MEMORIAL UNIVERSITY

FACULTY OF ENGINEERING & APPLIED SCIENCE

ECE 8000/10: Engineering Design Project II

Final Report

April 8th, 2026

Supervisor & Client: Jonathan Anderson

Russell Corbett	**Jadzia Penney**	**Matthew Jones**	**Aaron Oates**
202143632	201909587	202118204	202105417
rjcorbett@mun.ca	jalexandriap@mun.ca	matthewj@mun.ca	ajoates@mun.ca

Contents

1. Introduction 5
2. Problem Definition and Analysis 5
    2.1. Requirements 5
    2.2. Constraints 5
    2.3. Design/Selection Criteria 6
3. Design Alternatives 7
4. Design 9
    4.1. Mechanical 9
    4.2. Electrical 10
    4.3. Software 12
        4.3.1. Voice 13
        4.3.2. Sensors 15
        4.3.3. Music 15
        4.3.4. Sibling Connection 15
        4.3.5. Web Backend 16
        4.3.6. Web Frontend 16
        4.3.7. Mobile App 17
            4.3.7.1. Android App 17
            4.3.7.2. iOS App 17
        4.3.8. Home Assistant Connection 18
        4.3.9. MQTT Connection 18
5. Design Analysis 19
    5.1. Meeting Design Requirements 19
    5.2. Meeting Design Criteria 20
    5.3. Project Evaluation 21
6. Impact Analysis 21
    6.1. Environmental Impact 21
    6.2. Societal and Individual Impact 22
    6.3. Safety Concerns 22
        6.3.1. Physical Safety Risks 22
        6.3.2. Home Automation Safety 22
        6.3.3. Cybersecurity Vulnerabilities 23
    6.4. Mitigation Strategies 23
7. Conclusions 23
    7.1. Future Work 24
8. References 25
9. Appendix 27
A Bill of Materials 28
B System Overview 29
C Service Lifecycle 30

D Threading Model 31
E Voice Pipeline 32
F Data Flow 33
G External Integrations 34

List of Figures

Figure 1 Satillite1 HAT [1] 7
Figure 2 Raspberry Pi Compute Module [2] 7
Figure 3 Complete RoomRelay 9
Figure 4 RoomRelay Enclosure CAD Model - Displayed in Bambu Labs 10
Figure 5 RoomRelay Custom PCB Schematic 11
Figure 6 RoomRelay Custom PCB Layout 11
Figure 7 RoomRelay Sensor Suite Board 12
Figure 8 Voice Pipeline Flowchart 14
Figure 9 Web Frontend Interface 17
Figure 10 RoomRelay iOS Mobile App 18
Figure 11 RoomRelay data sent to Home Assistant via MQTT 19
Figure 12 System Overview [3] 29
Figure 13 Service Lifecycle [3] 30
Figure 14 Threading Model [3] 31
Figure 15 Voice Pipeline [3] 32
Figure 16 Data Flow [3] 33
Figure 17 External Integrations [3] 34

List of Tables

Table 1 Design Criteria Table 6
Table 2 Sensors Alternatives Comparison Table 8
Table 3 Compute Method Alternatives Comparison Table 8
Table 4 BOM Table for a RoomRelay Device 28

1. Introduction

RoomRelay is a privacy-focused smart home assistant designed to keep user data inside the home. As users become increasingly aware of the privacy issues surrounding current market options for smart home devices, there is a clear need for a device with user privacy at its core. The RoomRelay was designed for this use case, and comes with a sensor suite, web and app interface, and integration with Home Assistant [4] to meet user needs. There are of course trade-offs because of the loss of internet connection, but this product ensures that this choice is up to the user, giving them back autonomy in their smart home.

2. Problem Definition and Analysis

The Open Home Foundation [5] states "A home is a sanctuary. A place of refuge. A place free from danger—and free from prying eyes. A home is where you can be yourself without compromising your data, your ideals, your behavior, or your independence". SafeHome [6] explains that devices like the Amazon Echo and Google Nest use cloud servers to process commands after a wake-word is detected. Data from these conversations can be sold to third parties, particularly advertisers, as [6] states "insights derived from your conversations... can influence targeted advertising and product recommendations". Furthermore, according to survey results from the Internet Society [7], 73% of users think people who use these devices should be worried about eavesdroppers, and 63% said that the way that smart home devices collect data about people and their behaviors is "creepy". Based on this information it is clear that there is a gap in the smart home assistant market. Users want control over their data and their privacy in their own homes.

2.1. Requirements

The major requirement when it comes to designing a device that aids in solving the above problem is ensuring that the device is completely private. There can be no data sent outside of the home where it could be processed and sold to third parties. However, the device will not be useful if it cannot compete with current market options for a smart home assistant. This means that there is a requirement for basic functionalities such as the ability to control other devices in the home (such as thermostats and lights), process voice accurately and deliver timely responses, and is robust enough to reconnect to other devices on its own in the event of a power outage.

2.2. Constraints

While removing cloud processing would allow a smart home assistant to be private, it also removes certain functions that can only be achieved by connecting to cloud servers. Essentially, any live data that could not be measured by a sensor on the device would not be readable. So, questions like the weather outside, or any queries that would require an internet search could not be answered. Furthermore, one would not be able to connect the device directly to commonly used music platforms like Spotify or Apple Music, and instead

would need to download music to the device. Users would also need to manually update the device as the device would never be online for live updates.

In order to keep up with current market options, a private smart home assistant would need to be small and unobtrusive so that it does not take up too much space in the home. Furthermore to compete with other options price wise the device would need to cost around $100.

2.3. Design/Selection Criteria

Criteria	Importance	Rating
Customizability	User is able to choose their own settings and have autonomy over their device, including adding/removing sensors and configuring behavior through web and app interfaces	High
Cost Efficiency	Device should be affordable to end users compared to commercial alternatives while maintaining quality and features	Medium
Physical Durability	Device must withstand normal household use including accidental knocks, falls, and continuous operation without damage to internal components	High
Power and Redundancy	Device must operate reliably with PoE and provide extended battery backup during power outages to maintain continuous smart home functionality	High
Interaction Quality	Audio (speaker/microphone), visual (LEDs), and tactile (buttons) interfaces must provide clear, responsive feedback for effective user interaction	High
UI and Accessibility	Web and mobile interfaces must be intuitive and easy to navigate for users with varying technical skill levels	Medium
Connectivity and Communication	Device must support fast, reliable communication over LAN (wired Ethernet) when powered and Bluetooth when offline with minimal latency for queries	High
Expandability and Modularity	Hardware and software must use clear interfaces allowing future upgrades, additional sensors, or feature extensions without major redesign	Medium

Table 1: Design Criteria Table

3. Design Alternatives

Before arriving at the final design, the team considered two alternative approaches. The first, and simplest alternative would be to use a Satellite1 from FutureProofHomes [1] as in Figure 1. This is a pre-built, privacy-focused smart home assistant HAT (Hardware On Top) that could be purchased for the project. The team would then need to build the Natural Language Processing (NLP) layer. This would meet the project's requirements and completely handle any hardware design. However, there would be little room for customization, leaving the team with an entirely pre-built product that they could add very little to.

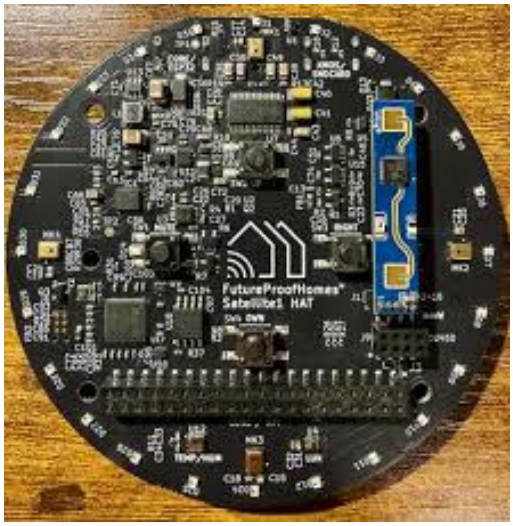


Figure 1: Satillite1 HAT [1]

A secondary alternative considered was to use a Raspberry Pi Compute Module, as seen in Figure 2, as the main component of the RoomRelay system. The Compute Module would be a component of the custom PCB, interface with all of the sensors, process voice commands and relay speech to the user. However, this would increase the complexity of the PCB greatly, as it would need to be designed to act as a carrier board for the Compute Module. This would include all input/output ports and power management. So, while the RoomRelay would have been smaller and more custom, this extra complexity was unnecessary for building a prototype to test the device's overall functionality.

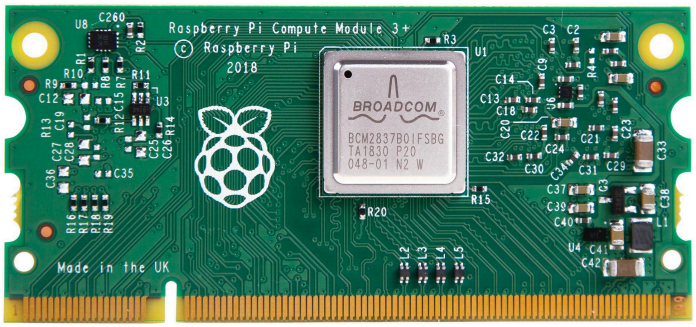


Figure 2: Raspberry Pi Compute Module [2]

For clarity, the RoomRelay design that was chosen, and alternative approaches were evaluated in Table 2 and Table 3.

	Custom Sensor PCB	**Satellite1 HAT**
Size	Long, 43.5mm by 150.6mm	88mm by 88mm [8]
Customizability	Completely custom	Completely pre-built [8]
Environmental Sensing	Temperature, humidity, luminosity, air quality, motion, and presence	"room temperature, humidity, luminosity, and human presence (when paired with a mmWave radar sensor)"[8]

Table 2: Sensors Alternatives Comparison Table

Criterion	**Raspberry Pi 5 [9]**	**Pi Compute Module [10]**
Form Factor	Standard SBC 85.6 × 56mm	Compact module 55 × 40mm
Power Consumption	12W peak (8-10W typical)	6.5W max (more efficient)
I/O Connectivity	40-pin GPIO 4× USB, Ethernet 2× HDMI	Requires custom carrier board for all I/O
Storage	MicroSD card (user-replaceable)	Onboard eMMC or external
Processing Power	BCM2712 Quad Cortex-A76 2.4GHz	BCM2712 Quad Cortex-A76 2.4GHz
Prototype Cost	$244.95 (8GB board) [11]	~$273 (8GB ram, 32GB storage) [12]

Table 3: Compute Method Alternatives Comparison Table

Please note that although the current (April 7, 2026) purchasing price of a Raspberry Pi 5 8GB is CAD $244.95, the team purchased theirs in January 2026 for CAD $180.77.

4. Design

RoomRelay is a privacy-first smart home hub built on the Raspberry Pi 5 platform, designed to provide comprehensive room monitoring and voice control without relying on cloud services. The system integrates environmental sensing (temperature, humidity, air quality, and light), occupancy detection (PIR motion and mmWave radar), and fully offline voice processing through OpenWakeWord [13], Whisper ASR [14], custom natural language understanding, and Piper text-to-speech [15]. A custom sensor PCB consolidates environmental data collection while visual feedback is provided through RGB status LEDs and tactile control via physical buttons. The device connects to existing smart home ecosystems through Home Assistant [4] integration and includes backup power capabilities via a UPS HAT with quad 18650 batteries, ensuring continuous operation during power interruptions. Network connectivity is achieved through Power over Ethernet (PoE), simplifying deployment while the custom 3D-printed enclosure houses all components in a compact, room-appropriate form factor. A companion mobile application provides configuration, monitoring, and control capabilities, creating a complete ecosystem for privacy-conscious smart home automation. These connections are shown in Appendix B.

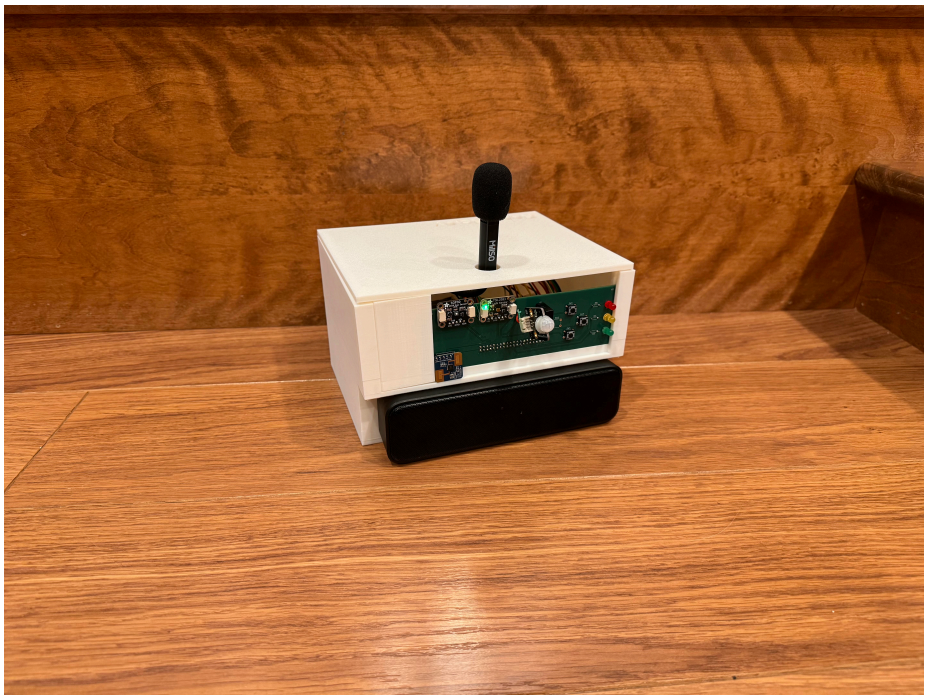


Figure 3: Complete RoomRelay

As seen in Figure 3, the RoomRelay enclosure exposes the speaker, sensor suite, microphone, and PoE input. It has air flow holes on top for heat management.

4.1. Mechanical

The final mechanical design of a RoomRelay device included a Raspberry Pi 5, UPS Module, 4 Batteries, PoE Module, a fan, mic, speaker, a custom PCB sensor suite, and a 3D printed

enclosure. The enclosure of the device, as shown in Figure 4, was designed by Ash Peddle (employee at MUN Student Design Hub) and printed by the team.

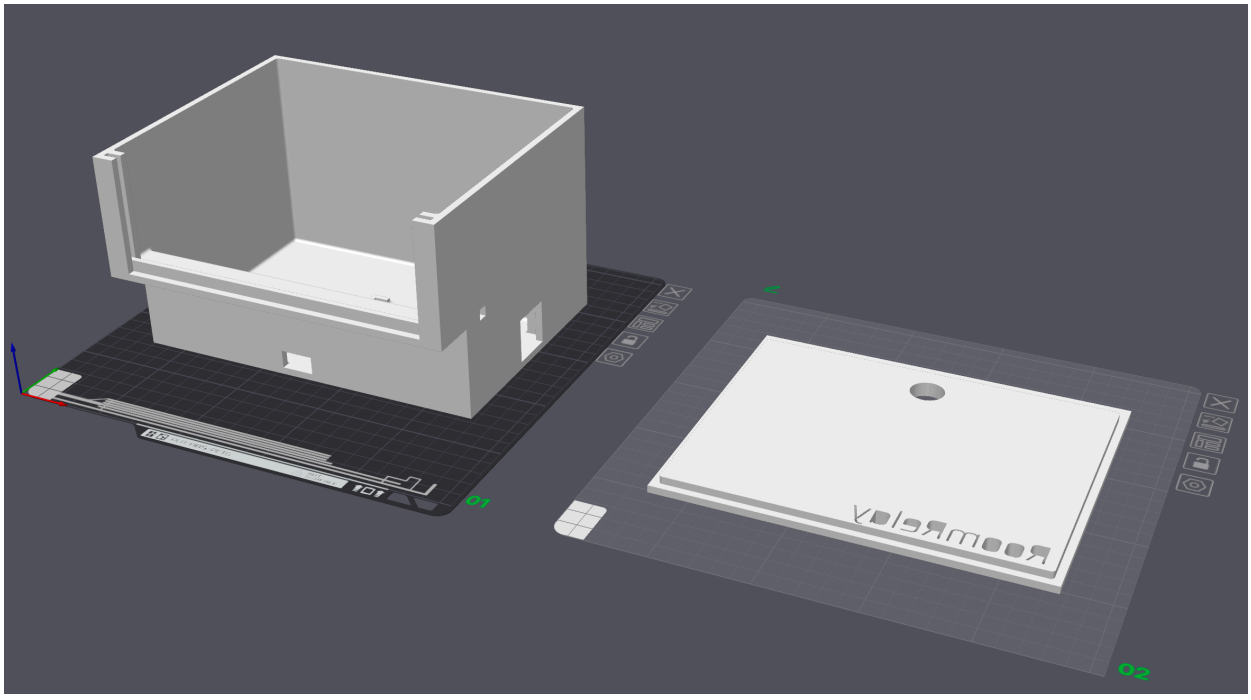


Figure 4: RoomRelay Enclosure CAD Model - Displayed in Bambu Labs

The device measures 20cm (W) x 12.75cm (H) x 16.5cm (D), and weighs 1.2kg (unplugged).

4.2. Electrical

The RoomRelay electrical system contains both off-the-shelf components and a custom PCB sensor suite. The device contains an off-the-shelf PoE injector so that the device can be powered from a single Ethernet cable, keeping installation simple for the average user. This connection provides power over PoE+ (further referred to in the report as simply PoE) delivering up to 30W to the UPS HAT (which uses 5V and up to 5A, therefore can draw up to 25W), which powers the Raspberry Pi 5, and the Pi provides power to all of the other components. Through live testing the device draws 12-15W approximately on average, and can draw up to 25W when recharging the batteries after a power outage. The team left the device running for long periods (the longest stint with no power interruption caused by user interference was one week) and there were no observed brown out events due to current draw from the device. Each RoomRelay module has a speaker and microphone that connects to the Pi via USB for power and data. The UPS HAT can power the system for up to 9 hours in the event of a power loss (determined through live testing).

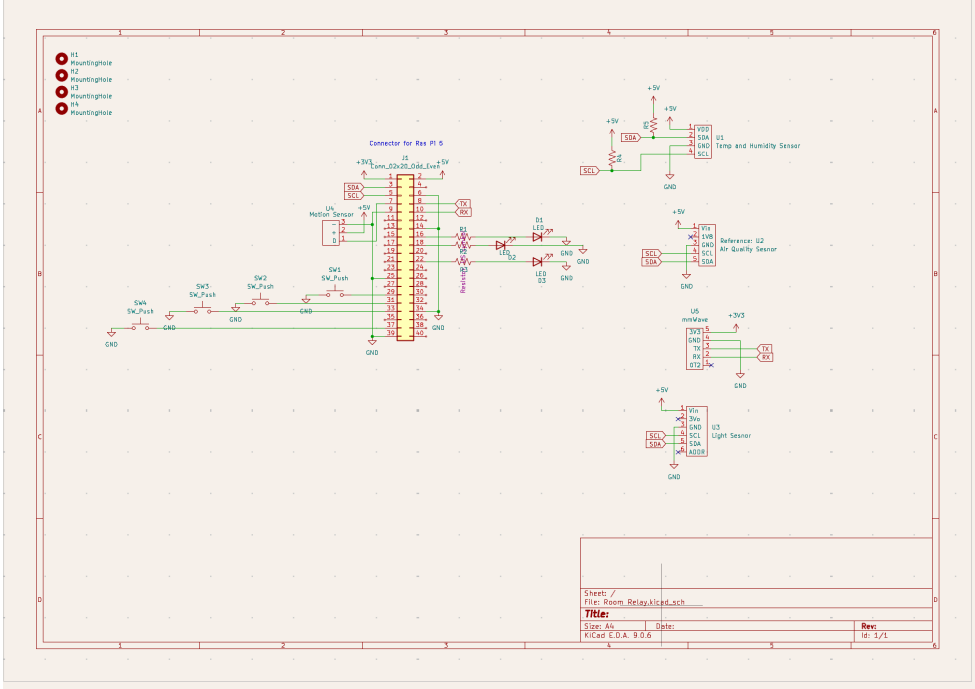


Figure 5: RoomRelay Custom PCB Schematic

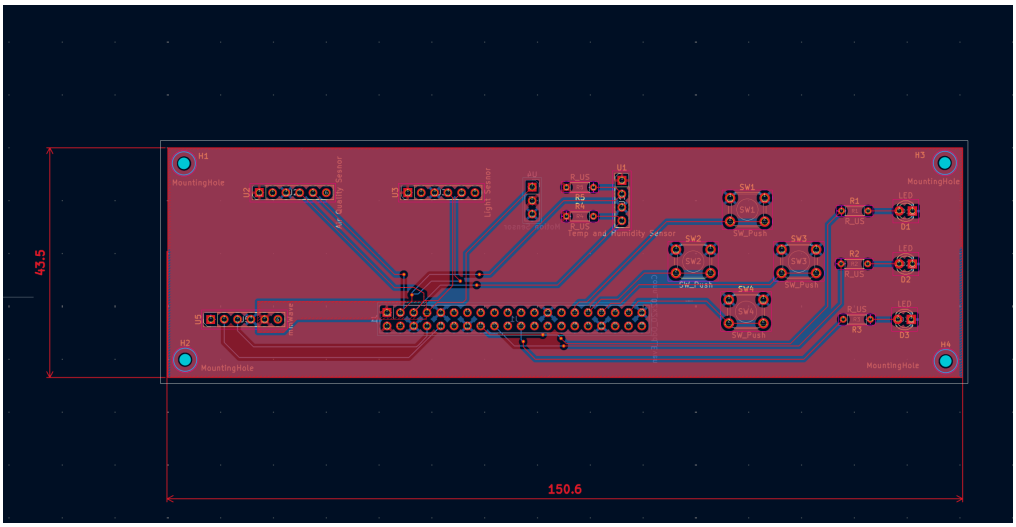


Figure 6: RoomRelay Custom PCB Layout

The sensor suite is custom PCB designed for the RoomRelay as seen in Figure 7. It was designed using KiCAD, an open-source PCB design tool. The schematic and layout can be seen in Figure 5 and Figure 6 respectively It contains all of the sensors used for environmental monitoring of the inside of a home, buttons, and status LEDs. The buttons are used for pause/play, mute, and volume control of the device without the need to access the web or app interface. The red LED turns on when the device is set to mute (when the mic is not listening for the wake word). The Yellow LED turns on if the device has switched to battery back-up. The Green LED turns on to indicate that the device is powered on. There are five environmental sensors attached to the sensor suite:

- The Adafruit SGP30 TVOC/eCO2 Gas Sensor [16]
- The Adafruit TSL2591 High Dynamic Range Digital Light Sensor [17]
- The DFRobot PIR Motion Sensor [18]
- The Waveshare Human Micro-Motion Detection Sensor [19]
- The SparkFun Electronics Humidity and Temperature Sensor [20]

The majority of the above sensors communicate with the Pi over I2C, except for the mmWave sensor which is over UART, and PiR motion is digital high/low. Each sensor delivers environmental information about the home; the quality of the air, the temperature, the humidity, the light levels, and whether there is presence detected in each room of the home that the RoomRelay module is placed in.

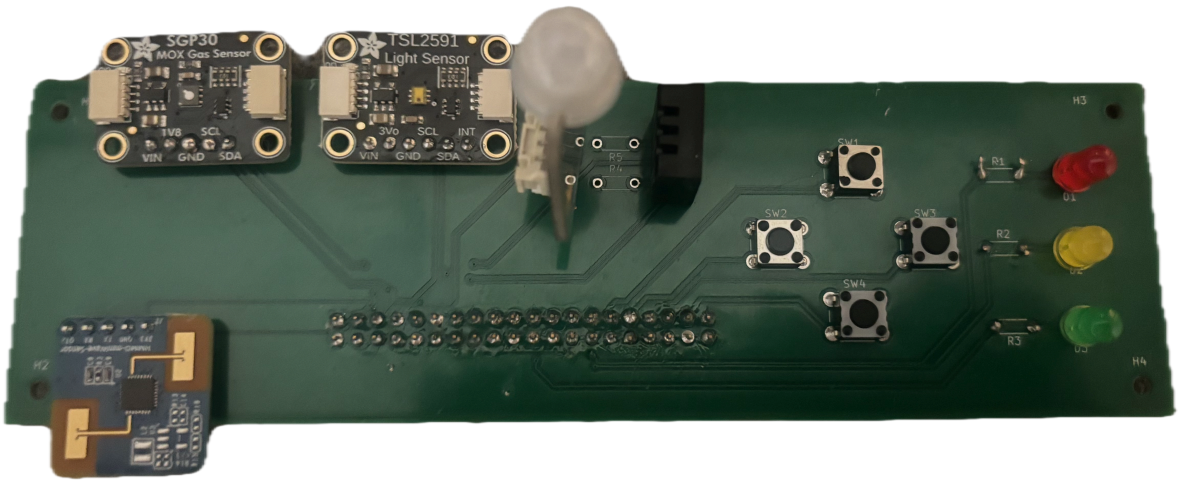


Figure 7: RoomRelay Sensor Suite Board

4.3. Software

The software of a RoomRelay is managed by a supervisor and split into many threads sharing a common configuration and context. The supervisor ensures that all the threads stay running and that the breaking of one thread does not interrupt another. The supervisor is started through the main RoomRelay script, executing manually or through a systemd script. The current threads ran and managed by the supervisor include voice, sensors, music, sibling connection, web backend and frontend, mobile app connection, Home Assistant [4] connection, and MQTT [21] connection. Each of these threads bring the concept of the RoomRelay together and demonstrate the final project. The startup order is shown in Appendix C, and threading model in Appendix D.

4.3.1. Voice

The voice thread manages the full voice interaction pipeline for the RoomRelay device, running continuously in its own thread inside the `VoiceAssistant` class. Audio is captured from a USB microphone using PyAudio [22] in 16-bit mono PCM format, sampled at 16 kHz in 1280-sample chunks. On first run, the device automatically detects the best available input device, preferring USB microphones, and saves the chosen device index and sample rate to the settings database so it is reused on subsequent boots.

Wake word detection is handled by OpenWakeWord [13], using the pre-trained `hey_jarvis_v0.1` ONNX model alongside shared melspectrogram and embedding ONNX models. Every audio chunk captured from the microphone is passed through the OpenWakeWord model in real time. When the model's confidence score exceeds the configurable `WAKEWORD_THRESHOLD` setting, the pipeline transitions from listening to recording. If music is currently playing, the volume is immediately ducked to 10% so the microphone can capture speech cleanly.

Once a wake word is detected, the device plays a short audio chime and begins recording the user's command. Recording continues until a period of silence is detected, specifically, until the RMS amplitude of incoming audio has stayed below the `SILENCE_THRESHOLD` for a duration of `SILENCE_DURATION` seconds, subject to a minimum recording length of `MIN_RECORD_DURATION` seconds and a hard cap of `MAX_RECORD_DURATION` seconds, all of which are configurable through the settings database.

The recorded audio is then transcribed by faster-whisper, a re-implementation of OpenAI's Whisper model [14] optimized for CPU inference. On Linux (the production Raspberry Pi environment), the `tiny` model is used with `float32` compute to minimize memory and latency; on macOS (the development environment) the `small` model is used for improved accuracy. If Whisper returns a transcript with a language confidence probability below 60%, the result is discarded and the pipeline resets without generating a response, preventing spurious activations from background noise.

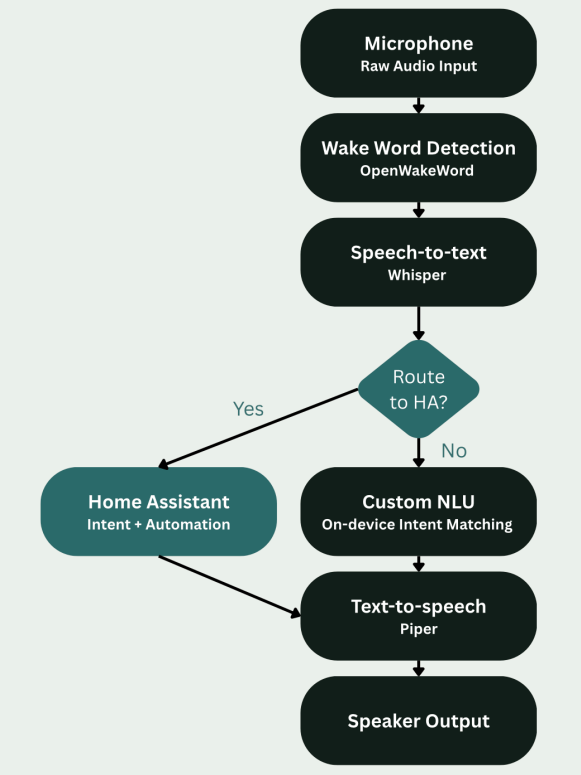


Figure 8: Voice Pipeline Flowchart

The transcript is passed to the `NaturalLanguageProcessor` (NLP), which uses regex pattern matching to route the command to the appropriate handler. Supported command categories include: sensor queries (temperature, humidity, eCO_2, TVOC, light level, motion, battery charge and voltage, power source); system queries (CPU temperature, CPU load, RAM usage, disk space, current time, current date); music control (play, pause, resume, and set or adjust volume); cross-room sensor queries that forward a request to a named sibling device over the network (e.g. "what's the temperature in the kitchen?"); device-to-device broadcasts that send a spoken message to all connected sibling RoomRelay devices; timer setting; and a Home Assistant passthrough (as shown in Figure 8, and further in depth in Appendix E) that forwards any command prefixed with "ask Home Assistant to" directly to the Home Assistant Assist pipeline over WebSocket, returning the spoken response from Home Assistant. A small set of easter-egg responses are also included for commands such as "marco", "tell me a joke", "what does the fox say", and "what is the meaning of life".

The NLP returns a response string and an optional action code. The response text is synthetised to speech using Piper [15], a fast local neural TTS engine, with the `en_US-lessac-medium` ONNX voice model producing natural-sounding audio without any network dependency. The synthetised audio is written to a `response.wav` file and played back through the speaker via the `MusicService` (Section 4.3.3), using `pysoniq` as a fallback if the music backend is unavailable. Once playback completes, the music volume is restored to its pre-

duck level, and the full interaction, original transcript and spoken response, is logged to the `voice_logs.db` SQLite database [23] for display in the web dashboard's Voice History tab.

4.3.2. Sensors

The sensors thread is responsible for continuously polling all physical sensors attached to the RoomRelay device and storing their readings in a local SQLite database (`sensors.db`) every five seconds. Three separate logger classes run concurrently: `LocalSensorLogger` handles all I2C and GPIO-connected sensors, `MMWaveUartLogger` reads presence data from the mmWave radar over UART, and `GroveSerialLogger` reads data from an attached Arduino over a serial connection. The I2C sensors polled by `LocalSensorLogger` include the DHT20 [24] (temperature and humidity at address 0x38), the SGP30 [25] (eCO_2 and TVOC air quality at address 0x58), the TSL2591 [26] (ambient light at address 0x29), and the Max17048 fuel gauge on the UPS HAT (battery voltage and state-of-charge at address 0x36). In addition to hardware sensors, `LocalSensorLogger` also samples system metrics including CPU temperature, CPU load percentage, and RAM usage. All readings are written to the `SensorDatabase` via a thread-safe `add_sensor_reading()` call, which then fires a registered publish callback so that new values are immediately forwarded to the MQTT publisher without any additional polling.

4.3.3. Music

The music thread manages audio playback on the RoomRelay device through the `MusicService` class. Playback is handled by spawning an external process, with the service automatically selecting the best available backend at runtime - trying `mpg123`, `ffmpeg`/`aplay`, `ffplay`, `cvlc`, and `mpv` in order, and falling back to `pygame` if none are found. The service exposes controls for playing a file from the configured music library directory, pausing and resuming (using `SIGSTOP`/`SIGCONT` on Linux), and adjusting volume in absolute or relative terms via ALSA's amixer. A key integration with the voice pipeline is volume ducking: when a wake word is detected, `MusicService.duck_volume_for_voice()` temporarily reduces playback to 10% so the user's speech can be captured clearly, and `restore_volume_after_voice()` returns it to its previous level once the response has been spoken. Music can be controlled through voice commands ("play music", "pause", "resume", "set volume to 50%"), through the GPIO volume buttons on the front panel, or through the web dashboard.

4.3.4. Sibling Connection

RoomRelay devices connect to each other through IP when online, and through Bluetooth when offline. This allows them to share information and broadcast across rooms. When RoomRelay devices are initially paired (currently pairing is only available over IP), they will share their unique identifiers, as well as their Bluetooth addresses. They will then initialize a Bluetooth pairing sequence between each other. The supervisor monitors the state of the battery and network connectivity to determine which transport to use. When a sibling is contacted, the `SiblingRegistry` first attempts to reach it over LAN by sending an HTTP request to its known IP address and API port. If that probe fails, for example because the device is on a different network segment or has no internet, it falls back to Bluetooth

RFCOMM using the paired Bluetooth address stored during the initial pairing handshake. The `SIBLINGS_FORCE_BT` setting can be used to skip the LAN probe entirely and always communicate over Bluetooth. Each delivery attempt is logged to the `sibling_broadcast_deliveries` database table with a `transport` field recording whether the message was delivered over LAN or Bluetooth.

4.3.5. Web Backend

The web backend is a FastAPI [27] application served by a Uvicorn ASGI server on port 8000. It exposes a RESTful HTTP API and a set of WebSocket endpoints that together power the web dashboard and allow external clients to read and configure the device. All endpoints except a small set of public paths (`/health`, `/auth/status`, `/siblings/handshake`, and `/siblings/announce`) require a Bearer token in the `Authorization` header, which is validated against the `API_TOKEN` stored in the settings database. WebSocket connections authenticate via a `?token=` query parameter. The API is organized into logical groups covering settings management, sensor history queries, live sensor streaming, voice log retrieval, music playback control, sibling device management, and Home Assistant integration. Real-time data, including sensor snapshots, music player state, voice events, and sibling broadcast history, is pushed to connected browsers over persistent WebSocket connections, avoiding the need for client-side polling.

4.3.6. Web Frontend

The web frontend (as shown in Figure 9) is hosted on the RoomRelay device and is written in plain HTML, CSS, and JavaScript, to minimize packages used and overall complexity. It is served as static files by a lightweight Python HTTP server on port 8080, separate from the API server. The single-page application is organized into five tabs: Overview, Siblings, Settings, Sensors, and Voice History, each loading its data from the FastAPI backend on port 8000 using either REST calls or WebSocket connections. The Overview tab displays a live grid of all sensor readings, a list of nearby sibling devices, a broadcast message field, and a music player control panel. The Sensors tab uses Chart.js to render time-series graphs of any sensor type with configurable time ranges and bucketing intervals. The Settings tab renders an editable form for all device configuration keys, grouped by user-facing, advanced, and developer categories. Authentication is handled by storing the API token in the browser's local storage and attaching it as a Bearer header on all requests.

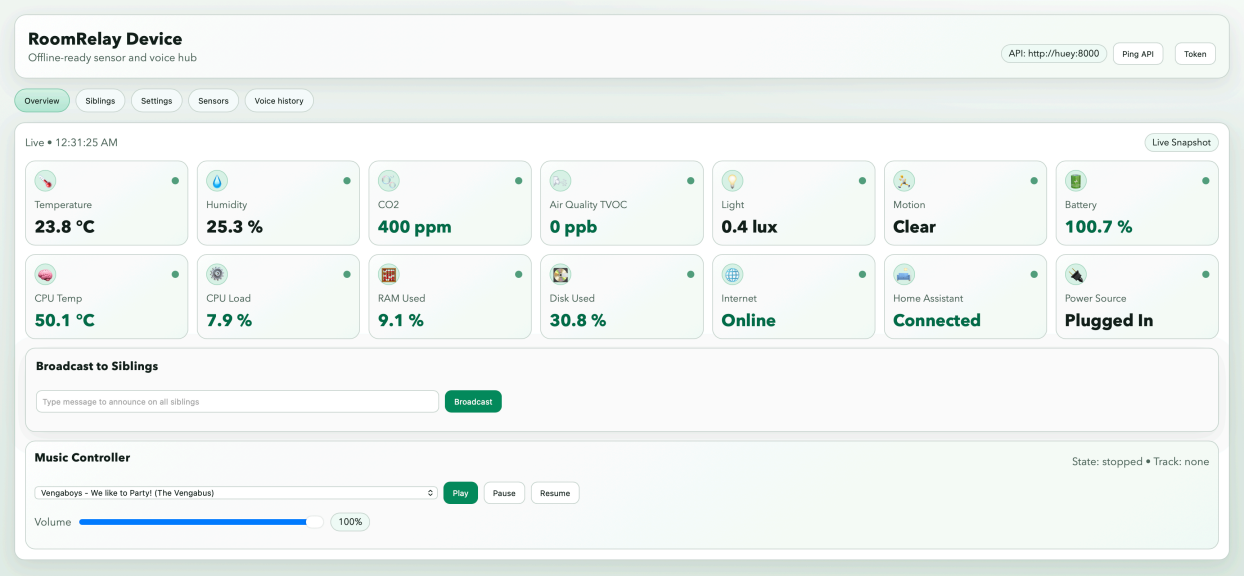


Figure 9: Web Frontend Interface

4.3.7. Mobile App

There are two versions of the RoomRelay mobile App: one for iOS and one for Android.

4.3.7.1. Android App

The Android app is the simpler of the two and was designed for rapid prototyping of Bluetooth connection. It has a homepage that connects to 4 additional pages: scan for devices, sensor reading, voice logs, and a quick view dashboard. The app uses FlutterBluePlus [28], the "Bluetooth Low Energy (BLE) plugin for Flutter" to connect to the RoomRelay device.

- **Scan For Devices Page:** Shows nearby BLE devices able to be paired with, including RoomRelay devices.
- **Sensor Readings Screen:** Shows live updates for the sensor readings on this page.
- **Voice Logs Screen:** Shows a record of all voice logs that have been processed on the RoomRelay.
- **Dashboards Screen:** Shows the user historical data from sensor readings.

4.3.7.2. iOS App

The iOS app is more feature-complete than the Android version and was developed to explore a greater number of the project's stretch goals. It connects to a RoomRelay device over the local Wi-Fi network using the device's REST API and WebSocket endpoints, rather than relying solely on BLE. This allows it to access the full range of data exposed by the web backend, including live sensor readings, voice command history, sibling device status, and playback controls. The app communicates with the device using the `mqtt_client` Dart package, enabling it to receive real-time sensor updates via the MQTT broker that the

RoomRelay publishes to. The settings screen allows users to configure the device IP address and API token directly from the app.

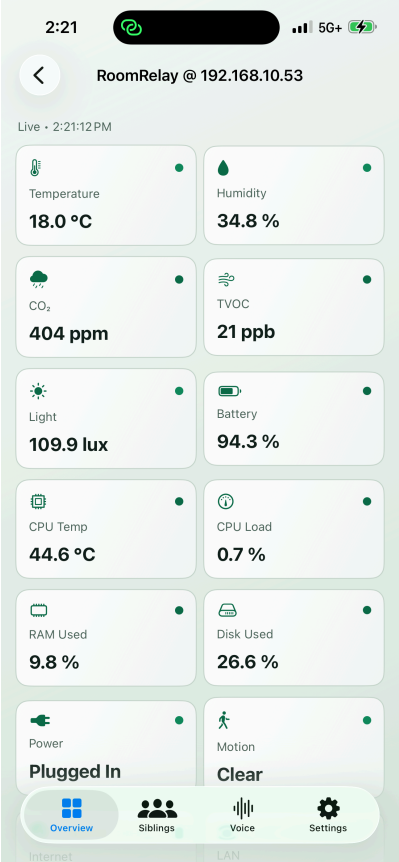


Figure 10: RoomRelay iOS Mobile App

4.3.8. Home Assistant Connection

The Home Assistant integration is structured as a managed WebSocket client paired with a supervising manager that maintains connectivity and configuration. The `HomeAssistantAssistWS` class is responsible for establishing and authenticating a WebSocket connection to the Home Assistant API, sending conversational text requests, and maintaining session state such as message IDs and conversational context. It also includes basic fault tolerance by attempting to reconnect if the connection is lost. The `HomeAssistantManager` acts as a supervisor, running its own thread and continuously monitoring whether the integration is enabled in through the settings database. It handles creating and maintaining the client, ensuring thread-safe access, automatically reconnecting on failure, and periodically pinging the server to keep the connection alive.

4.3.9. MQTT Connection

The MQTT connection is managed by the `MqttPublisher` class using the `paho-mqtt` client library. On startup, it connects to the configured MQTT broker using the `MQTT_HOST`, `MQTT_PORT`, `MQTT_USERNAME`, and `MQTT_PASSWORD` settings, and immediately publishes an online

availability message to `roomrelay/{node_id}/status` with a last-will message of offline so that any subscriber, including Home Assistant, can detect if the device goes offline unexpectedly. Each time a sensor reading is stored in the database, the registered publish callback sends the value to `roomrelay/{node_id}/sensors/{sensor_type}` as a retained message. On the first publish for any given sensor type, `MqttPublisher` also sends a Home Assistant MQTT Discovery configuration payload to the `homeassistant/` prefix, which causes the sensor to appear automatically in Home Assistant's entity registry with the correct unit, device class, and icon, requiring no manual configuration in Home Assistant. The publisher also subscribes to `roomrelay/{node_id}/set/mic_mute`, allowing Home Assistant automations or other MQTT clients to remotely mute or unmute the device's microphone.

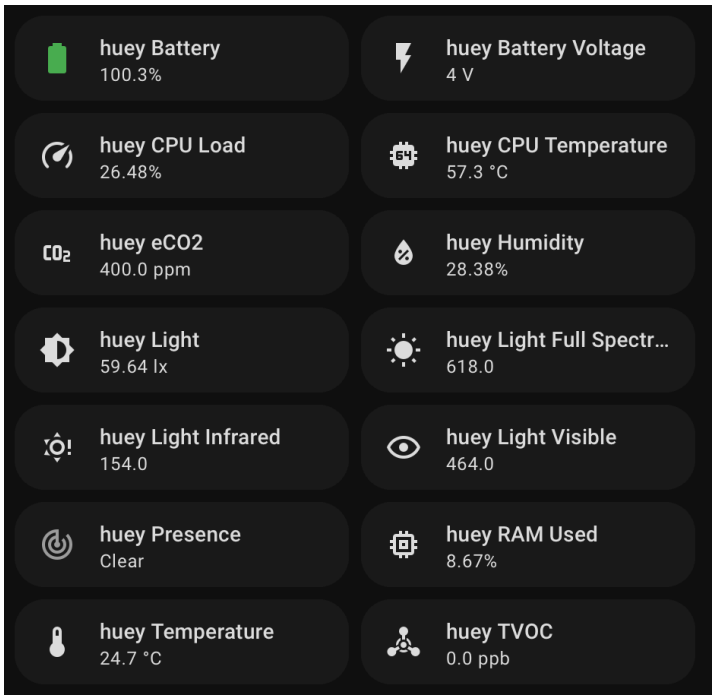


Figure 11: RoomRelay data sent to Home Assistant via MQTT

5. Design Analysis

5.1. Meeting Design Requirements

The design outlined in the above sections meets and goes beyond the requirements outlined in Section 2.1. The main requirement for the device was that it be completely private. All voice processing occurs on the device quickly and accurately, and at no point does any data leave the home network. It can quickly reconnect to other devices when they are turned on after losing power. Through integration with Home Assistant [4], the RoomRelay can perform basic tasks expected of a smart home assistant, such as turning on lights or adjusting the temperature.

The RoomRelay also includes features that are not commonly found in other smart home assistants. There is a battery backup that will keep the device on during a power outage.

Furthermore, while on battery backup, the RoomRelay can connect to other RoomRelay modules over Bluetooth, allowing the user to receive updates about the home. The user can also download music directly to the device, so even without power, they can still use the RoomRelay as a smart speaker. The included sensor suite can provide data about the home that other smart home assistants cannot read. The web and app interfaces further allow the user not only to control the device but also to have important information about the home at their fingertips, giving them autonomy over their privacy and their home environment.

5.2. Meeting Design Criteria

The team attempted to meet all design criteria, dedicating the most time to those with a high priority in Section 2.3. This put customizability, physical durability, power/redundancy, interaction quality and connectivity/communication at the forefront of the RoomRelay design. The device has many customizable features that the user can toggle on or off. For example, they can customize the microphone's sensitivity based on whether the RoomRelay is often in a loud environment. They can download whatever music they would like to the device. Furthermore, by integrating with Home Assistant [4], they can choose which devices the RoomRelay can control in their home. The device is robust enough to withstand being knocked over without too much damage to any components, though, because of the nature of the current prototype, some connections may need to be remade inside the device, such as the jumper cables. The device is powered entirely by PoE, with battery backup in case of a power outage. The power switch is seamless, and the batteries begin charging as soon as the device receives PoE after switching. The device can respond to the wake word and quickly reply, making for high-quality interactions. Furthermore, the buttons are fully integrated and work effectively for control without speech. The LEDs are easily visible, allowing users to see the status of their RoomRelay device. The device has a reliable connection to other RoomRelays and smart home devices. When power is disconnected, the multiple RoomRelay devices can still maintain a connection with each other over Bluetooth.

Most criteria with medium priority have all at least been partially met. The UI is easy to read, and there are multiple ways for the user to interact with the device, including tactile buttons, speech, and the web/app interface, making it more accessible. Furthermore, the web/app interface displays the data collected from the sensor suite in multiple ways so the user can see both the history and information about the home in the present moment. This data is presented in green, orange, or red to indicate to the user whether the reading is good, poor, or somewhere in between, without needing to understand exactly what terms like TVOC levels mean. The device is somewhat expandable and modular. It can be updated manually to easily add new software features. New sensors can be connected to the Pi, and programming can be added for them, but cannot be added to the PCB itself, since that would require a complete redesign and reprint of the PCB, which would not likely be done by the average user.

The only criteria that has not been met is cost efficiency. The current prototype for the RoomRelay is less cost-efficient than the team hoped, coming in at $482 (Appendix A) rather

than $150, the original desired cost for the device. This is not particularly affordable for the average user and should be improved upon in the future.

5.3. Project Evaluation

Overall, the team is satisfied with the prototype of RoomRelay that was created. RoomRelay met all the requirements from the problem definition and nearly met all of the design criteria. In the beginning of the project the team created a schedule that included both internal deadlines for the design and testing of the prototype as well as external deadlines for required deliverables. While some of these internal deadlines had to be moved, all goals set for this project were achieved and a working prototype was created. Furthermore, the team was able to meet some stretch goals like playing music and MQTT integration.

Though the prototype was a success there were some obstacles over the course of the design. For example, when the software for the project began, all elements of the RoomRelay were initially running in the same thread. The voice, sensors, music, and Home Assistant, were all running concurrently. This proved difficult as more features were added. It was decided then that the team would refactor the existing systems to work in a supervisor and thread system, where the supervisor would monitor all the worker threads and keep them all running independent of each other. They would share a common configuration and context to share data and messages, but other than that, if one thread were to fail, it would not interrupt others, and the supervisor would restart it. It was during this time that an important lesson was learned about development. Breaking the project into components that could run and be tested individually, then integrated later would allow for a smoother design process overall. This was taken into account across all parts of the design, each team member working individually on their component to create one prototype.

6. Impact Analysis

The deployment and use of RoomRelay as a privacy-first smart home assistant carries many implications across environmental, societal, individual, and safety dimensions.

6.1. Environmental Impact

RoomRelay's environmental footprint includes both challenges and advantages compared to commercial alternatives. Each device contains electronic components (Raspberry Pi 5, custom PCB, sensors, lithium-ion batteries, and plastic enclosure) that contribute to e-waste at end-of-life. The device consumes approximately 15-20W during operation, resulting 131-175kWh annual energy consumption. This is a meaningful addition to household electricity, up to $26 with current Newfoundland electricity rates [29].

However, the modular design provides environmental benefits. Unlike integrated commercial smart speakers, RoomRelay allows component-level repair and upgrade rather than full device replacement. Users can replace individual sensors, batteries, or modules without discarding the entire system. Additionally, by processing all voice commands locally, RoomRelay eliminates ongoing cloud computation energy demands associated with commercial assistants.

The use of PLA+ filament for the 3D-printed enclosure provides a more sustainable alternative to injection-molded plastics, as PLA is derived from renewable resources and is biodegradable under industrial composting conditions.

6.2. Societal and Individual Impact

Since smart home assistants are a relatively new technology, there is not much know about their long-term impact. However, they can aid in making homes more accessible especially for those who are vision impaired as it can control the devices in the home like lights and do things like check to make sure an oven has been turned off by integrating to other smart home devices. That being said, some aspects of the RoomRelay are not easily accessible for the average user. For example, because the device is completely offline, all updates would need to be done manually by entering through a secure shell, which the average user may not understand how to do. Furthermore, the accessibility of the device is limited, there is no screen on the device so anything that the user might want to view on a screen must be done through the web and app interface which is not currently screen readable.

6.3. Safety Concerns

6.3.1. Physical Safety Risks

RoomRelay's use of lithium-ion batteries introduces fire safety considerations. These batteries can experience thermal runaway if damaged, improperly charged, or exposed to extreme temperatures [30]. The UPS HAT includes charge management circuitry and the enclosure provides ventilation, but users must follow proper battery handling and replacement procedures.

Thermal management is critical. The Raspberry Pi 5 generates significant heat during sustained processing. The active cooling fan mitigates this risk, but fan failure could lead to component damage and potential fire hazard. The enclosure design includes ventilation holes for passive cooling backup.

6.3.2. Home Automation Safety

Integration with Home Assistant creates safety implications if the voice system is compromised or malfunctions. Unintended commands could manipulate HVAC systems (endangering vulnerable individuals or causing property damage), disable security lighting, unlock doors, or activate high-power appliances creating fire hazards.

RoomRelay's local processing architecture actually mitigates some attack vectors present in cloud-based systems. Without internet connectivity, the device is not vulnerable to remote exploitation through cloud service compromises. Attacks will require local network access. The pattern-matching natural language processing reduces risks of unintended command interpretation compared to more complex language models.

6.3.3. Cybersecurity Vulnerabilities

While offline architecture eliminates cloud-related attacks, local network security becomes critical. The web API on port 8000, if compromised through weak API tokens or network sniffing, could allow attackers to access sensor data, issue Home Assistant commands, monitor presence detection, or control the microphone. MQTT broker connections require strong authentication and TLS encryption to prevent monitoring or manipulation. Bluetooth pairing for sibling devices introduces proximity-based vulnerabilities if not properly secured.

Physical security is most important. Unlike cloud devices, physical access to RoomRelay provides complete control. An attacker could extract the SD card to access all stored data including sensor histories, voice logs, and credentials, or modify software to install backdoors. This enhances the need for encrypted storage of sensitive credentials and physical security in deployment environments.

6.4. Mitigation Strategies

Many identified impacts can be mitigated through responsible design and deployment

- **Environmental:** Provide repair guides, partner with e-waste recycling programs, optimize software for lower power consumption
- **Accessibility:** Develop comprehensive setup documentation, implement more feedback methods (visual, tactile, audio)
- **Safety:** Implement rate limiting for high-risk commands, thermal monitoring with automatic shutdown, battery health alerts
- **Security:** Generate cryptographically strong API tokens by default, enforce MQTT TLS encryption, implement intrusion detection

7. Conclusions

The team was determined to solve the problem that current smart home systems have not addressed. The team successfully made a privacy-first smart home device that processes all requests and data locally with no need for cloud services or external servers. The device met all the requirements outlined in Section 2.1 and met most of the design criteria outlined in Section 2.3. The team considered some alternative designs to help complete the project, but ultimately, the drawbacks that came with these alternatives made the team decide on other designs that could achieve the solution.

For the final design, there were mechanical, electrical and software components that had to be designed. The mechanical design consisted of the Raspberry Pi 5, UPS Module, 4 batteries, PoE Module, a fan, a custom PCB sensor suite, and a 3D-printed enclosure. For the electrical design, a PCB was designed to make the sensor suite, which included the sensors, status LEDS, and buttons for volume control, pause/play and, mute. The software design consisted of designing worker systems that are observed by a supervisor to ensure if one system has an issue, the other aspects of the software remain running. The systems designed included

voice, sensors, music, sibling connection, web backend and frontend, iOS and Android app, Home Assistant connection and MQTT connection.

For the end design, the team satisfied all the requirements they set out to accomplish as well as meet many of the design criteria during the implementation of the project, and as a result, the project was a success as the team was able to hit the objectives associated with the minimum viable product as well as some additional stretch goals. There are associated risks for the RoomRelay design but mitigation strategies have been thought about for future implementations. Overall, the product demonstrates a privacy-first smart home assistant while offering the comparable services of an Amazon Echo and Google Nest, without the need for external voice processing servers.

7.1. Future Work

Transitioning RoomRelay from prototype to product requires addressing three key areas: configurable privacy settings, hardware optimization, and expanded functionality.

- **Configurable Privacy Options:** Future versions should allow users to selectively enable cloud-based features while maintaining local-first defaults. Users could opt-in to internet weather queries, web search, or streaming music (Spotify, Apple Music) with explicit consent and visual indicators when external services are accessed. This respects a users privacy while giving them a choice for greater feature access.
- **Miniaturization and Cost Reduction:** The current prototype ($482, 20 x 12.75 x 16.5cm) requires optimization for market viability. Migrating to a Compute Module-based carrier PCB would shrink components, eliminate internal cabling, and reduce size by 50-60% (target: 12 x 8 x 10cm).
- **Enhanced Capabilities:** Additional features would improve usability: speaker identification for multi-user households, multi-language support, further development of mmWave (occupancy counting), and Matter protocol support.

8. References

[1] FutureProof Homes, "FutureProof Homes Docs." [Online]. Available: https://docs.futureproofhomes.net/satellite1-introduction/

[2] "PiShop." [Online]. Available: https://www.pishop.ca/product-category/raspberry-pi/raspberry-pi-boards/compute-modules/

[3] "Diagrams: Diagram as Code." [Online]. Available: https://diagrams.mingrammer.com/

[4] Home Assistant, "Home Assistant Homepage." [Online]. Available: https://www.home-assistant.io/

[5] Open Home Foundation, "Open Home Foundation about page." [Online]. Available: https://www.openhomefoundation.org/about/

[6] Rob Gabriele, "Do Alexa and Google Assistant Devices Compromise Your Privacy?." [Online]. Available: https://www.safehome.org/home-automation/disable-smart-speaker/#:~:text=Smart%20speakers%20like%20Amazon%20Echo%20and%20Google,and%20unauthorized%20access**%20*%20**Third%2Dparty%20data%20sharing**

[7] "Concerns Over Privacy and Security Contribute to Consumer Distrust in Connected Devices." [Online]. Available: https://www.internetsociety.org/news/press-releases/2019/concerns-over-privacy-and-security-contribute-to-consumer-distrust-in-connected-devices/

[8] FutrueProofHomes, "Satellite1 Dev Kit." [Online]. Available: https://futureproofhomes.net/products/satellite1-pcb-dev-kit?srsltid=AfmBOorRqwAJeIgMZSTLo_DRhTXQlOo6zAe-mcMq0L_13vR_UExmplwR

[9] Raspberry Pi Ltd, "Raspberry Pi 5 Product Brief." [Online]. Available: https://pip-assets.raspberrypi.com/categories/892-raspberry-pi-5/documents/RP-008348-DS-6-raspberry-pi-5-product-brief.pdf

[10] Raspberry Pi Ltd, "Raspberry Pi Compute Module 5 Datasheet." [Online]. Available: https://pip-assets.raspberrypi.com/categories/944-raspberry-pi-compute-module-5/documents/RP-008180-DS-6-cm5-datasheet.pdf

[11] PiShop.ca, "Raspberry Pi 5/8GB." [Online]. Available: https://www.pishop.ca/product/raspberry-pi-5-8gb/

[12] PiShop.ca, "Raspberry Pi Compute Module 5, Wireless, 8GB RAM, 32GB eMMC - CM5108032." [Online]. Available: https://www.pishop.ca/product/raspberry-pi-compute-module-5-wireless-8gb-ram-32gb-emmc-cm5108032/

[13] David Scripka, "openWakeWord." [Online]. Available: https://github.com/dscripka/openWakeWord

[14] OpenAI, "Whisper." [Online]. Available: https://github.com/openai/whisper

[15] Michael Hansen, “Piper: A Fast, Local Neural Text to Speech System.” [Online]. Available: https://github.com/rhasspy/piper

[16] Adafruit, “Adafruit SGP30 TVOC/eCO2 Gas Sensor.” [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-sgp30-gas-tvoc-eco2-mox-sensor.pdf

[17] Adafruit, “Adafruit TSL2591 High Dynamic Range Digital Light Sensor - STEMMA QT.” [Online]. Available: https://www.adafruit.com/product/1980?srsltid=AfmBOorzTfd6Rh_hCAXqOoovl7GdJQxf2KA8Jew4hUaP2mAeghf5Ec5x

[18] DFRobot, “PIR Motion Sensor V1.0 SKU:SEN0171.” [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/2182/SEN0171_Web.pdf

[19] PiShop, “Human Micro-Motion Detection Sensor, 24GHz mmWave Radar, S3KM1110, FMCW Technology.” [Online]. Available: https://www.pishop.ca/product/human-micro-motion-detection-sensor-24ghz-mmwave-radar-s3km1110-fmcw-technology/

[20] SparkFun, “Data Sheet DHT20.” [Online]. Available: https://cdn.sparkfun.com/assets/8/a/1/5/0/DHT20.pdf

[21] OASIS, “MQTT: The Standard for IoT Messaging.” [Online]. Available: https://mqtt.org/

[22] Hubert Pham, “PyAudio.” [Online]. Available: https://pypi.org/project/PyAudio/

[23] SQLite Consortium, “SQLite Home Page.” [Online]. Available: https://sqlite.org/index.html

[24] L. ASAIR (Aosong Electronics Co., “DHT20 Datasheet – Humidity and Temperature Module.” [Online]. Available: https://cdn.sparkfun.com/assets/8/a/1/5/0/DHT20.pdf

[25] Sensirion AG, “SGP30 – Multi-gas (VOC and CO_2eq) sensor.” [Online]. Available: https://sensirion.com/products/catalog/SGP30

[26] Adafruit Industries LLC, “TSL2591 High Dynamic Range Digital Light Sensor.” [Online]. Available: https://www.adafruit.com/product/1980

[27] Sebastián Ramírez, “FastAPI: Modern, Fast Web Framework for Building APIs.” [Online]. Available: https://fastapi.tiangolo.com/

[28] Jamcorder, “flutter_blue_plus | Flutter package.” [Online]. Available: https://pub.dev/packages/flutter_blue_plus

[29] NL Hydro, “Current Rates.” [Online]. Available: https://nlhydro.com/electicity-rates/current-rates/#:~:text=For%20customers%20on%20the%20Island,is%203.154%20cents%20per%20kWh.

[30] ENVO Drive, “Thermal Runaway in Lithium-Ion Batteries: Causes, Risks, and Prevention.” [Online]. Available: https://envodrive.com/blogs/articles/thermal-runaway-in-lithium-ion-batteries-causes-risks-and-prevention?srsltid=AfmBOopy-V6E8zRvDOwKxzc4AdPR1ynBN5jnN5F1SlhuWfdxfJi9T_JM

9. Appendix

Appendix A: Bill of Materials

Item	Source	Price (CAD)
Raspberry Pi 5 8GB	pishop.ca	$137.00
Pi UPS Hat	aliexpress.com	$71.32
Pi UPS Batteries x4	amazon.ca	$26.99
PoE Splitter	amazon.ca	$43.99
Speaker	amazon.ca	$23.98
Microphone	amazon.ca	$19.98
3D Print Enclosure *Bambu Labs PLA+ Filament*	custom	$11.01
Pi Active Cooler	amazon.ca	$15.99
Humidity and Temp Sensor	digikey.ca	$13.80
Air Quality Sensor	digikey.ca	$26.98
Light Sensor	digikey.ca	$10.03
Motion Sensor	digikey.ca	$7.55
mmWave Sensor	pishop.ca	$5.95
Resistors	provided by ECE Lab	$0.15
Buttons	digikey.ca	$0.14
Red LED	digikey.ca	$0.23
Green LED	digikey.ca	$0.79
Yellow LED	digikey.ca	$0.46
Male to Female Cables	pishop.ca	$2.95
Tax		**$62.89**
TOTAL		**$482.19**

Table 4: BOM Table for a RoomRelay Device

Appendix B: System Overview

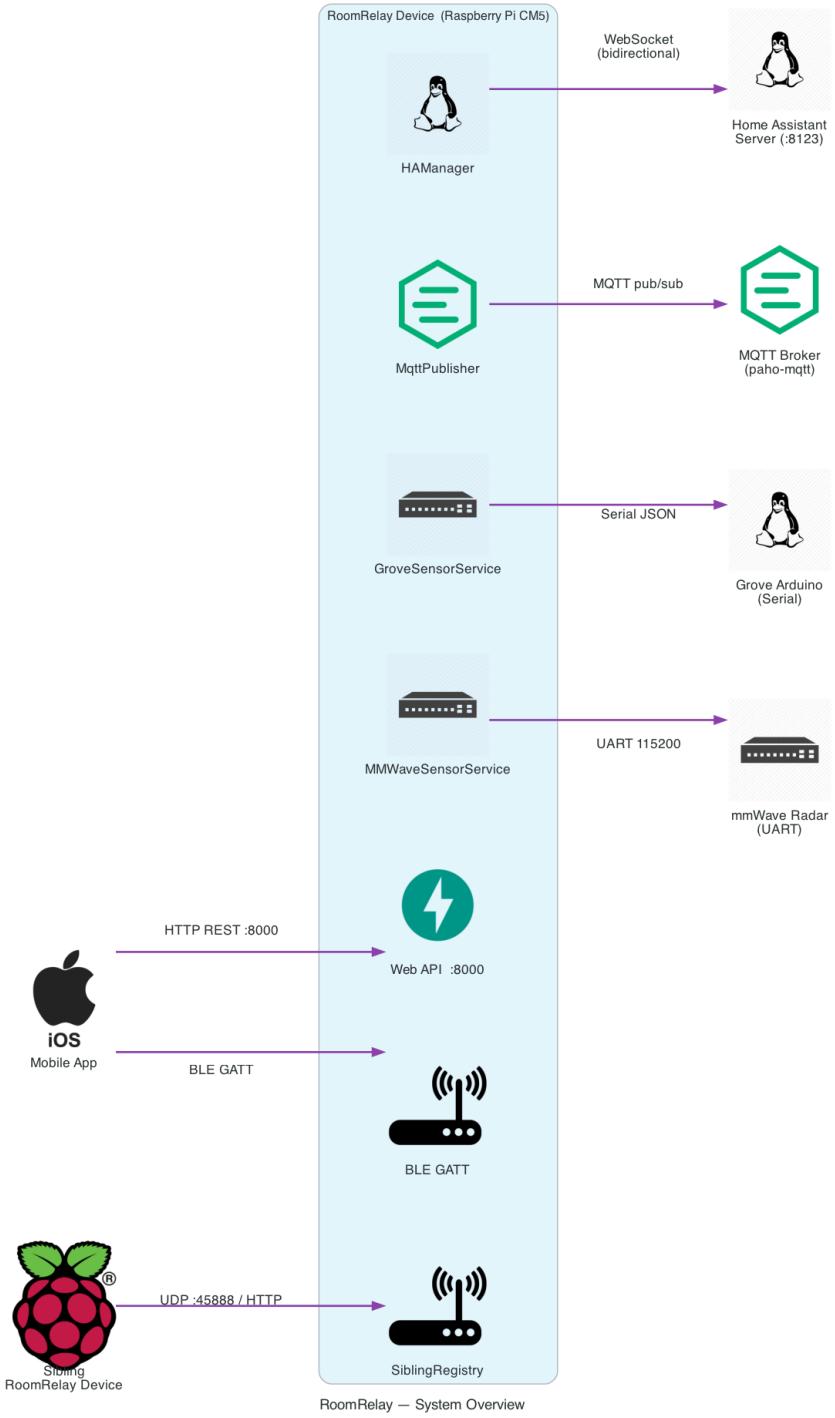


Figure 12: System Overview [3]

Appendix C: Service Lifecycle

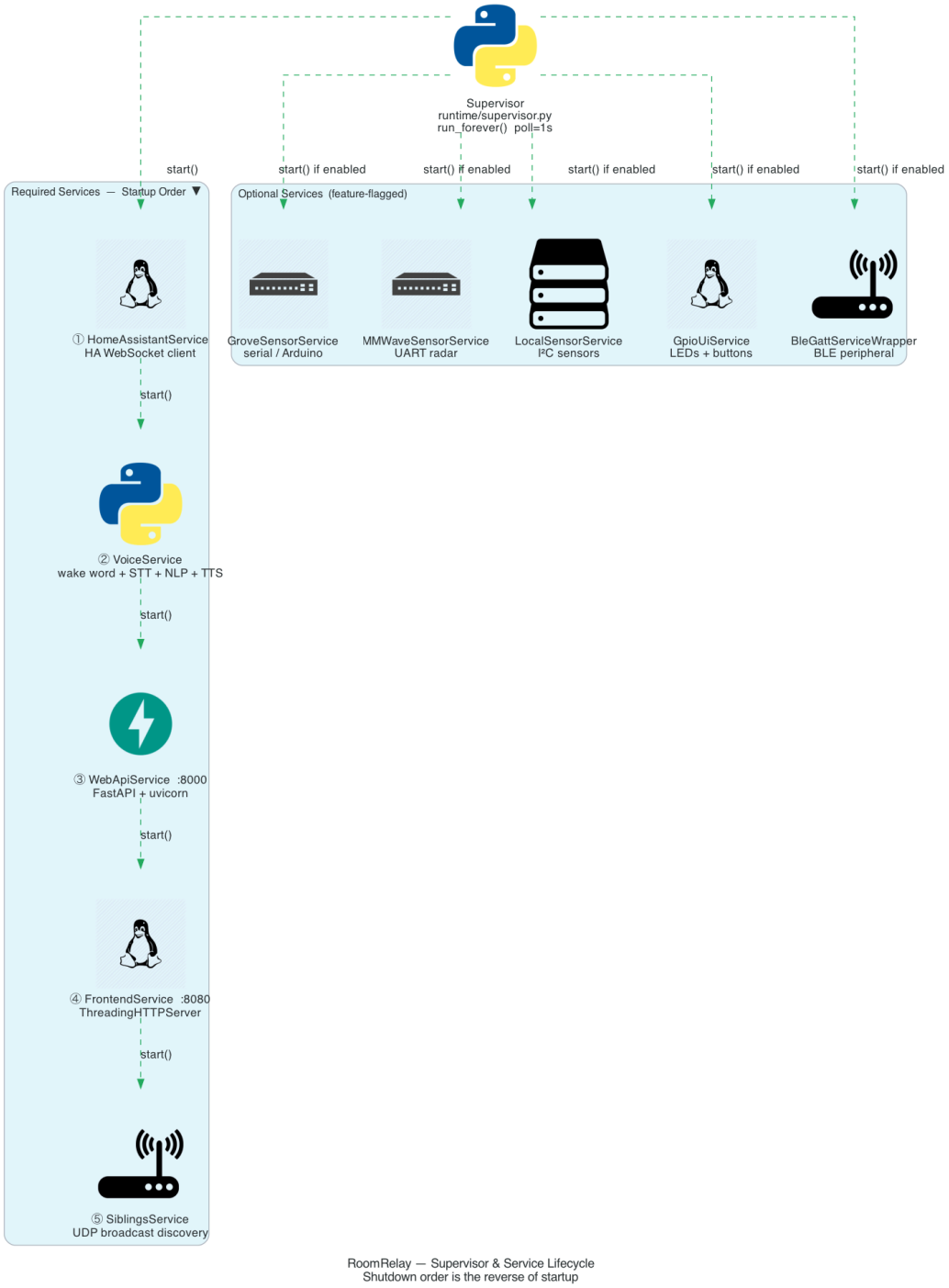


Figure 13: Service Lifecycle [3]

Appendix D: Threading Model

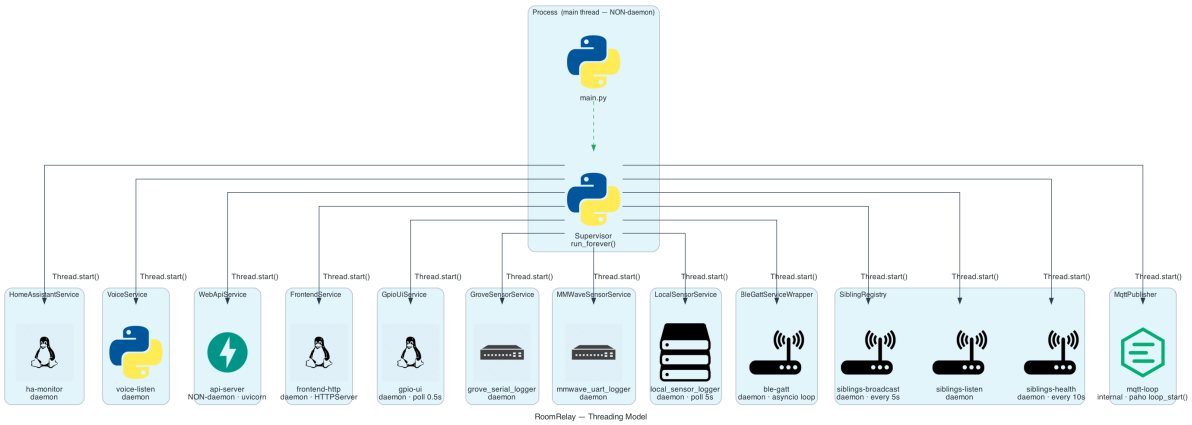


Figure 14: Threading Model [3]

Appendix E: Voice Pipeline

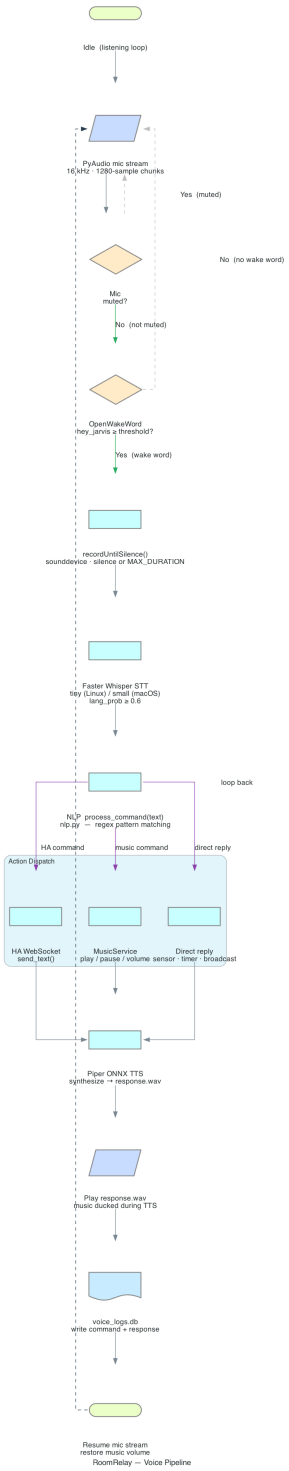


Figure 15: Voice Pipeline [3]

Appendix F: Data Flow

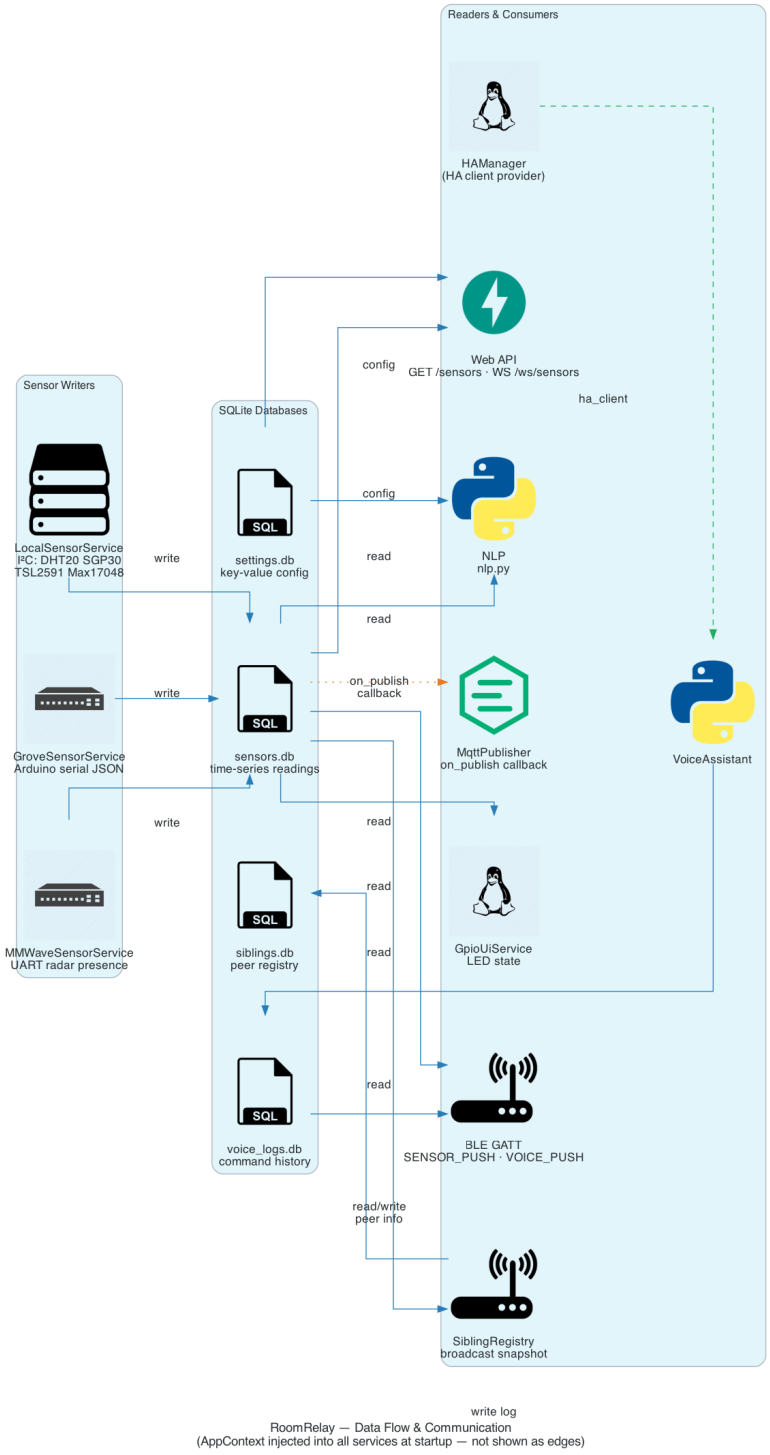


Figure 16: Data Flow [3]

Appendix G: External Integrations

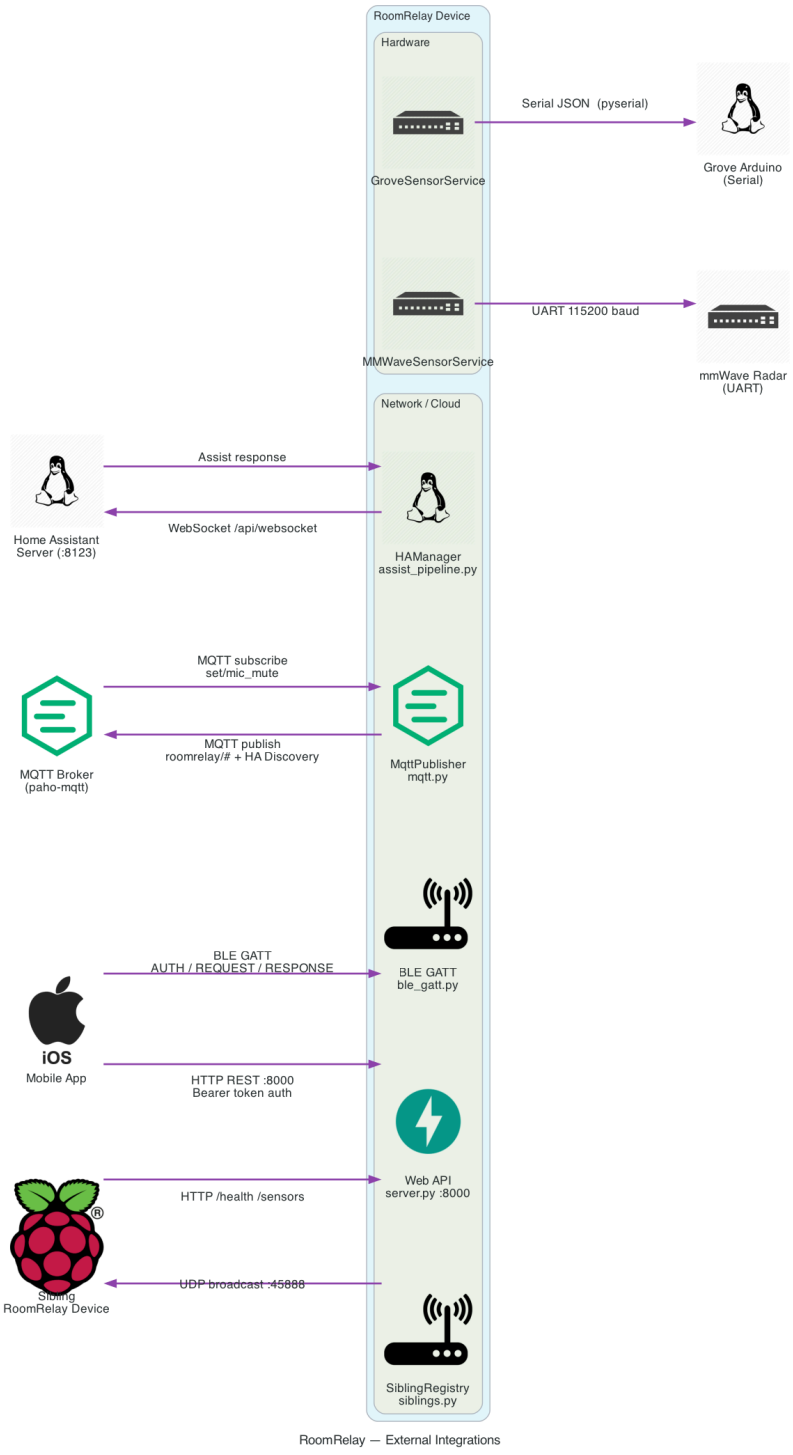


Figure 17: External Integrations [3]