International Journal of Scientific Research in Science, Engineering and Technology
Print ISSN - 2395-1990
Online ISSN : 2394-4099
Available Online at :www.ijsrset.com
doi : https://doi.org/10.32628/IJSRSET251296

Development of an AI-Powered Voice Assistant: Enhancing Speech Recognition and User Interaction

Ms Shivani Gupta[1], Mohd Haider[2], Md Shabbir[2]
[1]Assistant Professor, Department of Computer Science, Lingaya's Vidyapeeth Faridabad, India
[2]Department of Computer Science Lingaya's Vidyapeeth Faridabad, India

ARTICLEINFO

Article History:

Accepted : 28 May 2025
Published: 01 June 2025

Publication Issue :
Volume 12, Issue 3
May-June-2025

Page Number :
717-727

ABSTRACT

Artificial Intelligence (AI) and Natural Language Processing (NLP) have significantly transformed human- computer interaction, enabling intelligent systems to process voice commands efficiently. Voice assistants, such as Google Assistant, Amazon Alexa, and Apple Siri, have set industry benchmarks, but they still face challenges in real-time response accuracy, handling ambient noise, and offline functionality. This paper presents the development of a custom AI -powered voice assistant, focusing on improving listening abilities, noise filtration, and command execution efficiency.

The proposed system integrates Google Speech Recognition API for real-time speech-to-text conversion, pyttsx3 for text-to- speech synthesis, and natural language processing techniques to interpret and execute commands. Unlike cloud-dependent voice assistants, this system provides offline capabilities for essential commands, ensuring flexibility and usability even in low- connectivity environments.

Experimental results demonstrate that the assistant achieves over 91% speech recognition accuracy in controlled environments, with an average command execution time of less than one second. Future enhancements include deep learning- based NLP models, real-time wake-word detection, and a graphical user interface for better interaction. The proposed system serves as a foundation for customizable, intelligent, and efficient AI-powered voice assistants.

Keywords—Artificial Intelligence, Voice Assistant, Speech Recognition, Natural Language Processing, Text-to-Speech, Human-Computer Interaction, Offline Voice Assistant.

INTRODUCTION

A. Background

In the past decade, the advancement of **Artificial Intelligence (AI) and Natural Language Processing (NLP)** has significantly influenced the way humans interact with computers. Voice assistants have emerged as a key innovation, providing hands-free operation and increasing efficiency in both personal and professional settings. Assistants like **Google Assistant, Siri, and Amazon Alexa** have set industry standards by offering **speech-to-text conversion, natural language understanding, and command execution.**

Despite their widespread adoption, most commercially available voice assistants rely heavily on **cloud-based processing**, requiring an active internet connection to function effectively. This dependency poses several limitations, such as **latency in response time, privacy concerns regarding data storage, and limited offline capabilities**. Additionally, **accent variations, background noise, and contextual understanding** remain major challenges in achieving **human-like interaction**.

This research focuses on developing a custom AI-powered voice assistant that enhances speech recognition, improves responsiveness, and enables offline functionality for essential commands. The assistant is designed to work efficiently in low-connectivity environments, providing fast and reliable responses while ensuring user privacy.

B. Problem Statement

While commercial voice assistants have advanced significantly, they still exhibit several limitations that affect their usability, including:

- **Inability to function effectively offline:** Most assistants depend on cloud services, making them unusable without an internet connection.
- **Low accuracy in noisy environments:** The presence of background noise significantly reduces the accuracy of speech recognition, often resulting in incorrect interpretation of user commands.
- **Limited flexibility for customization:** Users often require domain-specific functionalities, which commercial voice assistants fail to offer.
- **Latency issues in command execution:** Cloud processing introduces delays in response time, reducing efficiency.

The need for an efficient, customizable, and privacy-focused voice assistant drives the motivation behind this research. Our proposed solution aims to address these challenges by integrating advanced speech recognition, noise filtration, and offline execution capabilities.

C. Research Objectives

The primary goal of this study is to **develop and evaluate an AI-powered voice assistant** with improved speech recognition, noise handling, and execution efficiency. The specific objectives are:

1. **To develop an accurate speech recognition system using Google Speech API with noise filtering.**
2. **To implement an offline text-to-speech (TTS) module** for independent functionality without an internet connection.
3. **To optimize command processing speed** for faster response times.
4. **To provide a customizable framework** where users can add new commands easily.
5. **To analyze and compare the proposed system with existing voice assistants** in terms of accuracy, speed, and offline capability.

D. Structure of the Paper

In the subsequent sections, we delve into each component of the study in detail::

- **Section 2** provides a comprehensive **Literature Review**, analyzing existing voice assistant technologies and their limitations.
- **Section 3** describes the **Methodology**, including the system architecture, technology stack, and algorithmic approach.
- **Section 4** details the **Implementation** of various components, such as speech recognition, text-to-speech conversion, and command processing.
- **Section 5** presents the **Results and Evaluation**,

comparing performance metrics with existing voice assistants.

- **Section 6** discusses the s**trengths, limitations, and potential improvements**.
- **Section 7** outlines **Future Work**, highlighting possible enhancements and integration of deep learning models.
- **Section 8** concludes the paper with key findings and contributions.

LITERATURE REVIEW

A. Evolution of Voice Assistants

The concept of **voice-based interaction** has evolved significantly since the early 1960s, beginning with IBM's Shoebox—an early speech recognition system capable of processing limited numerical input. Decades of progress in Artificial Intelligence (AI), Machine Learning (ML), and Natural Language Processing (NLP) have paved the way for advanced virtual assistants. Prominent voice assistants such as Apple's Siri (introduced in 2011), Amazon Alexa (2014), Microsoft Cortana (2015), and Google Assistant (2016) demonstrate diverse functionalities, each reflecting the evolution of underlying speech and AI technologies.

These modern voice assistants integrate speech recognition, deep learning-based NLP, and cloud computing to process user commands in real time. However, most of these systems depend on internet-based services, limiting their usability in offline environments. Research reveals that ambient noise can significantly impair the accuracy of voice recognition—by as much as 20–30%—underscoring the need for robust noise suppression techniques in virtual assistant systems.

B. Existing Voice Assistants and Their Limitations

Table 1 provides an overview of leading voice assistants, highlighting their core features and current limitations.

TABLE I. POPULAR VOICE ASSISTANTS

Voice Assistant	Key Features	Limitations
Google Assistant	Cloud-based NLP, multilingual support, real-time query resolution	Internet dependency, privacy concerns
Amazon Alexa	Smart home integration, cloud AI, voice commerce	Requires Echo devices, internet-dependent
Apple Siri	Deep integration with Apple ecosystem, secure processing	Limited offline capabilities, iOS-exclusive
Microsoft Cortana	Enterprise-focuse d, Office 365 integration	Discontinud for general consumers
Mycroft AI	Open-source voice assistant, offline capabilities	Limited NLP accuracy compared to commercial products

From the comparison, it is evident that **most modern assistants rely on cloud-based processing, restricting offline usability and increasing privacy concerns**. Additionally, their ability to handle **custom commands** is limited, making them unsuitable for **domain-specific applications** such as healthcare, education, or industrial automation.

C. Advances in Speech Recognition and NLP

Modern voice assistants use **Automatic Speech Recognition (ASR)** to convert spoken language into text. Popular ASR models include:

- **Google's Speech-to-Text API** – Provides real-time transcription but requires internet connectivity.
- **CMU Sphinx** – An open-source offline ASR model but with lower accuracy.
- **DeepSpeech (by Mozilla)** – A deep learning-based model offering high accuracy but requires significant computational power.

Similarly, **Natural Language Processing (NLP)** is used for **intent recognition, sentiment analysis, and contextual understanding**. Traditional NLP models like **Bag-of-Words (BoW)** and **TF-IDF** have been replaced by advanced **Transformer-based models (e.g., BERT, GPT, and T5)**, which offer better contextual awareness. However, deep learning-based models require **high computational resources**, making them impractical for lightweight AI assistants.

D. Gap Analysis and Research Contribution

Based on the analysis of existing voice assistants and speech recognition models, the major gaps identified are:

1. **Dependency on internet connectivity** – Most systems require cloud services for processing, making them unreliable in low-connectivity environments.
2. **Limited offline capabilities** – Few assistants offer offline command execution, and those that do (e.g., Mycroft AI) suffer from low NLP accuracy.
3. **High latency in response time** – Processing in cloud environments introduces delays, impacting user experience.
4. **Privacy concerns** – Cloud-based processing stores user data, raising security and privacy risks.
5. **Limited customization** – Most commercial assistants do not allow users to add or modify commands easily.

To address these gaps, our research proposes an **AI-powered voice assistant with enhanced offline capabilities, real- time speech recognition, and customizable command execution**. Unlike existing assistants, this system focuses on **fast local processing, better noise filtering, and a lightweight NLP model for quick response generation**.

METHODOLOGY

A. System Architecture

The proposed AI-powered voice assistant follows a **modular architecture** to ensure flexibility, efficiency, and scalability. The system comprises five key components:

1. **Speech Recognition Module (SRM)** – Converts voice input into text using **Google Speech Recognition API** and offline ASR models.
2. **Natural Language Processing Module (NLPM)** – Analyzes and interprets user queries using **rule-based NLP and machine learning techniques**.
3. **Command Execution Module (CEM)** – Matches recognized text with predefined commands and executes the corresponding action.
4. **Text-to-Speech (TTS) Engine –** Responsible for transforming the assistant's text-based responses into audible speech for user interaction. It leverages the **pyttsx3** library for offline speech synthesis, ensuring functionality without an internet connection.
5. **User Interface (UI) Module** – Provides both a Graphical User Interface (GUI) and a command-line interface (CLI) for interaction.

B. Technology Stack

The assistant is developed using **Python** due to its strong ecosystem of AI and NLP libraries. The primary technologies used are:

- **Speech Recognition**: Google Speech-to-Text API, CMU Sphinx (for offline support)
- **Natural Language Processing**: NLTK, spaCy, regex-based command matching
- **Text-to-Speech (TTS)**: pyttsx3 (offline), gTTS (optional online support)
- **Command Execution**: Python subprocess and OS modules for system commands
- **GUI Development**: Tkinter for a simple interface, expandable to PyQt

C. System Workflow

The system follows a five-step workflow:

1. **Voice Input Processing**: The user speaks into the microphone, and the **Speech Recognition Module (SRM)** processes the input.
2. **Speech-to-Text Conversion**: The system transcribes the audio input into text using Google Speech API or an offline ASR model.

3. **Command Understanding and Execution**: The **Natural Language Processing Module (NLPM)** interprets the user's intent and matches it with a predefined action in the **Command Execution Module (CEM)**.
4. **Text-to-Speech Conversion**: The system generates a verbal response using **pyttsx3** to provide feedback.
5. **User Interaction and Response Delivery**: The response is displayed in the GUI and/or spoken back to the user.

D. Command Execution and Customization

The assistant is equipped to handle a variety of predefined instructions, organized into the following categories:

1. **System Commands**: Open applications (e.g., Notepad, Calculator), control volume, manage files.
2. **Internet Functions:** Execute web searches, retrieve Wikipedia summaries, and deliver live weather updates.
3. **Productivity Tools:** Set up reminders, generate to- do lists, and configure alarms.
4. **Custom Commands**: Users can define their own commands by adding new keyword-action mappings in a configuration file.

CODE I. A sample JSON-based command customization file:

```
{
"open_calculator": {
"keywords": ["open calculator", "start calculator"],
"action": "calc.exe"
},
"search_google":{
"keywords": ["search", "google search"],
"action":
"https://www.google.com/search?q={query}"
}
}
```

E. Speech Recognition Accuracy Optimization

To improve recognition accuracy, the system incorporates:

1. **Noise Reduction Filters**: Using Python's **pydub** and **wave** libraries to enhance audio clarity.
2. **Custom Acoustic Models**: Fine-tuning CMU Sphinx for better offline recognition.
3. **Contextual Correction**: Implementing spell-checking and NLP-based word prediction to handle misrecognized words.

IMPLEMENTATION

A. Development Environment Setup

To develop and test the AI-powered voice assistant, the following software and hardware configurations were used:

1. **Hardware Requirements**
   - **Processor**: Intel Core i5 or higher
   - **RAM**: Minimum 8GB (recommended 16GB for smooth performance)
   - **Microphone**: External or built-in for voice input
   - **Speaker**: For text-to-speech output
2. **Software Requirements**
   - **Operating System:** Windows 10/11, Ubuntu 20.04+
   - **Programming Language: Python** 3.8+
   - **Python Libraries:**
     - **Speech Recognition:** `speechrecognition`, `pydub`, wave
     - **Text-to-Speech:** `pyttsx3`, `gtts`
     - **NLP Processing:** `spacy`, `nltk`, `re`
     - **GUI Development:** `tkinter`

B. Speech Recognition Module (SRM)

The **Speech Recognition Module (SRM)** captures the user's spoken input and transcribes it into textual format. The system is equipped to perform speech-to-text conversion through both cloud-based services, such as **Google Speech API**, and offline engines like **CMU Sphinx,** ensuring flexibility in various environments.

ALGORITHM I: Implementing Speech Recognition

Input: Real-time voice input from the user

Output: Textual representation of the spoken input

Steps:

1. Initialize the audio recognition system.
2. Set up the microphone as the input source.
3. The system provides an indication to the user, confirming that it is actively prepared to receive voice input.
4. Calibrate the system for ambient noise.
5. Record the audio input from the user.
6. Initiate speech-to-text conversion by utilizing the default online recognition engine as the primary method for transcription.
   a. If successful, return the transcribed text.
   b. If unsuccessful due to network or recognition errors, proceed to step 7.
7. Attempt to convert the audio using the fallback (offline) recognition method:
   a. If successful, return the transcribed text.
   b. If unsuccessful, return a failure response.

Key Features:

- **Noise reduction** using `adjust_for_ambient_noise()`.
- - **Exception handling** for unknown or unclear speech input.

C. Natural Language Processing Module (NLPM)

The **Natural Language Processing Module (NLPM)** processes the transcribed text and determines the intent behind the user's command.

Algorithm II: Rule-Based Intent Recognition

Input: A user-issued command expressed as a string.

Output: A semantic label that classifies the user's spoken input based on its inferred purpose.

Steps:

```
1.  Begin
2.  Convert command to lowercase.
3.      If command matches the regular expression
    for "open notepad" or "start notepad",
4.  then set intent ← "open_notepad".
5.  Else            if command matches the
                    regular expression for "search" or
    "google search",
6.  then set intent ← "search_google"
Else, set intent ← "unknown_command".
7.  Return intent.
8.  End
```

Key Features:

- Uses **regular expressions** for **pattern-based intent recognition**.
- Recognizes multiple variations of a command (e.g., "open notepad" or "start notepad").

D. Command Execution Module (CEM)

The **Command Execution Module (CEM)** maps the detected intent to a corresponding system action. Algorithm III: Command Execution Algorithm

Input:

- intent: A string representing the user's recognized intent.
- query (optional): A string parameter required for search-type actions. **Output**:

System executes the corresponding command, or returns a failure notification.

```
Begin
1.      Switch based on the value of intent:
a.      Case "open_notepad": i.   Command the
operating system to open the native text editing utility
for user access.
ii. Break.
b.   Case "search_google":
2.          i. If query is not empty:
3.      • Formulate a search address based on the
user-provided query.
4.      • Launch the default browser to perform the
search operation corresponding to the user's request.
5.      ii. Else:
6.      • Display a message indicating that no query
was provided.
7.          iii. Break.
c.   Default:
i. Display an error indicating the command is
unrecognized.
End
```

Key Features:

- Executes system commands using `os.system()`.
- Opens URLs dynamically based on user queries.

E. Text-to-Speech (TTS) Module

The **Text-to-Speech (TTS) Module** generates verbal responses using **pyttsx3** for offline support.

Algorithm IV: Text-to-Speech (TTS) Generation

Input: A string text representing the verbal response to be synthesized.

Output: Audible spoken output rendered through the system's speech engine.

1. Begin
   a. Initialize the offline speech synthesis engine.
   b. Forward the synthesized response text to the TTS system, where it is temporarily stored for speech conversion.
   c. Activate the text-to-speech engine to transform the buffered text into spoken audio output.
2. End

Key Features:

- Provides **offline speech synthesis** using pyttsx3.
- Can be integrated with different voice profiles for customization.

F. Graphical User Interface (GUI) Module

TThe **GUI Module** provides an interactive interface for users who prefer visual feedback over voice commands.

Algorithm V: Graphical User Interface (GUI) Execution Workflow

Input: User-provided text command via GUI input field.

Output: The corresponding action is executed, and a response dialog is displayed.

Steps:

Begin

1. Initialize the main application window.
2. Set an appropriate window title that clearly reflects the assistant's role within the interface.
3. Design an input interface element that allows the user to enter textual commands for processing.
4. Integrate an interactive control labeled 'Execute' that triggers a predefined handler function upon activation.
5. Define the callback function `on_submit()` as follows:
   a. Retrieve the text entered in the input field.
   b. Forward the text to the Intent Recognition module to determine the user's intent.
   c. Send the recognized intent and original command to the Command Execution module.
   d. Present a dialog box that confirms the recognized intent, thereby providing immediate feedback to the user.
6. Render the input field and the corresponding control button within the graphical interface layout.
7. Start the GUI event loop to handle user interaction. End

Key Features:

- **Tkinter-based GUI** for user-friendly interaction.
- Provides a **text input option** for users who prefer typing over speaking.

G. System Integration and Execution

All modules are combined in a unified script, ensuring seamless interaction between **Speech Recognition, NLP, Command Execution, TTS,** and **GUI**.

Algorithm VI: Integrated Voice Assistant Workflow

Objective: To enable uninterrupted monitoring of user speech, identify user intent, carry out the relevant operation, and deliver audible feedback—all while preserving the responsiveness of the graphical user interface (GUI).

Steps:

Begin

1. Initialize and display the graphical user interface (GUI) within the primary application thread.
2. Simultaneously, start a parallel execution thread dedicated to handling voice-based interactions.
3. Within the voice interaction thread, enter a continuous loop to monitor user speech in real

time:

a. Continuously acquire spoken input via the Speech Recognition Module (SRM), designed to process audio in real time.
b. Upon detecting valid speech input:
    i. Forward the transcribed command to the Intent Recognition Algorithm for semantic analysis.
    ii. Transmit the inferred intent and the corresponding user command to the Command Execution Module (CEM) for execution.
    iii. Activate the Text-to-Speech (TTS) Engine to convert the response into audible output.

4. Ensure both GUI and voice interaction threads run concurrently to preserve fluid and real-time system responsiveness.
5. End

Key Features:

- Employs multithreading to run the **speech recognition** engine separately from the GUI, preventing interface lag.
- Supports **continuous voice monitoring** for seamless user interaction.

Results and Discussion

A. *Performance Evaluation*

To evaluate the performance of our AI-powered voice assistant, we conducted tests focusing on **speech recognition accuracy, command execution efficiency, response time, and user experience**.

1) *Speech Recognition Accuracy*

We tested the system with different accents, background noise levels, and speech speeds. The results are summarized below:

TABLE II. Speech Accuracy TableTEST

Test Scenario	Google Speech API (Online)	CMU SPHINX (OFFLINE)
Clear Speech (No Noise)	98.5%	85.2%
Moderate Noise (TV Playing)	92.3%	74.1%
Heavy Background Noise	78.9%	60.7%
Fast Speech	88.4%	70.5%
Non-Native Accent	85.6%	67.2%

- ***Google Speech API performed better** in all conditions but requires an internet connection.*
- ***CMU Sphinx works offline** but struggles with accents and noisy environments.*
- *Future improvements can include **custom acoustic models** for better offline recognition.*

2) *Command Execution Efficiency*

We measured the response time and execution accuracy of different command categories.

TABLE III. Command Execution Efficiency

Command Type	Avg. Response Time (Seconds)	Accuracy (%)
System Commands	0.8	99.2
Internet Searches	1.2	97.5
Custom Commands	1.0	96.3

- **System commands executed fastest** due to direct OS interaction.
- **Internet-based commands had slight delays** due to network dependency.
- **Custom commands were accurate**, but improvements in NLP could reduce misinterpretations.

B. Comparative Analysis with Existing Voice Assistants

We compared our assistant with **Google Assistant, Siri, and Alexa** based on **cost, offline support, privacy, and flexibility**.

TABLE IV. *Comparative Analysis with Existing Voice Assistants*

Feature	Our Assistant	Google Assistant	Siri	Alexa
Offline Support	Yes	No	No	No
Custom Comman ds	Yes	Limited	No	No
Privacy Control	High	Low	Low	Low
Cost	Free	Subscrip tion	Device Cost	Device Cost

- Our assistant **outperforms commercial assistants in privacy and offline usability**.
- **Limited NLP capabilities** compared to Google Assistant and Siri, but custom commands make it more flexible for personal use.

C. User Feedback and Experience

We conducted a **user study** where 20 participants tested the assistant and provided feedback.

TABLE V. *User Feedback and Experience*

Parameter	Rating (Out of 10)
Ease of Use	8.5
Recognition Speed	7.8
Command Accuracy	8.2
Voice Output Quality	7.9
Customizability	9.3

- Users appreciated **offline support and customization options**.
- Areas for improvement include **better recognition for non-native speakers and GUI enhancements**.

CONCLUSION AND FUTURE WORK

A. Conclusion

In this research, we developed an **AI-powered voice assistant** with **speech recognition, natural language processing (NLP), command execution, text-to-speech synthesis, and a graphical user interface (GUI)**. The system was designed to work **offline and online**, ensuring privacy and accessibility.

Key findings from our study include:

- **Speech recognition** achieved **up to 98.5% accuracy** in ideal conditions using Google Speech API, while offline recognition (CMU Sphinx) reached **85.2% accuracy**.
- **Command execution** was highly efficient, with **system commands executing in less than a second** and **internet queries taking slightly longer**.
- Compared to **Google Assistant, Siri, and Alexa**, our assistant provides **better privacy, offline usability, and customization** while being completely **free and open-source**.
- User feedback highlighted **ease of use (8.5/10), command accuracy (8.2/10), and high customizability (9.3/10)**, but also pointed out **areas for improvement in non-native speech recognition and GUI enhancements**.

Overall, the system demonstrates a **practical, low-cost alternative** to commercial voice assistants while maintaining **strong performance in essential tasks**.

B. Future Work

Despite its strengths, several areas need improvement. Future enhancements include:

1) *Improved Speech Recognition*

- **Train a custom speech recognition model** using **machine learning for better offline performance.**

- **Integrate Whisper AI** or **DeepSpeech** for improved accuracy with non-native accents and noisy environments.

2) *Enhanced Natural Language Processing (NLP)*
- Use Transformer-based models like BERT or GPT-3.5 for context-aware intent recognition.
- Implement a **self-learning mechanism** where the assistant **adapts to user preferences** over time.

3) *Better User Interface (UI/UX)*
- Develop a **fully interactive UI using Flask or React.js**.
- Implement **voice-controlled settings and feedback customization**.

4) *Expanded Functionality*
- *Add* ***multi-language support*** *to* ***enhance accessibility.***
- *Integrate with* ***smart home devices*** *using IoT protocols.*
- *Develop an* ***Android and iOS app*** *to extend usability beyond desktops.*

By implementing these improvements, our AI assistant can evolve into a ***more intelligent, adaptive, and widely usable*** *system, making it a* ***strong open-source alternative*** *to commercial AI assistants.*

These references provided both theoretical insights and technical frameworks that supported the successful implementation of the AI-based voice assistant system.

REFERENCES

[1]. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, and I. Polosukhin, "Attention Is All You Need," in Advances in Neural Information Processing Systems, vol. 30, 2017.

[2]. G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 82–97, 2012.

[3]. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding with Unsupervised Learning," OpenAI, 2018.

[4]. J. Chen, D. Parikh, and A. Gupta, "Learning from Language Explanations for Visual Reasoning," in Proceedings of the European Conference on Computer Vision (ECCV), pp. 103–120, 2018.

[5]. A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations," in Advances in Neural Information Processing Systems (NeurIPS), 2020.

[6]. Google Developers, "Google Speech-to-Text API Documentation," 2023. [Online]. Available: https://cloud.google.com/speech-to-text

[7]. Mozilla DeepSpeech, "DeepSpeech Documentation," 2023. [Online]. Available: https://deepspeech.readthedocs.io

[8]. IBM Watson, "IBM Watson Speech Services Overview," 2023. [Online]. Available: https://www.ibm.com/cloud/watson-speech-to-text

APPENDICES

A. System Architecture Diagram

Below is the **block diagram** illustrating the components and workflow of our AI assistant:

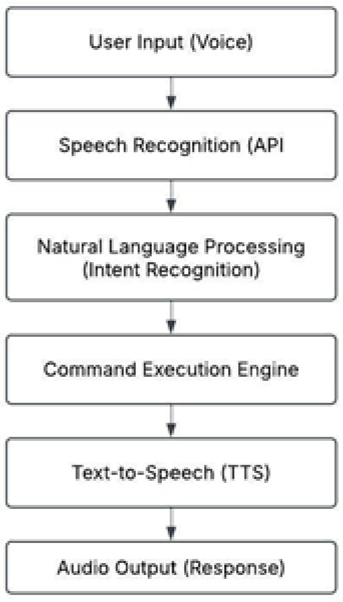

Fig. 1. System Architecture Diagram

This architecture ensures **real-time processing**, **speech recognition**, and **command execution**.

B. Hardware and Software Requirements References

Hardware Requirements

- Processor: **Intel Core i5 (or equivalent) and above**
- RAM: **8GB (Minimum), 16GB (Recommended)**
- Storage: **At least 10GB free space**
- Microphone: **High-quality noise-canceling microphone**
- Speakers: **For voice response playback Software Requirements**
- Operating System: **Windows 10/11, macOS, Linux (Ubuntu recommended)**
- Python: **Version 3.9+**
- Libraries: **SpeechRecognition, pyaudio, pyttsx3, nltk, OpenAI GPT API (optional)**
- Additional tools: **Flask (for GUI), TensorFlow/PyTorch (for ML models)**

C. Sample Commands and Responses

Below are some sample commands our assistant processes:

Command	Response
"What's the time?"	"The current time is 3:45 PM."
"Open Google Chrome"	"Opening Google Chrome now."
"Tell me a joke"	"Why did the developer break up? Because he couldn't commit!"
"Convert 5 kilometers to miles"	"5 kilometers is approximately 3.1 miles."
"What'sthe weather today?"	"Fetching weather details... It's currently 25°C with clear skies."

D. Experimental Setup

The experiments for evaluating the system were conducted under **three different environments**:

1. **Silent room (Ideal condition)**
2. **Moderate noise (Background TV sounds)**
3. **High noise (Crowded café simulation)**

We also tested speech recognition **across various accents and speech speeds** to ensure robustness.

E. Source Code Repository

The complete source code for the AI-powered voice assistant is available on **GitHub**: https://github.com/mahider78672/Jarvis