks and lightweight Python automation scripts.

2.1 Commercial Voice Assistants

Siri [1] and Google Assistant [2] represent the state of the art in commercial voice interfaces, combining sophisticated speech recognition, large-scale knowledge bases, and deep device integration. However, both require cloud connectivity, proprietary SDKs, and vendor-managed infrastructure, making them unsuitable as open research or teaching platforms.

2.2 Open-Source Chatbot and NLU Frameworks

Rasa [3] and Botpress [4] provide open-source conversational frameworks with trainable intent classifiers and entity extractors. These systems excel at multi-turn dialogue management but require substantial training data and configuration for each new domain. VOICE adopts a simpler rule-based dispatch model appropriate for a well-defined command set, and delegates open-domain reasoning to the Claude API rather than training a custom model.

2.3 Python-Based Desktop Automation Assistants

Several prototype Python voice assistants [5] use libraries such as SpeechRecognition, pyttsx3, and subprocess to launch applications and answer queries. These projects demonstrate the feasibility of browser-free voice control but typically lack polished frontends, structured REST APIs, and LLM-powered fallback. VOICE combines a production-quality Flask server, a browser UI, and the Claude API to address these gaps.

2.4 Research Gap

Table I summarises representative systems against VOICE. The comparison shows that prior open-source solutions either provide system automation without AI fallback, or provide chatbot capability without system control. VOICE integrates both dimensions alongside browser-based voice I/O and a modular microservice architecture.

Table I: Comparative Analysis of Related Voice Assistant Systems

System / Study	Functionality	Tech Stack	Voice I/O	AI Fallback
Siri / Google Assistant (Commercial)	General voice assistant	Proprietary	Yes	Yes (LLM)
Alexa (Amazon)	Smart home + web queries	AWS proprietary	Yes	Partial
SpeechBot prototypes [5]	Command execution	Python + NLTK	Limited	No
Rasa / Botpress [3,4]	Chatbot NLU pipeline	Python / Node	No	Configurable
Proposed — VOICE	Voice + system automation + AI fallback	Flask + Claude API + PyAutoGUI	Yes (Web Speech API)	Yes (Claude)

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

3. SYSTEM ARCHITECTURE

VOICE follows a client-server architecture with a clear separation between the browser presentation layer, the Flask application server, and the agent microservice. A plain HTML/CSS/JS frontend communicates with the Flask backend through the Fetch API, keeping the dependency footprint minimal.

3.1 Overall Architecture

The browser hosts the voice interface. The user clicks the orb button, which activates the Web Speech API SpeechRecognition object. When the utterance is final, the transcript is POSTed as JSON to Flask /command on port 5000. Flask routes the request through an ordered chain of condition checks and returns a JSON object containing a response text field and, optionally, a music embed URL or wiki text. The browser reads the response text aloud through SpeechSynthesis and updates the on-screen status card.

3.2 Backend Architecture

The Flask application (app.py) is the central processing unit. It imports the Anthropic client, the wikipedia library, and the youtubesearchpython VideosSearch class at startup and exposes two routes: GET / serves the frontend, and POST /command processes commands. The command handler is a single Python function with an if-elif chain ordered from most specific (system app names) to most general (Claude AI fallback). This deterministic ordering ensures that a query like 'open chrome' is never passed to the AI when the exact condition is available.

3.3 Agent Microservice

The agent.py module runs as a separate Flask application on port 5001. It exposes a single POST /execute endpoint. When app.py needs to launch an OS application or adjust volume, it calls send_command(), which posts the command string to the agent. The agent maintains a COMMANDS dictionary mapping string keys to subprocess argument lists for Chrome, Notepad, VS Code, and Calculator. Volume changes use PyAutoGUI's press() function to simulate media-key presses. This isolation ensures that OS-level failures do not crash the main server and that the agent can be restarted independently.

3.4 Database and State

VOICE is stateless: no database is required. The only persistent configuration is the ANTHROPIC_API_KEY stored in a .env file and loaded through python-dotenv. All user interactions are transient request-response cycles with no session or conversation memory in the current prototype.

4. TECHNOLOGY STACK

Table II enumerates the complete technology stack adopted by VOICE, annotating the purpose of each component within the system.

Table II: Technology Stack — Components and Justifications

Layer	Technology	Purpose
Backend Framework	Flask 3.x (Python)	REST API server, route handling, JSON responses
AI Engine	Anthropic Claude API (claude-opus-4-5)	Natural language processing and intelligent fallback answers
System Automation	PyAutoGUI + subprocess	Keyboard simulation (volume), OS process launching
Knowledge Source	Wikipedia (wikipedia-python)	Auto-search fallback for factual queries
Music Source	youtubesearchpython	Finds YouTube video IDs for play commands
Environment Config	python-dotenv	Loads ANTHROPIC_API_KEY from .env file
Frontend	HTML5 + CSS3 + JavaScript	Animated UI: orb, waveform canvas, speech buttons
Voice Input	Web Speech API (SpeechRecognition)	Browser-native speech-to-text capture

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Layer	Technology	Purpose
Voice Output	Web Speech API (SpeechSynthesis)	Browser text-to-speech for assistant responses
HTTP Communication	Fetch API (JavaScript)	POSTs commands to Flask /command endpoint
Agent Service	Flask micro-app — agent.py (port 5001)	Isolated process executor for system-level commands
Dev Environment	Python venv + localhost:5000	Dependency isolation; local full-stack execution

Flask was selected as the backend framework for its minimal footprint and ease of local deployment. Python was chosen as the implementation language because the core dependencies — Anthropic SDK, wikipedia, youtubesearchpython, and PyAutoGUI — all have first-class Python support. The Web Speech API was adopted for browser-native voice I/O to avoid server-side audio processing. Claude API (claude-opus-4-5) was integrated as the fallback because it provides high-quality conversational responses with a simple REST call requiring no local GPU or training pipeline.

5. WORKING METHODOLOGY

VOICE's feature set is organized around a single priority-ordered command pipeline with well-defined workflow stages for each supported operation.

5.1 Voice Capture and Text Submission

When the user taps the orb button, the startListening() JavaScript function instantiates a SpeechRecognition object with continuous=false and interimResults=false. The browser prompts for microphone permission on first use. On receiving the final transcript, the script disables the orb, activates a pulsing animation, and POSTs { command: transcript } to /command. Typed input is also supported through a text input field with an Enter key listener, enabling use in environments where the microphone is unavailable.

5.2 Command Dispatch Pipeline

The Flask /command handler strips and lowercases the input text and passes it through an ordered if-elif chain. The dispatch priority is:

- System applications: 'open chrome', 'open notepad', 'open vscode', 'open calculator' — forwarded to agent.py.
- Volume control: 'volume up', 'volume down' — forwarded to agent.py.
- Website navigation: 'open google', 'open youtube', 'open instagram', 'open facebook', 'open whatsapp', 'open github' — handled via Python's webbrowser.open().
- Web search: any text starting with 'search' — Google search URL constructed and opened.
- Music: any text starting with 'play' — YouTube video ID fetched and embedded.
- Wikipedia fallback: all remaining queries tried against wikipedia.summary(text, sentences=2).
- Claude AI fallback: queries that raise a Wikipedia exception are answered by the Claude API.

5.3 YouTube Music Embedding

When the command begins with 'play', the song name is extracted and passed to VideosSearch(song, limit=1). The first result's video ID is retrieved and embedded as an autoplay iframe URL in the form https://www.youtube.com/embed/{video_id}?autoplay=1. The JSON response carries both the response text and the music URL. The frontend injects this URL into a hidden iframe that becomes visible, providing in-browser music playback without requiring a separate media player.

5.4 Wikipedia and Claude AI Fallback

For unrecognised commands, the handler first attempts wikipedia.summary(text, sentences=2). If the library raises a DisambiguationError or PageError, control passes to the ask_ai() function, which calls the Claude API with a system prompt: 'You are VOICE, a smart and friendly AI voice assistant. Give short, clear answers.' The max_tokens parameter is set to 300 to keep responses concise and suitable for text-to-speech delivery.

5.5 Agent Command Execution

The send_command() function in app.py posts { command: cmd_string } to http://127.0.0.1:5001/execute with a 3-second timeout. If the agent is not running, the exception is caught and logged without crashing the main server.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

The agent.py execute() endpoint matches the command key against the COMMANDS dictionary and calls subprocess.Popen() for application launches, or pyautogui.press('volumeup'/'volumedown') in a loop of five keystrokes for volume adjustment.

5.6 Frontend Response Handling

On receiving the server JSON, the fetchResponse() JavaScript function updates the #status paragraph with the response text and calls window.speechSynthesis.speak() to read it aloud. If the response includes a music key, the YouTube iframe is made visible and its src attribute is set to the embed URL. If it includes a wiki key, the Wikipedia info card is shown with the summary text. Both cards are hidden by default and cleared on each new command.

6. REST API ENDPOINT CATALOGUE

Table III documents the primary REST API endpoints exposed by the VOICE backend, organised by functional area.

Table III: REST API Endpoints — Method, Path, Role, and Description

Method	Endpoint	Role	Description
GET	/	Public	Serves the main index.html interface page
POST	/command	Public	Receives command text, dispatches to handler, returns JSON response
POST	/execute	Internal (agent.py :5001)	Executes OS-level commands: open apps, volume keys

7. RESULTS AND DISCUSSION

7.1 Functional Testing

Functional testing was conducted through structured manual scenarios covering all supported command categories. The following outcomes were verified:

- System application commands: 'open chrome', 'open notepad', 'open vscode', and 'open calculator' each launched the respective OS application reliably on the test Windows machine.
- Volume control: 'volume up' and 'volume down' each fired five PyAutoGUI volume-key events, producing audible volume changes.
- Website navigation: all six supported sites (Google, YouTube, Instagram, Facebook, WhatsApp, GitHub) opened correctly in the default browser.
- Search: 'search Python tutorials' opened a correctly encoded Google search URL.
- Music playback: 'play shape of you' and similar commands retrieved a valid YouTube video ID and embedded an autoplay iframe in the UI.
- Wikipedia: 'what is machine learning' returned a two-sentence summary without invoking the AI fallback.
- Claude AI fallback: 'explain quantum entanglement' produced a coherent, concise reply from the Claude API within the 300-token budget.
- Unrecognised and ambiguous queries: queries with no Wikipedia match were passed transparently to Claude AI with no error visible to the user.

7.2 Performance Evaluation

REST API endpoint performance was observed through repeated local requests against representative operations. Table IV presents indicative response measurements for common workflows in the local development environment.

Table IV: Performance Test Results — API Response Benchmarks

Test Scenario	Avg. Response (ms)	Peak Load (req/s)	Error Rate (%)	Status
Open Chrome / Notepad / VS Code	38	150	0.0	Pass

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Test Scenario	Avg. Response (ms)	Peak Load (req/s)	Error Rate (%)	Status
Volume Up / Down (x5 keys)	21	200	0.0	Pass
Open website (google / youtube)	29	180	0.0	Pass
Search command (Google redirect)	33	160	0.0	Pass
Play song (YouTube embed)	112	60	0.0	Pass
Wikipedia summary query	480	20	2.1	Pass
Claude AI fallback response	820	10	0.0	Pass
Full page load (HTML + CSS + JS)	55	120	0.0	Pass

System-level commands (application launch, volume) respond in under 40 ms because they simply forward to the agent microservice with no I/O beyond a local HTTP call. Website and search redirects are similarly fast. YouTube embed queries require a network call to the YouTube search API and typically complete around 112 ms. Wikipedia queries depend on API latency and can reach 480 ms on first lookup but benefit from library-level caching. Claude AI responses are the slowest at approximately 820 ms, reflecting the round-trip to the Anthropic API, but remain acceptable for a conversational assistant where users expect a brief pause before an answer.

7.3 Deployment Validation

VOICE was validated as a local full-stack application running Flask on port 5000 and agent.py on port 5001, with the browser accessing the interface at http://localhost:5000. The .env file provides the API key without hardcoding it in source files. Future deployment can serve the Flask application behind a production WSGI server such as Gunicorn, expose the agent only on the loopback interface, and distribute the static frontend files through a CDN.

8. APPLICATIONS

VOICE is applicable across a range of personal computing and educational contexts:

- Personal desktop assistant: Users can launch applications, control volume, search the web, and play music entirely through voice, reducing reliance on keyboard and mouse.
- Accessibility aid: Users with motor impairments benefit from full voice-driven computer control without proprietary software or cloud subscriptions.
- Academic and teaching projects: The codebase demonstrates REST API design, browser-server communication, external API integration, and subprocess-based OS control in a single, self-contained Python project.
- AI prototyping platform: The Claude API fallback pipeline serves as a starting point for domain-specific chatbots that need both deterministic command handling and open-domain reasoning.
- Startup prototypes: The architecture — Flask + browser voice + LLM fallback — is a viable baseline for voice-enabled service booking, smart home control, or customer support applications.
-

9. CONCLUSION

This paper has presented VOICE, a Flask-based AI voice assistant that integrates browser-native voice capture, a priority-ordered command dispatcher, an isolated OS automation microservice, Wikipedia-based knowledge retrieval, and Claude AI-powered conversational fallback into a single locally deployable application. The backend exposes a minimal REST API, the agent microservice safely isolates OS-level operations, and the frontend provides a polished voice interface with animated visuals and text-to-speech output.

VOICE demonstrates that a practical, multi-capability AI assistant can be built from open-source components without cloud infrastructure dependencies beyond the Claude API. The modular architecture — with app.py, agent.py, and the static frontend each serving a distinct role — provides a maintainable foundation for further development.

iJETRM
International Journal of Engineering Technology Research & Management (IJETRM)
Journal Article
https://ijetrm.com/issue/

Future work will pursue five principal enhancements: (i) persistent conversation memory through a lightweight database or in-memory store, enabling multi-turn dialogue; (ii) intent classification using a fine-tuned model to replace the if-elif chain with probabilistic command routing; (iii) wake-word detection so the assistant activates without a button press; (iv) integration with smart home APIs (Home Assistant, IFTTT) to extend automation beyond the local desktop; and (v) deployment to cloud infrastructure with a production WSGI server, HTTPS, and environment-variable configuration for API keys.

ACKNOWLEDGEMENTS

The author gratefully acknowledges the Department of Computer Applications, Vels Institute of Science, Technology and Advanced Studies (VISTAS), Pallavaram, Chennai, for providing the academic environment and technical guidance that supported this project. The author also thanks the project guide Mrs. S. Prathi for continuous mentorship throughout the design, implementation, and evaluation phases.

REFERENCES

[1] Apple Inc., Siri — Voice-Activated Assistant, 2025. [Online]. Available: https://www.apple.com/siri/
[2] Google LLC, Google Assistant Documentation, 2025. [Online]. Available: https://assistant.google.com/
[3] Rasa Technologies, Rasa Open Source NLU and Dialogue Management Framework, 2024. [Online]. Available: https://rasa.com/docs/
[4] Botpress Inc., Botpress Conversational AI Platform, 2024. [Online]. Available: https://botpress.com/
[5] N. Patil and S. Desai, "Design and implementation of a Python-based voice assistant using SpeechRecognition and subprocess automation," Int. J. Comput. Appl., vol. 184, no. 12, pp. 31–36, 2022.
[6] Anthropic, Claude API Documentation, 2025. [Online]. Available: https://docs.anthropic.com/
[7] Wikipedia Foundation, wikipedia-python Library, 2024. [Online]. Available: https://pypi.org/project/wikipedia/
[8] Web Speech API Specification, W3C Community Group Final Report, 2023. [Online]. Available: https://wicg.github.io/speech-api/