KAI: A Real-Time Speech-Based Assistant System for Task Automation with Online Information Retrieval

1st Kshitij Singh
Department of Computer Science
Netaji Subhas University of Technology
New Delhi, India
kshitij.singh.ug22@nsut.ac.in

2nd Aryan Raj Singh
Department of Computer Science
Netaji Subhas University of Technology
New Delhi, India
aryan.singh.ug22@nsut.ac.in

3rd Sanir
Department of Computer Science
Netaji Subhas University of Technology
New Delhi, India
sanir.ug22@nsut.ac.in

***Abstract*—The proliferation of voice-based interaction paradigms necessitates exploring customizable and transparent alternatives to mainstream cloud-dependent virtual assistants. This paper presents the design and implementation of a desktop voice assistant built using Python. The system leverages standard libraries for core functionalities: speech recognition (`speech_recognition` with Google's backend), text-to-speech synthesis (`pyttsx3` utilizing macOS's `nsss` engine), and global hotkey activation/deactivation (`pynput`). It integrates with various external APIs and local system commands to perform tasks such as launching applications (`subprocess`), retrieving online information (IP address lookup, Google search, Wikipedia summaries, YouTube playback via `pywhatkit`, news headlines via NewsAPI, weather forecasts via OpenWeatherMap), querying knowledge bases (IMDb via `imdbpy`, WolframAlpha), and engaging in basic conversational exchanges. The assistant features a modular design separating online functionalities and employs hotkeys for user-controlled listening, enhancing usability and privacy. This work demonstrates the feasibility of creating a capable desktop assistant using readily available tools, offering a foundation for further customization and development.**

***Index Terms*—Voice Assistant, Speech Recognition, Text-to-Speech (TTS), Python, Human-Computer Interaction (HCI), Desktop Automation, API Integration, Hotkeys, `speech_recognition`, `pyttsx3`, `pynput`**

I. Introduction

Voice User Interfaces (VUIs) have become increasingly prevalent, with commercial assistants like Siri, Alexa, and Google Assistant integrating into various devices. While offering convenience through hands-free operation, these systems often operate as "black boxes," raising concerns about data privacy and offering limited customizability for specific desktop workflows [1]. Traditional desktop interaction relies heavily on keyboard and mouse input, which can be inefficient for certain quick tasks or information lookups. The motivation behind this work stems from the desire for a more transparent, user-controlled, and potentially extensible voice assistant tailored for desktop environments. Existing commercial solutions lack deep integration with arbitrary desktop applications and workflows unless specifically supported. This paper details the development of a prototype desktop voice assistant using Python, aiming to bridge this gap.

The System is designed to:

- Listen for voice commands upon activation via a global hotkey.
- Recognize spoken commands using a readily available speech recognition service.
- Interpret commands based on predefined keywords to trigger specific actions.
- Execute tasks including launching local applications, performing web searches, querying online databases and APIs for information (news, weather, movies, general knowledge, calculations), and finding the system's IP address.
- Provide spoken feedback to the user using a text-to-speech engine.
- Allow pausing of the listening process via another global hotkey.

This work contributes a practical implementation demonstrating the synergy of various Python libraries to create a functional desktop voice assistant, emphasizing modular design and user-controlled activation through hotkeys.

II. Related Work

The development of automated assistants draws from several fields. Commercial voice assistants (Amazon Alexa, Google Assistant, Apple Siri) represent the state-of-the-art in cloud-based NLU and extensive skill ecosystems but often lack transparency and deep desktop customizability [1]. Open-source alternatives like Mycroft [2] and Rhasspy [3] offer greater control and privacy, often employing more sophisticated local NLU engines (like Snips NLU, Rasa NLU) and wake-word detection. However, they typically involve a more complex setup and architecture compared to the direct scripting approach presented here.

Core technologies leveraged include:

- **Speech-to-Text (STT):** Early voice recognition systems relied on Hidden Markov Models (HMMs). Modern

approaches, including the Google Web Speech API (accessed via the `speech_recognition` library [**?**]), utilize deep neural networks, offering high accuracy but requiring internet connectivity.

- **Text-to-Speech (TTS):** Synthesis techniques have evolved from concatenative methods to neural network-based parametric synthesis. The `pyttsx3` library [**?**] serves as a wrapper around native OS TTS engines (such as 'nsss' on macOS, used in this project), enabling offline speech synthesis.
- **Intent Recognition:** While advanced systems employ natural language understanding (NLU) pipelines, this project adopts a simpler keyword-spotting technique that matches predefined phrases in the recognized text to corresponding functions.
- **API Integration:** External services are accessed via RESTful APIs (e.g., NewsAPI [**?**], OpenWeatherMap [**?**], WolframAlpha [**?**]), allowing the assistant to deliver enriched functionality without requiring complex domain-specific logic to be implemented locally.

Our work combines these elements, focusing on ease of implementation using high-level Python libraries and direct integration with desktop functions and web services, using hotkeys as a primary interaction trigger.

III. System Architecture and Methodology

The proposed system operates based on a main loop controlled by a listening state, toggled via global hotkeys. Its architecture comprises three logical stages: Activation and Input, Command Processing and Execution, and Feedback.

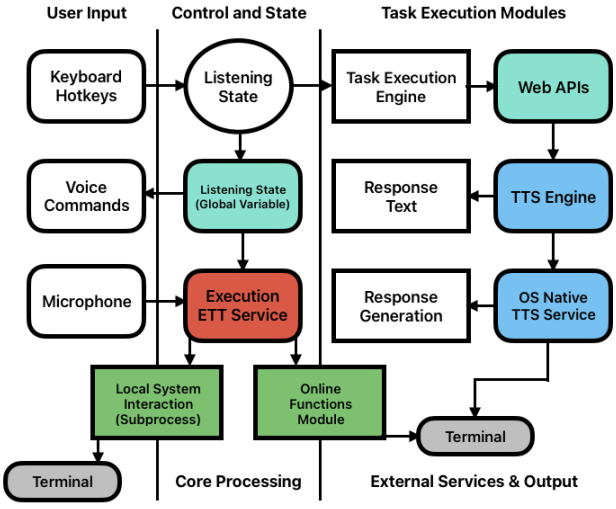

Fig. 1: Overview of the AI Virtual Assistant

A. System State Management

A global boolean variable `listening` controls whether the system actively listens for commands. This state is modified by callback functions (`on_activate_k`, `on_activate_p`) that are triggered by the `pynput` hotkey listener, which runs in a dedicated background thread using the `threading` module.

B. Speech Input and Recognition

- **Initialization:** `sr.Recognizer` is initialized. Microphone selection uses a hardcoded `device_index` (which may be a limitation). Ambient noise calibration is performed using `adjust_for_ambient_noise`.
- **Capture and Recognition:** `r.listen()` captures audio when `listening` is `True`. `r.recognize_google()` sends the audio to Google's cloud service for transcription.
- **Error Handling:** A `try-except` block handles potential `sr.UnknownValueError` or `sr.RequestError`, prompting the user to repeat the command.

C. Command Processing and Task Execution

- **Parsing:** The recognized query (converted to lowercase) is checked against a series of `if/elif` conditions containing keywords (e.g., `"open terminal"`, `"ip address"`, `"youtube"`, `"wikipedia"`, `"calculate"`, `"what is"`).
- **Local Execution:** The command `subprocess.run(["open", "-a", app_name])` is used to launch macOS applications.
- **Online Execution:** Functions defined in `online.py` handle API calls and web interactions:
  - `find_my_ip()`: Uses `requests` to query `api.ipify.org`.
  - `search_on_wikipedia()`: Uses the `wikipedia` library.
  - `search_on_google()`, `youtube()`: Use the `pywhatkit` library (likely wraps browser automation).
  - `get_news()`: Uses `requests` to query `NewsAPI`.
  - `weather_forecast()`: Uses `requests` to query the `OpenWeatherMap` API. Requires city input via `input()`.
  - **Movie search:** Uses the `imdbpy` library.
  - **Calculation/Knowledge:** Uses the `wolframalpha` library.
- **Configuration:** API keys and user/bot names are loaded from environment variables using `python-decouple` for better security and configuration management.

D. Speech Output (TTS)

- **Initialization:** `pyttsx3.init('nsss')` selects the macOS native TTS engine. Voice selection logic attempts to find a preferred voice (e.g., `'daniel'`). The speech rate and volume are also configured.
- **Synthesis:** The method `engine.say(text)` queues the text for speech, and `engine.runAndWait()` blocks further execution until the speech is finished.

IV. Implementation Details

The system is implemented in Python 3.x and relies on several external libraries.

A. Core Libraries

- **speech_recognition:** For capturing microphone input and interfacing with the Google Web Speech API for speech-to-text (STT). Requires `PyAudio` for microphone access.
- **pyttsx3:** A platform-independent wrapper for TTS engines; configured to use macOS's `'nsss'` engine.
- **pynput:** For monitoring global keyboard hotkeys to control the listening state.
- **requests:** For making HTTP requests to various APIs such as `ipify`, `NewsAPI`, and `OpenWeatherMap`.
- **subprocess:** For executing system commands, specifically launching applications on macOS.
- **threading:** To run the hotkey listener concurrently without blocking the main program loop.
- **python-decouple:** To securely manage configuration variables (API keys, user/bot names) via environment variables or a `.env` file.
- **wikipedia:** High-level wrapper for accessing and retrieving Wikipedia content.
- **pywhatkit:** Library for automating web browser actions such as Google searches and YouTube playback.
- **imdbpy:** Library for accessing IMDb movie database information.
- **wolframalpha:** Library for querying the WolframAlpha computational knowledge engine.

B. Hardware

- A standard PC or laptop (tested on MacOS).
- A functional microphone (internal or external).
- Speakers or headphones for audio output.

C. Platform Dependencies

- **TTS Engine:** `pyttsx3.init('nsss')` is specific to macOS. Other platforms require different engine names, such as `'sapi5'` on Windows or `'espeak'` on Linux.
- **Application Launching:** The command `subprocess.run(["open", ...])` is used on macOS to open applications or files. Equivalent commands on other platforms include `start` on Windows and `xdg-open` on Linux. Application paths (e.g., `/System/Applications/`, `/Applications/`) are also macOS-specific.
- **Microphone Index:** The use of `sr.Microphone(device_index=2)` is hardcoded and may vary across systems, especially when external microphones are connected or disconnected.

V. Functionality and Limitations

A. Functionality Showcase

The system successfully implements the following voice commands:

- **Greeting:** Provides a time-dependent greeting upon startup.
- **Basic Conversation:** Responds to simple prompts like `"how are you"`.
- **Application Launch:** Opens macOS applications such as Terminal, Notes, Photo Booth, and Discord using their respective system paths.
- **Information Retrieval:**
  - Provides the public IP address.
  - Searches Wikipedia for a topic and reads out a summary.
  - Performs a Google search by opening the browser.
  - Plays a specified video on YouTube via browser automation.
  - Fetches and reads out current news headlines.
  - Provides weather forecast (requires typed city input).
  - Searches IMDb for movies and reads out details (e.g., rating, cast, plot).
  - Performs calculations using the WolframAlpha API.
  - Answers general knowledge questions (e.g., `"what is"`, `"who is"`) using WolframAlpha.
- **Control:** Starts/stops listening using `<cmd> + <ctrl> + k/p` hotkeys. Terminates on hearing `"stop"` or `"exit"`.

B. Performance

- **Responsiveness:** Latency is primarily influenced by the network round-trip time when interacting with the Google Speech-to-Text (STT) service and other external APIs. Local tasks, such as application launching, execute nearly instantaneously. Text-to-speech (TTS) generation is also fast.
- **Recognition Accuracy:** Depends on the performance of the Google Web Speech API, the quality of the microphone, and ambient noise conditions. Accuracy is generally high for clear and well-articulated speech.

C. Limitations

- **Platform Dependence:** As discussed in Section **??**, the TTS engine selection, application launch commands, and file paths are specific to macOS. The use of a hardcoded microphone index also reduces portability across systems.
- **Basic NLU:** The system relies solely on keyword spotting within user queries. It lacks the ability to interpret phrasing variations, contextual meaning, or complex instructions. This approach is prone to errors when keywords appear out of context.
- **Network Dependency:** Core speech-to-text (STT) functionality and all online capabilities (e.g., news, weather, Wikipedia, WolframAlpha) require a stable internet connection.
- **Hardcoded Configuration:** While API keys are externalized using `decouple`, the microphone index and application paths remain hardcoded in the source code, limiting flexibility and maintainability.

- **Modal Input Mixing:** The `weather_forecast()` function disrupts the voice-only interaction model by requiring typed input (`input()`) for the city name.
- **Error Handling:** Basic `try-except` blocks are implemented for STT. However, API error handling (e.g., network issues, invalid API keys, rate limits, or empty results) is minimal and lacks user-friendly feedback.
- **Scalability:** The linear `if/elif` structure used for command parsing becomes increasingly difficult to manage and inefficient as the number of supported commands grows.
- **Security:** Although `decouple` helps protect secrets, ensuring the security of the `.env` file is essential. The use of `pywhatkit` for launching web searches introduces potential security concerns depending on browser settings and search result content. Additionally, no liveness detection is implemented, leaving the system vulnerable to voice replay attacks (though this is less critical for a desktop assistant than for authentication systems).

VI. Comparison with Existing Voice Assistants (Conceptual)

Compared to mainstream commercial assistants and advanced open-source frameworks, this system offers:

Advantages:

- **Transparency:** The full codebase is available and understandable.
- **Customizability:** Relatively easy to add new commands, integrate with specific local scripts or applications, and modify behavior.
- **Control:** Hotkey activation provides explicit user control over when the assistant listens, potentially enhancing privacy perception compared to always-on wake-word detection.
- **Offline TTS:** Utilizes the OS's native engine for speech output.

Disadvantages:

- **Limited NLU:** Significantly less sophisticated language understanding; cannot handle conversational context, ambiguity, or complex grammar.
- **Platform Dependent:** Not readily cross-platform without modification.
- **Network Dependent STT:** Relies on Google's cloud service for speech recognition.
- **Smaller Feature Set:** Lacks the vast ecosystem of skills/integrations found in commercial platforms.
- **No Wake-Word:** Requires explicit hotkey activation.

VII. Conclusion and Future Work

This paper successfully presented the development of a functional Python-based desktop voice assistant. By integrating libraries for speech recognition, text-to-speech, hotkey monitoring, system process execution, and various web APIs, the system demonstrates the capability to automate desktop tasks and retrieve online information via voice commands. The modular separation of online functions (online.py) and the use of hotkeys for activation enhance maintainability and user control.

Despite its functionality, the system has limitations that offer clear directions for future work:

Proposed Enhancements

Cross-Platform Compatibility:

Refactor platform-specific code (TTS engine selection, application opening commands, paths) using conditional logic or cross-platform libraries. Implement dynamic microphone detection.

Enhanced NLU:

Replace the if/elif structure with a more robust intent recognition engine (e.g., using libraries like Rasa NLU, spaCy for rule-based matching, or even a simple intent parsing class) to handle more varied phrasing.

Voice-Based Secondary Input:

Modify functions like `weather_forecast` to accept follow-up information (like city name) via voice instead of keyboard input.

Improved Error Handling:

Implement more specific exception handling for API calls (network errors, invalid responses, rate limits) and provide clearer feedback to the user.

Wake-Word Activation:

Integrate a lightweight wake-word engine (e.g., Porcupine) as an alternative or addition to hotkey activation.

Configuration Interface:

Develop a simple GUI or configuration file standard for users to easily set preferences (microphone, voice, application paths, API keys).

Context Management:

Implement basic state tracking to allow for simple multi-turn interactions or clarifications.

Expanded Skillset:

Integrate with more APIs and local applications (e.g., calendar, email clients, media players).

In conclusion, this project serves as a valuable proof-of-concept and a flexible foundation. With targeted enhancements addressing robustness, natural language understanding, and cross-platform support, it has the potential to evolve into a more powerful and personalized desktop voice assistant.

REFERENCES

1) J. Pearl, "The Seven Tools of Causal Inference, with Reflections on Machine Learning," *Communications of the ACM*, vol. 62, no. 3, pp. 54-60, Mar. 2019. (Note: This is an example reference style; replace with relevant citations for VUI background/privacy).
2) Mycroft AI. [Online]. Available: https://mycroft.ai/
3) Rhasspy Voice Assistant. [Online]. Available: https://rhasspy.readthedocs.io/
4) SpeechRecognition Library Documentation. [Online]. Available: https://pypi.org/project/SpeechRecognition/
5) pyttsx3 Library Documentation. [Online]. Available: https://pypi.org/project/pyttsx3/
6) NewsAPI Documentation. [Online]. Available: https://newsapi.org/docs
7) OpenWeatherMap API Documentation. [Online]. Available: https://openweathermap.org/api
8) WolframAlpha API Documentation. [Online]. Available: https://products.wolframalpha.com/api/
9) Picovoice Porcupine Wake Word Engine. [Online]. Available: https://picovoice.ai/platform/porcupine/