IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 7, October 2026
Impact Factor: 8.2

AI Based Voice Command Assistant Using Python

Nawale Abhijit Bhausaheb[1], Kapse Kartik Balu[2], Nat Sandesh Raju[3], Nat Adesh Raju[4], Prof. V. N. Dhakane[5], Dr. H. B. Jadhav[6], Prof. G. R. Sarode[7]
Student, Department of Computer Engineering[1234]
Faculty, Department of Computer Engineering[567]
Adsul's Technical Campus, Chas, Ahilyanagar, Maharashtra, India

Abstract: *Voice-driven interaction is rapidly becoming a natural alternative to the keyboard and mouse, yet popular assistants such as Siri, Alexa and Google Assistant remain closed-source and bound to specific hardware ecosystems. This paper presents the design and implementation of an AI based voice command assistant built entirely in Python. The system captures live audio through a microphone, converts speech to text using the SpeechRecognition library, extracts user intent with lightweight natural language processing (NLTK/spaCy), executes the matched action such as web search, application launch or reminder setting, and finally responds to the user through the offline text-to-speech engine pyttsx3. The assistant is organised as a linear five-stage pipeline that is easy to understand, extend and run on modest hardware. The paper describes the architecture, technology stack, command-processing flow, applications, advantages and limitations of the system, and outlines future work including multilingual (Hindi and Marathi) support, offline recognition using Vosk, IoT integration and conversational memory.*

Keywords: Voice Assistant, Speech Recognition, Text-to-Speech, Natural Language Processing, Python, Human–Computer Interaction, pyttsx3

I. INTRODUCTION

Human–computer interaction has traditionally relied on the keyboard, mouse and touch screen. As computing moves into cars, homes and wearable devices, these input methods become inconvenient, and speech has emerged as the most natural interface for everyday tasks [1]. A voice command assistant is a software application that listens to spoken input, converts it into text, interprets the intent behind it and performs an action or gives a spoken reply, all without the user touching a keyboard or mouse. Commercial assistants such as Siri, Alexa, Google Assistant and Cortana have made this idea mainstream, but they are proprietary and tied to specific platforms [2].

This paper presents an AI based voice command assistant implemented from scratch in Python. The system combines three well-established technologies, namely speech recognition, natural language processing (NLP) and speech synthesis, into a single lightweight desktop assistant. It is inspired by commercial products but is fully open and customisable, so that students, developers and researchers can study, extend and adapt it to their own needs.

The remainder of the paper is organised as follows. Section II reviews related work. Section III states the problem and objectives. Section IV describes the system architecture and methodology. Section V lists the technology stack, Section VI presents the supported features, and Section VII discusses applications. Sections VIII and IX discuss advantages, limitations and future scope, and Section X concludes the paper.

II. LITERATURE REVIEW

Automatic speech recognition (ASR) has progressed from Gaussian-mixture hidden Markov models to deep neural acoustic models, which delivered large accuracy gains in the last decade [3]. Open toolkits such as Kaldi made state-of-the-art research systems available to the community [4], while end-to-end approaches such as Deep Speech [5] and,

Copyright to IJARSCT
www.ijarsct.co.in
DOI: 10.48175/IJARSCT-40483

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 7, October 2026
Impact Factor: 8.2

more recently, large-scale weakly supervised models such as Whisper [6] have further simplified the recognition pipeline and improved robustness to accents and noise.

For application developers, high-level Python wrappers have lowered the entry barrier. The SpeechRecognition package offers a unified interface to several recognition engines and web APIs [7], and Vosk provides compact offline models that run on modest hardware [8]. On the output side, pyttsx3 is a cross-platform, offline text-to-speech library that wraps native engines such as SAPI5, NSSpeechSynthesizer and eSpeak [9]. For language understanding, NLTK [10] and spaCy [11] supply tokenisation, part-of-speech tagging and entity recognition that can be used to extract keywords and intent from short commands.

Comparative studies of commercial assistants show that they differ noticeably in recognition quality and supported functions, and that all of them depend on cloud services [2]. In addition, studies on commercial ASR systems report unequal error rates across speaker groups, which highlights the need for transparent and adaptable components [12]. These observations motivate an open, modular and easily extensible assistant such as the one proposed here.

III. PROBLEM STATEMENT AND OBJECTIVES

A. Problem Statement

Most computing tasks still require typing or clicking, which slows down users who are multitasking, driving or have limited mobility. People with visual or motor impairments often struggle with traditional keyboard-and-mouse interfaces, creating an accessibility gap. Finally, popular assistants are closed-source and tied to particular hardware or platforms, which limits customisation and learning opportunities.

B. Objectives

- To build a working speech-to-text pipeline using Python's SpeechRecognition library.
- To interpret user intent from natural spoken commands using simple NLP techniques.
- To execute real actions such as web search, application launch and reminders based on recognised commands.
- To respond to the user with natural-sounding speech through text-to-speech synthesis.

IV. SYSTEM ARCHITECTURE AND METHODOLOGY

A. Overall Architecture

The proposed assistant follows a linear pipeline in which each stage hands its output to the next, from raw audio to a spoken reply, as shown in Fig. 1. The five stages are voice input, speech-to-text conversion, NLP processing, command execution and voice output. This modular design allows any stage to be replaced, for example swapping the recognition engine, without affecting the others.

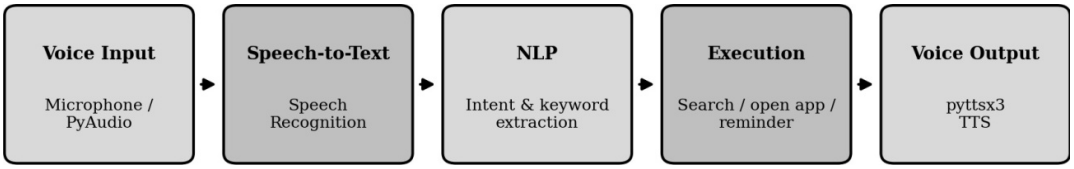


Fig. 1. Architecture of the AI based voice command assistant

B. Process Flow

The operation of the assistant can be summarised in five steps.

- Listen: the microphone stays active and listens for a wake word or a command. Live audio is captured through PyAudio.
- Recognise: the captured audio is streamed to the recogniser and converted into text.
- Interpret: keywords in the recognised text are matched against known command intents using lightweight NLP.
- Execute: the function mapped to the matched intent runs, for example opening an application or searching the web.
- Respond: pyttsx3 converts the result into speech and plays it back to the user.

Copyright to IJARSCT
www.ijarsct.co.in
DOI: 10.48175/IJARSCT-40483

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 7, October 2026
Impact Factor: 8.2

C. Voice Input and Speech-to-Text

Audio is acquired from the default microphone using PyAudio. Ambient noise is sampled briefly so that the recogniser can adjust its energy threshold, after which the utterance is passed to the SpeechRecognition module, which returns the transcribed text. If the speech cannot be understood or the service is unreachable, the assistant asks the user to repeat the command instead of failing silently.

D. NLP Processing and Intent Matching

The transcribed text is converted to lower case, tokenised and cleaned of stop words using NLTK or spaCy. Keywords and phrases are then compared with a table of predefined intents such as search, open, remind and question. The matched intent, together with any extracted argument (for example the search term or application name), is forwarded to the command executor. Because the approach is rule and keyword based, new commands can be added by simply registering a new keyword set and its handler function.

E. Command Execution and Voice Output

Each intent is bound to a Python function. Web searches and website requests are handled through the standard webbrowser module, while applications such as Notepad, Calculator and the browser are launched through the os module. Reminders and alarms are scheduled using the system clock. The textual result of every action is finally passed to pyttsx3, which speaks it aloud so that the user receives immediate audible feedback.

V. TECHNOLOGY STACK

The assistant is implemented with freely available Python libraries, summarised in Table I.

TABLE I: TECHNOLOGIES USED IN THE PROPOSED SYSTEM

Component	Technology	Purpose
Core language	Python 3	Main programming language
Speech-to-text	SpeechRecognition	Converts speech to text
Text-to-speech	pyttsx3	Offline speech synthesis
Audio capture	PyAudio	Live microphone input
NLP	NLTK / spaCy	Lightweight intent parsing
System actions	webbrowser / os	Executes system and web actions

VI. FEATURES AND SUPPORTED COMMANDS

The assistant currently provides the capabilities listed in Table II. Each feature is exposed as a spoken command and every response is also delivered by voice.

TABLE II: KEY FEATURES AND EXAMPLE VOICE COMMANDS

Feature	Description	Example command
Voice-to-text	Real-time transcription of spoken commands	Any spoken command
Text-to-speech	Spoken replies for every response	Assistant reply
Web search	Opens browser searches on request	“Search Python tutorials”
App control	Launches Notepad, Calculator, browser	“Open calculator”
Reminders and alarms	Sets time-based voice reminders	“Remind me at 5 PM”
Q&A lookup	Answers general questions by quick lookup	“Who is the Prime Minister of India”

VII. APPLICATIONS

- Smart home control: voice-controlled lights, appliances and IoT devices.
- Accessibility tools: hands-free computing for users with visual or motor impairments.
- Personal productivity: reminders, quick searches and task automation.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 7, October 2026
Impact Factor: 8.2

• Customer service: voice-driven FAQ systems and information kiosks.

VIII. ADVANTAGES AND LIMITATIONS

A. Advantages

• Free and fully open source, with no vendor lock-in.
• Text-to-speech runs offline through pyttsx3.
• Easy to extend with new voice commands.
• Lightweight, and runs on modest hardware.

B. Limitations

- Speech recognition through web engines needs an internet connection.
- Accuracy drops with background noise and strong accents.
- Limited to predefined command patterns.
- No true conversational memory between commands.
- Quantitative evaluation of recognition accuracy (word error rate), response latency and intent-matching success under different noise conditions and speakers is planned as part of further work.

C. Comparison with Commercial Assistants

Table III summarises, in qualitative terms, how the proposed assistant differs from widely used commercial assistants. The comparison is based on the publicly documented design of those products and on published comparative studies [2], [13], not on measured benchmarks, and it is intended only to clarify the design space in which this work sits.

TABLE III: COMPARISON OF THE PROPOSED ASSISTANT WITH COMMERCIAL ASSISTANTS

Aspect	Commercial assistants	Proposed assistant
Source code	Closed-source	Open and fully modifiable
Speech processing	Largely cloud-based	Web engine today; offline mode planned
Platform	Tied to vendor ecosystems	Any desktop that runs Python 3
Customisation	Limited to vendor-approved skills	Direct access to intents and handlers
Cost	Device or subscription may be needed	Built on free libraries
Hardware	Phones, speakers, dedicated devices	Ordinary laptop and microphone

The principal trade-off is breadth versus transparency. Commercial assistants draw on very large cloud-hosted language and knowledge services and therefore handle open-ended requests far better than a keyword-driven system can. The proposed assistant gives up that breadth in exchange for full control over every stage of the pipeline, which is valuable for teaching, experimentation and for deployments in which data must not leave the machine.

D. Command Processing Walk-through

To make the processing flow concrete, Table IV traces four representative commands through the five stages of the pipeline. The traces illustrate the design of the system and are not measured results.

TABLE IV: EXAMPLE COMMAND TRACES THROUGH THE PIPELINE

Spoken command	Intent and argument	Action	Spoken reply
Search Python tutorials	search; “python tutorials”	Opens a browser search	“Searching for Python tutorials”
Open calculator	open; “calculator”	Launches the calculator	“Opening calculator”
Remind me at 5 PM	remind; time 17:00	Schedules a timed reminder	“Reminder set for 5 PM”
Who is the Prime Minister of India	question; whole query	Quick lookup of the answer	Answer read aloud

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 7, October 2026
Impact Factor: 8.2

In every case the same sequence applies. Time expressions are extracted first so that "5 PM" is not lost when stop words are removed, the remaining keywords select the intent, and whatever is left after the intent keyword is taken out becomes its argument. If no intent matches, the assistant answers that it did not understand and returns to listening.

E. Implementation Challenges and Design Considerations

Several practical issues shaped the design of the assistant, and they are worth recording because they are common to most small voice interfaces.

- Ambient noise: a short calibration of about one second before each listening session allows the recogniser to set its energy threshold relative to the room, which reduces false triggers from fans, traffic or conversation in the background.
- Latency: the delay a user perceives is the sum of audio capture, recognition, intent matching, action execution and speech synthesis. When a web recognition engine is used, the network round trip is expected to dominate this total, which is a further argument for the offline recogniser discussed in Section IX.
- Blocking speech output: the pyttsx3 engine blocks the calling thread while it speaks. Running synthesis in a separate thread keeps the listener responsive and makes it possible to stop a long reply when the user issues a new command.
- Error handling: the SpeechRecognition library signals an unintelligible utterance and an unreachable service with different exceptions. Treating them separately lets the assistant say "please repeat" in the first case and report a connectivity problem in the second, instead of failing silently.
- Platform differences: pyttsx3 relies on SAPI5, NSSpeechSynthesizer or eSpeak depending on the operating system, and application names and paths differ as well. A small per-platform table of launch commands keeps the intent handlers themselves platform independent.
- Intent ambiguity: commands such as "open YouTube" and "search YouTube" share keywords. Giving each intent a priority and preferring the longest matching keyword phrase resolves most conflicts without any statistical model.

F. Extensibility and Maintainability

Because each intent is bound to an ordinary Python function, a new capability is added by writing one handler and registering it together with its keyword set, for example a handler that reports the time or tells a joke. A dictionary that maps intent names to handlers keeps this registration in one place and avoids long chains of conditional statements.

The same separation makes the assistant easy to test. Handlers can be exercised with typed text instead of live audio, recorded transcripts can be replayed to check the intent matcher, and the recogniser and synthesiser can be replaced by simple stubs in automated tests. Logging every recognised phrase, matched intent and response time also supplies the raw data needed for the evaluation described later in this section.

G. Accessibility Considerations

Hands-free operation is the most direct accessibility benefit, since users with motor impairments can launch applications and search the web without a keyboard or pointing device. Spoken replies help users with low vision but exclude users who are deaf or hard of hearing, so the recognised text and the reply should also be shown on screen. Users with speech impairments or strong accents are more likely to be misrecognised [12], which argues for tolerant keyword matching, easy repetition of a command and an alternative text-input path.

H. Security and Privacy Considerations

A voice assistant listens continuously and can act on the machine, so privacy and safety deserve explicit attention. When a web recogniser is used, the captured audio leaves the device and is processed by a third-party service; users should therefore be told clearly when the microphone is active and which engine receives the audio. Local recognition

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 7, October 2026
Impact Factor: 8.2

and local synthesis remove this exposure, and embedded private-by-design voice platforms have been proposed for the same reason [14].

Because command execution can launch programs and change settings, the set of callable handlers should be an explicit allow-list, and the recognised text should never be passed to a shell or evaluated as code. Sensitive actions, for example deleting files or controlling physical devices, should ask for a spoken confirmation before they run. Finally, the unequal error rates reported across speaker groups [12] mean that an assistant which works well for one accent may frustrate another, so evaluation should include a diverse set of speakers.

I. Planned Evaluation Methodology

The evaluation planned for the next stage of the work is summarised in Table V. Each metric will be measured in three acoustic conditions, namely a quiet room, moderate background speech and a noisy environment, using a fixed list of commands covering every feature in Table II and several speakers with different accents. Standard resources such as the Speech Commands dataset [15] can supplement live recordings for the recognition tests.

TABLE V: METRICS PLANNED FOR EVALUATING THE ASSISTANT

Metric	Definition	How it is measured
Word error rate	(S + D + I) / N over reference words	Transcripts versus reference text
Intent accuracy	Correct intents / total commands	Labelled command list
Task success rate	Commands completed as requested	Observed action per command
Response latency	End of speech to start of reply	Timestamps in the program log
Usability	Perceived ease of use	System Usability Scale [16]

Here S, D and I are the numbers of substituted, deleted and inserted words and N is the number of words in the reference transcript. Reporting these figures for each noise condition will show where the current pipeline is weakest and will give a baseline against which the offline and multilingual extensions can be compared.

IX. FUTURE SCOPE

- Multilingual support: recognise and respond in multiple languages, including Hindi and Marathi.
- Offline recognition: integrate offline models such as Vosk to remove the internet dependency [8].
- IoT integration: connect to smart-home APIs for real device control.
- Conversational memory: add context tracking so that follow-up commands feel natural.

These directions are developed below. Fig. 2 shows how the planned components would extend the present five-stage pipeline while keeping its modular, stage-by-stage structure.

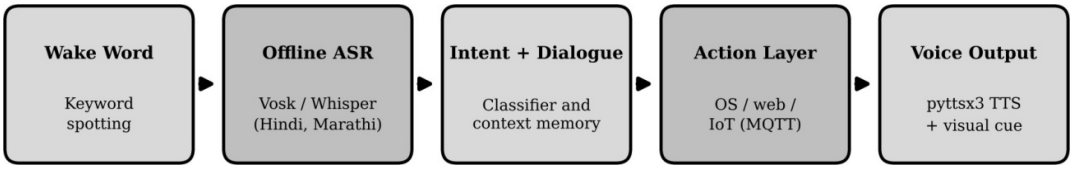


Fig. 2. Proposed extended architecture with wake word, offline recognition and dialogue memory

A. Multilingual Support

Many users in India speak Hindi or Marathi at home and often mix languages within a single sentence. Supporting them requires recognition models for those languages, intent keywords stored in Devanagari script as well as in Latin transliteration, and text-to-speech voices that can pronounce them. Web engines accept language codes such as hi-IN, while large multilingual models such as Whisper cover both languages [6]. The availability of local synthesis voices depends on the voices installed in the operating system, so this part of the work must be verified on each target platform.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 7, October 2026
Impact Factor: 8.2

B. Offline Recognition

Replacing the web recogniser with a local model removes the internet dependency, makes latency predictable and keeps audio on the device. Vosk offers compact models that run on modest hardware [8], whereas larger end-to-end models [5], [18] tend to be more accurate but need more memory and processing time. A hybrid design could use the small local model by default and fall back to a cloud engine only when confidence is low and the user has agreed to it.

C. IoT Integration

Smart-home control can be added by mapping intents to messages for a lightweight protocol such as MQTT or to the REST interface of a home-automation hub. A command like "turn on the light" would then publish a message to a topic that a Raspberry Pi or ESP32 relay board subscribes to. Because these actions affect the physical world, each device handler should confirm the target device and ask for confirmation when a command is ambiguous.

D. Conversational Memory and Learned Intent Models

Context tracking would store the last recognised intent and its arguments so that follow-ups such as "open it again" or "what about tomorrow" can be resolved. The keyword table could also be replaced or supplemented by a trained intent classifier based on a pre-trained transformer such as BERT [17], which tolerates paraphrases that a fixed keyword list would miss. Keeping the rule-based matcher as a fallback preserves the transparency that motivates this work.

E. Wake-Word Detection and Personalisation

At present the microphone listens for any command. A small always-on keyword-spotting model would keep the heavier recogniser idle until the wake word is heard, which saves power and reduces the amount of audio that is ever processed [15]. A configuration file in which users register their own phrases, applications and replies, together with a simple graphical indicator of the listening state, would further improve everyday usability.

F. Generative Language Models

Open-ended questions are the weakest point of a keyword-driven assistant. A generative language model could be called as an optional fallback when no intent matches, with its answer read aloud by the existing synthesiser. This would widen the range of questions the assistant can answer, but it brings its own concerns: responses can be wrong or invented, the call usually needs a network connection and sends the question to an external service, and the added delay is noticeable in spoken interaction. For these reasons it should remain optional, be clearly labelled as such and never trigger device actions without the confirmation step described in Section VIII.

G. Deployment on Embedded Hardware

Running the assistant on a low-cost single-board computer such as a Raspberry Pi would turn it into a dedicated, always-on device for a room or a workshop. This calls for a USB microphone, a compact offline model and careful use of memory, and it moves the assistant closer to the embedded voice platforms that already run entirely on the device [14]. Under such resource limits the lightweight, rule-based design of the present system becomes an advantage rather than a compromise.

TABLE VI: PROPOSED ROADMAP OF ENHANCEMENTS

Phase	Enhancement	Technology	Expected benefit
1	Offline recognition	Vosk or Whisper	No internet dependency
2	Multilingual input	Hindi and Marathi models	Wider user reach
3	IoT control	MQTT, home-automation hub	Real device control
4	Dialogue memory	Context store, BERT classifier	Natural follow-up commands

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 7, October 2026
Impact Factor: 8.2

X. CONCLUSION

This paper presented an AI based voice command assistant built entirely in Python. By chaining speech recognition, lightweight NLP, command execution and speech synthesis, the system allows users to operate common desktop functions hands-free using natural spoken commands. The project shows that a fully functional assistant is well within reach using open and free tools, and it lays the groundwork for smarter, more accessible and more inclusive hands-free computing. With the planned multilingual, offline and IoT extensions, the system can grow into a practical assistant for education, accessibility and home automation.

The main contribution of the work is a clear, modular reference design. Because every stage of the pipeline is a separate component with a narrow interface, students and developers can replace the recogniser, extend the intent table or add a new action handler without touching the rest of the code. The discussion of limitations, privacy and evaluation in this paper is intended to make clear what the present system can and cannot do, and to give later work a fair basis for comparison.

Future work will concentrate on measuring recognition accuracy, latency and task success under realistic noise conditions, on removing the dependence on web recognition, and on supporting Indian languages so that the assistant is useful to a wider community of users.

ACKNOWLEDGMENT

The authors thank the Department of Computer Engineering, Adsul's Technical Campus, Ahilyanagar, for providing the facilities and guidance that made this project possible.

REFERENCES

[1]. D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. draft. Stanford University, 2023.
[2]. G. López, L. Quesada, and L. A. Guerrero, "Alexa vs. Siri vs. Cortana vs. Google Assistant: A comparison of speech-based natural user interfaces," in Advances in Human Factors and Systems Interaction, Springer, 2018, pp. 241–250.
[3]. G. Hinton et al., "Deep neural networks for acoustic modeling in speech recognition," IEEE Signal Process. Mag., vol. 29, no. 6, pp. 82–97, Nov. 2012.
[4]. D. Povey et al., "The Kaldi speech recognition toolkit," in Proc. IEEE ASRU Workshop, 2011.
[5]. A. Hannun et al., "Deep Speech: Scaling up end-to-end speech recognition," arXiv:1412.5567, 2014.
[6]. A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in Proc. ICML, 2023.
[7]. A. Zhang, "SpeechRecognition: Library for performing speech recognition," Python Package Index. [Online]. Available: https://pypi.org/project/SpeechRecognition/
[8]. Alpha Cephei, "Vosk speech recognition toolkit." [Online]. Available: https://alphacephei.com/vosk/
[9]. N. M. Bhat, "pyttsx3: Offline, cross-platform text-to-speech library for Python," Python Package Index. [Online]. Available: https://pypi.org/project/pyttsx3/
[10]. S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python. Sebastopol, CA, USA: O'Reilly Media, 2009.
[11]. M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017.
[12]. A. Koenecke et al., "Racial disparities in automated speech recognition," Proc. Natl. Acad. Sci. U.S.A., vol. 117, no. 14, pp. 7684–7689, 2020.
[13]. M. B. Hoy, "Alexa, Siri, Cortana, and more: An introduction to voice assistants," Med. Ref. Serv. Q., vol. 37, no. 1, pp. 81–88, 2018.
[14]. A. Coucke et al., "Snips Voice Platform: An embedded spoken language understanding system for private-by-design voice interfaces," arXiv:1805.10190, 2018.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 7, October 2026
Impact Factor: 8.2

[15]. P. Warden, "Speech Commands: A dataset for limited-vocabulary speech recognition," arXiv:1804.03209, 2018.
[16]. J. Brooke, "SUS: A quick and dirty usability scale," in Usability Evaluation in Industry, London, U.K.: Taylor & Francis, 1996, pp. 189–194.
[17]. 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.
[18]. D. Amodei et al., "Deep Speech 2: End-to-end speech recognition in English and Mandarin," in Proc. ICML, 2016, pp. 173–182.