Facial Recognition Assistant Test (FRAT) Report

https://github.com/TheAvergers/FRAT

Aidan Donnellan

EGR 404

Monday 5th, 2025

1. Overview and Purpose

The “FRAT” prototype represents an integration of local device control with cloud-based artificial intelligence services to create a “comprehensive” voice-activated assistant system. The architecture enables users to interact through voice commands and face recognition technology to perform various functions including setting reminders and timers, controlling smart home devices, scheduling events, and requesting information through natural language queries.

The system architecture comprises a Raspberry Pi equipped with camera and microphone peripherals, a Windows PC hosting the primary assistant software, real-time audio and video streaming between these devices, and OpenAI cloud services for speech and language processing capabilities. The core functionality includes face recognition for user identification and personalized greetings, wake word detection utilizing OpenAI's Whisper for recognizing the activation phrase "Hey Assistant," command handling through a dedicated Command Handler module, artificial intelligence integration leveraging GPT-4 for processing general questions and complex requests, text-to-speech conversion for natural-sounding responses, and scheduling functionality with persistent storage to disk.

This prototype demonstrates the effective integration of disparate technologies to create a functional home assistant that achieves a balance between local processing capabilities and cloud-based artificial intelligence services.

2. Project Milestones

Our development process followed these key milestones:

First milestone focused on materials acquisition, including sourcing a Raspberry Pi, identifying appropriate facial recognition models, and selecting compatible microphone and camera hardware.

Second milestone concentrated on Raspberry Pi configuration to establish reliable streaming to the Windows computer, creating the foundation for our distributed processing architecture.

Third milestone involved developing the basic Python codebase and implementing initial transcription functionality, though this early version exhibited significant accuracy limitations and lacked response capabilities.

Fourth milestone integrated the OpenAI API, enabling communication with ChatGPT through the system interface, representing a significant advance in AI assistant functionality.

Fifth milestone added wake word detection capability and implemented fundamental commands including reminder setting and simulated lighting control, though transcription issues with ambient noise remained problematic.

Sixth milestone substantially improved command robustness through RAG API parsing implementation, added audio buffer management, and refined noise reduction settings, resulting in a significantly more consistent user experience.

Seventh milestone expanded the command set with timer, scheduling functions, and music playback support, while further refining the RAG implementation and upgrading the face recognition model, culminating in the current MVP implementation.

3. System Architecture

The Smart AI Home Assistant implements a comprehensive architecture comprising both hardware components and software modules working in concert:

Hardware Configuration

The hardware architecture consists of a Raspberry Pi equipped with camera and microphone peripherals functioning as the input device, a Windows PC serving as the main processing unit, and UDP streaming protocols facilitating device communication.

Software Modules

The software architecture implements several specialized modules:

The FaceRecognizer module processes video input to identify users through facial recognition. The AudioProcessor handles audio processing functions including wake word detection. The AssistantController serves as the central coordination unit for all component interactions. The CommandHandler processes and interprets user commands. The TextToSpeech module converts text-based responses to audible speech output.

Cloud Services Integration

The system leverages several cloud-based services: OpenAI's Whisper API provides speech recognition and transcription capabilities, GPT-4 delivers advanced language understanding for processing commands and answering questions, and OpenAI's text-to-speech API generates natural-sounding voice responses.

Data Storage

The system maintains persistent storage through JSON files including reminders.json for user reminders and scheduled_tasks.json for pending tasks and scheduled events.

The system operates on an event-driven model with dedicated threads handling specific tasks. The AssistantController functions as the central coordination unit, while face recognition and audio processing operate continuously in separate threads to ensure responsive performance.

3. Command Processing Pipeline

The command processing pipeline represents the core functionality of the system. It implements a Retrieval-Augmented Generation (RAG) approach to handle commands, combining traditional processing methods with artificial intelligence to enhance user interaction understanding.

Face Recognition and Greeting

The face recognition process begins with analyzing the video feed using OpenCV's face detection capabilities. Facial features are extracted and compared against a database of known faces. Upon successful identification, the system recognizes the user and initiates a personalized greeting. The user identity information is retained to enable personalization of subsequent interactions.

Wake Word Detection

The system continuously monitors audio input by processing short segments and analyzing each segment for the presence of the activation phrase "Hey Assistant." Upon detection, the system provides confirmation feedback through an audible signal and prepares to receive the subsequent command by focusing on capturing the user's request.

Audio Capture and Transcription

Following wake word activation, the system initiates audio recording, continuing until reaching either a maximum duration threshold or detecting a silence period indicating command completion. The recorded audio is transmitted to OpenAI's Whisper service for transcription, with the resulting text forwarded to the CommandHandler for processing.

Command Interpretation Using RAG

The system implements a sophisticated command interpretation approach combining retrieval and generative AI:

The retrieval component maintains a reference document containing examples of all supported commands, serving as a knowledge base for command interpretation. When a command is received, this reference guides the interpretation process.

For command normalization, the transcribed text and command reference are provided to GPT-4, which converts natural language input to standardized command formats. For instance, "Could you remind me to call mom tomorrow?" becomes the normalized command "add reminder call mom tomorrow," utilizing examples to guide AI understanding.

The command classification process examines the normalized command against known patterns to determine if it matches recognized command types such as timer, reminder, lighting control, etc. Commands without matches are categorized as "general queries," with the appropriate handler selected based on this classification.

This approach offers significant benefits including natural language understanding, extensibility for new command phrasings without code modifications, AI-assisted bridging between human speech and system functions, and graceful handling of requests outside built-in capabilities.

Command Execution

Based on command classification, the system executes appropriate actions:

Built-in commands initiate specific functions: timers trigger countdowns with alerts upon completion, reminders are saved with unique identifiers, lighting commands (currently simulated) would interface with smart home devices, and music playback accesses local audio files.

General queries are routed to GPT-4 with contextual information about the assistant role and potentially user identity for personalization. The AI generates conversational responses, and the system analyzes these responses for implied actions.

Text-to-Speech Output

The final stage converts response text to speech using OpenAI's synthesis API, playing audio through PC speakers. A locking mechanism prevents overlapping speech, and the system subsequently returns to its listening state awaiting the next command.

4. Key Module Descriptions

AssistantController

The AssistantController establishes and coordinates all system components, manages callbacks from wake word detection and face recognition processes, routes commands to appropriate handlers or AI services, manages conversation flow, and maintains conversation history for contextual understanding.

FaceRecognizer

The FaceRecognizer utilizes trained machine learning models for face detection and identification, triggers welcome messages for recognized users, provides visual feedback with facial identification labels, and processes frames through a queue system to ensure smooth operation.

AudioProcessor

The AudioProcessor captures continuous audio streams, detects wake word occurrences using Whisper, records and transcribes user commands, manages the various states of audio processing, and delivers audio feedback signals to enhance user experience.

CommandHandler

The CommandHandler interprets user commands through language understanding, manages reminder and task scheduling functions, executes built-in system operations,

leverages GPT-4 for understanding varied command phrasings, and maintains persistent data storage.

TextToSpeech

The TextToSpeech module converts text-based content to spoken audio using OpenAI's speech API, ensures clear voice output quality, and implements mechanisms to prevent overlapping speech output.

5. Limitations and Future Improvements

Current Limitations

Streaming Issues

The current streaming implementation presents challenges including video lag developing over extended operation periods, lack of automatic recovery mechanisms for connection failures, high bandwidth consumption for continuous streaming, and performance degradation during concurrent multi-function operation.

User Experience Issues

The interaction model requires refinement in several areas: limited recovery options exist for misunderstood commands, user interruption during processing is not supported, and face recognition performance depends on optimal lighting and direct viewing angles.

Design Limitations

The architectural design presents structural constraints including primary support for single-user interaction, inability to differentiate between voices, limited inter-conversation memory persistence, Windows-specific code dependencies, basic error handling implementation, and absence of component failure recovery mechanisms.

Security Weaknesses

The prototype lacks comprehensive security measures: command authentication is not implemented, network traffic remains unencrypted, sensitive functions lack protection mechanisms, data storage uses plain text formats, and command source verification is absent.

Future Improvements

Enhanced Local Processing

Migrating processing functions to the local device would address several limitations through: implementation of local wake word detection on the Raspberry Pi, deployment of on-device speech recognition for basic commands, execution of compact AI models directly on the device, local video processing to reduce streaming requirements, and balanced distribution between local and cloud processing.

Advanced Multi-Input Interaction

Improving multi-modal input integration would enhance system capabilities through: voice identification for user differentiation, directional audio processing to identify speakers, gesture recognition implementation, visual feedback mechanisms, and support for commands combining voice and gestural inputs.

Improved Conversation Capabilities

Enhancing conversation naturalness would require: support for follow-up queries without wake word repetition, contextual memory of conversation history, user interruption and correction capabilities, clarification requests for ambiguous commands, and personality adaptation to different users.

Technical Enhancements

Addressing technical limitations would involve: implementation of improved streaming protocols, video lag prevention mechanisms, cross-platform compatibility, simplified installation processes, and redundant systems for operational reliability.

Expanded Feature Set

Connecting with real-world systems would enable: integration with actual smart home platforms, calendar system integration, multi-room support with networked devices, and extensible skill development capability.

Concluding Statement

This Smart AI Home Assistant demonstrates the successful integration of streaming technology, computer vision, speech processing, and artificial intelligence in a home device context. While the current implementation exhibits certain limitations, its architectural design provides a robust foundation for future enhancements. The combination of local processing and cloud-based AI capabilities, particularly the RAG approach for command interpretation, indicates promising directions for developing increasingly intelligent, responsive, and privacy-conscious home assistant systems.