Jarvis: AI-Powered Voice Assistant Using Python

By :- Kalash Jambhulkar

Abstract

This project focuses on developing a voice-controlled AI desktop assistant named Jarvis using Python. The assistant performs a range of tasks—such as opening applications, translating text, fetching weather updates, searching the web, and playing YouTube videos—via natural speech commands. By integrating libraries like SpeechRecognition, Selenium, and PyAutoGUI, the system bridges human-computer interaction through natural language. Jarvis provides productivity, interactivity, and automation, demonstrating how AI simplifies everyday computing.

Introduction

In the modern digital world, artificial intelligence has redefined convenience and personalization. A significant example is the emergence of voice assistants such as Alexa, Siri, and Google Assistant. This project, Jarvis, is inspired by these systems but implemented entirely in Python to provide an intelligent, extensible, and offline-capable desktop assistant. Jarvis can recognize voice input, respond using speech synthesis, and execute system and internet-based tasks autonomously. The assistant not only accepts spoken instructions but also learns user intent through simple keyword processing and fuzzy logic matching, offering a human-like interactive experience.

Tools Used

- **Python 3.10+** – Core programming language for logic and integration.
- **SpeechRecognition** – Converts real-time voice input into text.
- **pyttsx3** – Provides text-to-speech output for audio responses.
- **Selenium & ChromeDriver** – Automates Brave/Chrome browsers for web tasks.
- **translate** – Performs multilingual text translation.
- **pyautogui & win32com** – For desktop automation and Windows app control.
- **fuzzywuzzy** – Enables fuzzy string matching to detect near app names.
- **datetime & subprocess** – For local operations, scheduling, and execution.

Steps Involved

1. **Speech Recognition Setup**: The recognizer captures microphone input and converts speech to text via the Google API.
2. **Text-to-Speech Integration**: Using pyttsx3, Jarvis gives spoken responses in real time.

3. **Intent Detection**: Commands are parsed and mapped to specific functions (e.g., open, play, translate, weather).
4. **Action Execution**:
    - Application control via os.startfile() and win32com.client.
    - Web automation executed using Selenium and Brave browser.
    - Translation function implemented through the translate library and language set.
5. **Error Handling & User Feedback**: Try/except blocks ensure smooth recovery from unrecognized speech or failed actions.
6. **Continuous Loop Interaction**: Jarvis remains active until the user says “exit,” dynamically responding with context tracking.

Conclusion

Jarvis successfully demonstrates the application of artificial intelligence and Python automation in seamless human-computer interaction. The system integrates multiple modules to handle speech, automation, and logical reasoning, making it capable of performing complex desktop and web functions through simple voice commands. This project enhances both accessibility and productivity, showing how AI assistants powered by open-source technologies can evolve into powerful personal companions in computing.

Future Scope

Future enhancements can include integration with OpenAI APIs for conversational learning, addition of natural language understanding (NLU), and GUI-based customization for broader accessibility on different operating systems.

PDFVoice: AI-Powered PDF to Audiobook Converter Using Python

By – Kalash Jambhulkar

Abstract

This project presents *PDFVoice*, an intelligent desktop application developed in Python that converts textual PDF documents into audio files using text-to-speech (TTS) processing. The system extracts readable text content from PDF files using *pdfplumber*, converts it into speech with *gTTS (Google Text-to-Speech)*, and plays the generated audio via *Pygame*. Built with *Tkinter*, the application provides users with an accessible graphical interface that supports uploading documents, adjusting playback speed and volume, pausing or stopping playback, and exporting the audiobook as an MP3 file. The project aims to enhance accessibility for visually impaired users and deliver a seamless reading experience through automation and artificial speech.

Introduction

In an age dominated by digital information, accessibility and convenience have become essential aspects of software design. Reading long or complex PDF documents can be challenging, especially for individuals with visual impairments or busy professionals who prefer auditory learning.
The *PDFVoice* system addresses these challenges by offering a simple yet effective Python-based GUI tool that converts readable text into a spoken audiobook format.

Using libraries like *pdfplumber* for text extraction, *gTTS* for TTS synthesis, and *Pygame* for playback, the application provides an integrated environment where users can transform static text into interactive audio with user-friendly controls. Through its real-time processing capabilities, multithreaded playback monitoring, and export feature, this system demonstrates practical integration of automation and artificial intelligence in accessibility-focused desktop tools.

Tools Used

- **Python 3.10+** – Core programming language for backend logic and GUI structure.
- **Tkinter** – Used for graphical interface design and user interaction.
- **pdfplumber** – Extracts text efficiently from PDF pages.
- **gTTS (Google Text-to-Speech)** – Converts text into natural-sounding audio.
- **Pygame** – Handles audio playback including play, pause, and volume control.
- **threading** – Enables non-blocking background operations during playback.
- **re (Regular Expressions)** – Cleans and refines extracted text for proper narration.
- **os & filedialog** – Supports file selection, export, and media handling.

Steps Involved

1. **GUI Initialization**
   The system initializes a Tkinter-based window with interactive buttons, text area, sliders, and status indicators for an intuitive user interface.

2. **PDF Upload and Extraction**
   On user command, *pdfplumber* scans each page of the uploaded document and extracts structured text while ignoring blank or image-only pages.

3. **Text Cleaning**
   The extracted text undergoes preprocessing via regex cleaning to remove excessive whitespace and formatting anomalies, ensuring smooth audio generation.

4. **Audio Generation**
   The cleaned text is converted into speech using *gTTS*. The application generates a temporary MP3 file for real-time playback.

5. **Playback Management**
   Pygame handles audio streaming, allowing users to **Play**, **Pause**, or **Stop** playback, supported by volume and speed adjustments for personalized listening.

6. **Export Functionality**
   The generated audio can be saved permanently as an MP3 file through the export button, enabling users to preserve or share their audiobook.

7. **Thread-Safe Interaction**
   The implementation of a background thread monitors active playback, ensuring a smooth GUI experience without freezing or lag.

8. **Keyboard Shortcuts and Accessibility**
   The program integrates standard shortcuts (e.g., **Ctrl+O**, **Ctrl+P**, **Ctrl+S**, **Ctrl+E**) and initializes default focus for improved accessibility.

Conclusion

The *PDFVoice* project successfully transforms the written content of PDFs into spoken-word audiobooks using open-source Python libraries. Its blend of GUI design, text extraction, TTS synthesis, and interactive playback demonstrates a seamless convergence of usability and technical efficiency. By focusing on accessibility and automation, *PDFVoice* expands the potential of assistive tools and offers an inclusive way to interact with textual data beyond the screen.

Future Scope

Enhancements to *PDFVoice* can include:

- Integration of **offline TTS engines** (e.g., pyttsx3) for non-internet usage.
- **Multilingual support** for non-English documents.
- Addition of **text highlighting** syncing with audio playback.
- **dDark mode UI** for greater visual comfort.

- Automatic **batch processing** of multiple PDF files.
- Machine learning-based **speaker customization** for more natural narration.