ResRemind: Smart Residential Reminder IoT System

HANNAH COATES, UCSD
SUBHON GHOSH, UCSD
FARHAT AHMED, UCSD
PRANEETH SURAPANENI, UCSD

1 Introduction

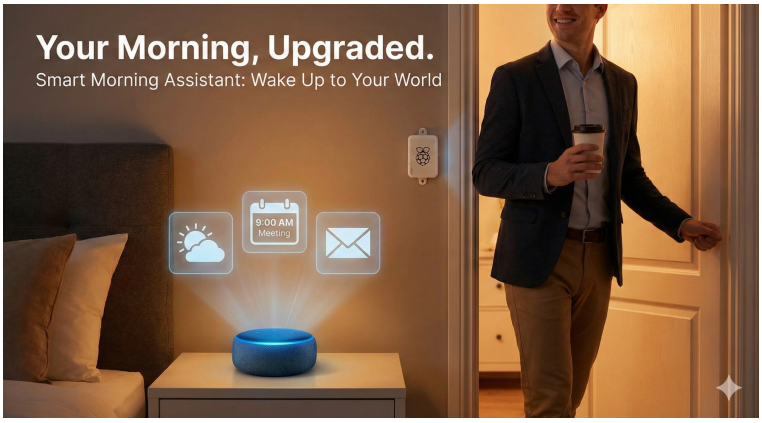

Fig. 1. ResRemind Concept

People often leave home without awareness of current weather or scheduled events, causing delays and missed appointments. Having to daily check your phone to receive these updates can be tedious and time consuming for users as they go about their morning routine. In order to track this information a user would be required to set aside additional time in their day where they focus on their phone or checking another medium to see the weather conditions for the day. People may also miss appointments when they don't check their calendar for the day. This project addresses these problems by implementing a seamless IoT experience into residential spaces.

The target population for our project could be anyone who finds they don't have time to check their calendar or the weather for the day. This could include busy students or professionals who have their schedule added to their calendars and want seamless, hands-free updates when leaving their rooms or homes.

This project will address the stated problem by providing the user with a seamless and automated experience in the morning. Utilizing both an Alexa and Raspberry Pi the project gives the user a clear morning announcement that tells the user their calendar events for the day as well as the indoor and outdoor weather conditions. In this way the user no longer needs to set aside a specific time for them to check the weather as well as their calendar. Instead as they leave

Author's Contact Information: Hannah Coates, UCSD
; Subhon Ghosh, UCSD
; Farhat Ahmed, UCSD
; Praneeth Surapaneni, UCSD.

their room in the morning it will trigger an Alexa announcement that will then read out the users information for the day. In this way the user will be more prepared and ready for their day.

In order to implement this project it will connect a Raspberry Pi 5 equipped with sensors to an Amazon Echo Dot through a custom Alexa Routine & Skill. When the door opens, the Pi sends a signal to the cloud, triggering Alexa to automatically announce the current time, weather, daily schedule, and traffic to the first calendar event.

A quick, context-aware reminder system increases punctuality, safety, and preparedness—especially for those with packed mornings or commutes. If the user has a busy morning they may not have the time to check their schedule or the weather conditions for the day so by providing them with an automated message in the morning as they walk out of their door provides them with an easier way for them to prepare for their day and allows them to continue through their morning routine while listening to their announcement. This quality of life improvement allows the user to expedite their morning routine and be better equipped for their day with the information they need to succeed.

The remainder of this paper is organized as follows: **2. Motivation and Background** discusses why we decided to pursue this project, **3. Design** describes our design process from initial planning to final design, **4. System Development** describes ResRemind's Architecture, Technology, and Features in detail, **5. Testing and Evaluation**describes our evaluation process, **6. Collaboration** describes the Team Structure and Weekly Process, and **7. Conclusion and Future Work** details our key takeaways and possible future work.

2 Motivation and Background

Fig. 2. Google Assistant

Previous work in this space includes the Google Assistant automation. The Google Assistant has a routine feature that allows users to automate a sequence of events triggered by a user action. An example of this is to have a sequence of tasks that kick off when the user dismisses their alarm. For example, once a user dismisses their alarm the google assistant will then provide information about the weather outside and traffic conditions. Our project will differ from this because instead of triggering on the user action of dismissing an alarm, it will be triggered when the user goes to leave the room. This is something that a user will always need to do on their way out. There could be some days where

a user may not want to set an alarm and on those days they would not be provided with the information that would be helpful for them to start their day. [1]

Fig. 3. Apple Home

Another similar technology that exists in this space is the Apple Home. Within the Apple Home users can create automations such that when they arrive home it can trigger a sequence of events such as adjusting the thermostat and turning on the light. This technology would be dependent on the user having their phone on them at all times and while this is quite common our technology would differ because it would be dependent on motion from the door or the user walking through rather than a device arriving at the house.[2]

In contrast to pre-existing systems mentioned, ResRemind system removes the need for users to manually check multiple apps. For busy students and professionals, it offers hands-free, real-time information the moment they leave—fitting naturally into existing routines, seamlessly disappearing into the user's daily life. Compared to existing assistants, it adds automation triggered by real-world context.

As busy college students the members of the team don't always have time to check the outdoor weather or their calendar for the day. This can consistently lead to them leaving the house ill prepared for the outdoor conditions leading to them not having the proper attire for the day. This can result in them forgetting an umbrella or having an additional jacket when it will be hot outside.

Additionally, as college students there is always so much on their minds so they may forget specific deadlines or meetings. This inspired the integration of Google Calendar into the automated announcement as it will automatically provide the user with an update of their schedule so they are reminded of their plan for the day. We considered that an announcement each time the user leaves the room could become quite repetitive so we adjusted the user interaction to query the user if they would like to continue the announcement to their calendar schedule or if it should be skipped for the time being. In this way, it provides the user with more flexibility in their announcement and allows them to customize their experience as needed.

3 Design

At the start of the project, the initial design included a Raspberry Pi, ultrasonic sensor and an Amazon Echo. In order to implement the project. the original plan was to use the ultrasonic sensor to determine the door status. Using the

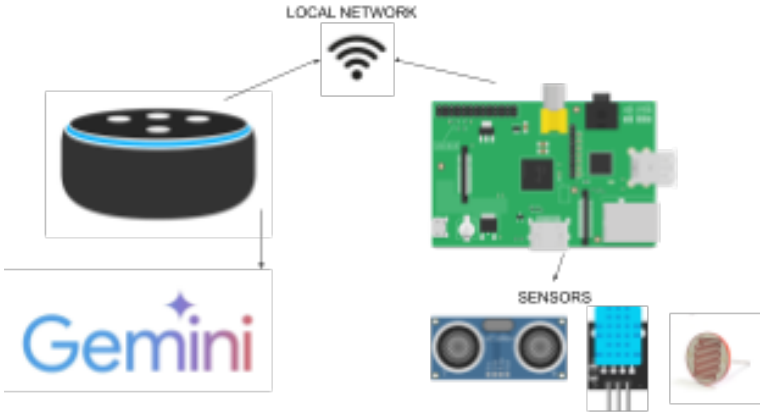

Fig. 4. Design Diagram

ultrasonic sensor and Raspberry Pi, it would be able to determine the distance to the door indicating if the door is open or closed. Once the door sensor status updated, the program would send a POST request to MongoDB Atlas (a cloud database) in order to update the database and show that the door had opened. In the initial design, the plan was to repeatedly send GET requests from the Amazon Alexa Skill to MongoDB in order to check the door status. If the door status had updated to "open", then the Raspberry Pi would send a POST request with the door status, and the Alexa Skill would then subsequently receive the data from its GET request and then access the user's calendar schedule through interfacing with Google Calendar API. This would be parsed through Gemini API, and the weather would also be retrieved from the AI agent. Finally, this parsed, human friendly output would be announced through the Echo Device.

We also decided to add break beam sensors to our system to detect a person walking. The ultrasonic sensor would track the door movement while the break beam sensors would be mounted on either side of the door frame and would detect if the user actually walked through the door. This ensured our detection of the user opening the door and walking through was more robust and accurate than our initial design. The physical system was augmented to include the DHT11 temperature and humidity sensor to detect indoor and humidity temperature. The indoor temperature serves as a helpful reference to the user in deciding how they should change their current attire to be comfortable outside. If a user is just comfortable in an indoor temperature of 67°F and the outdoor temperature is 57°F, they will probably want to wear more layers. Furthermore, the interfacing with Google Calendar API was moved from the Alexa Skill to the Raspberry Pi since the Alexa skill's development environment was too restrictive for installing the packages relevant to the Google Calendar API. As such, the POST Request sent from the Pi to the MongoDB database would include the door status, temperature, humidity, and the user's Google Calendar schedule for the day.

We made these updates to the logic for connecting the Alexa Skill with the sensor detection with Raspberry PI to ensure our concept of hands free announcements. The plan of using an infinite loop for the Alexa Skill proved untenable since custom Alexa Skills time out after 8 seconds. Instead, we opted to use a custom Alexa Routine which is prompted by our Virtual Smart Home switch. The Raspberry Pi simultaneously triggers the Virtual Smart Home switch and makes a POST request with the latest user data. The Alexa Routine would also be used to announce the outdoor weather of the user's location since Gemini could not effectively give the precise weather for a specific location. Once the Alexa Routine is triggered by the Pi, it launches the Alexa skill which then sends a GET Request to the Flask Server to retrieve

the door status, temperature, humidity, and schedule data from the MongoDB database. This is then fed into Gemini API to produce a user friendly summary.

4 System Development

4.1 Architecture

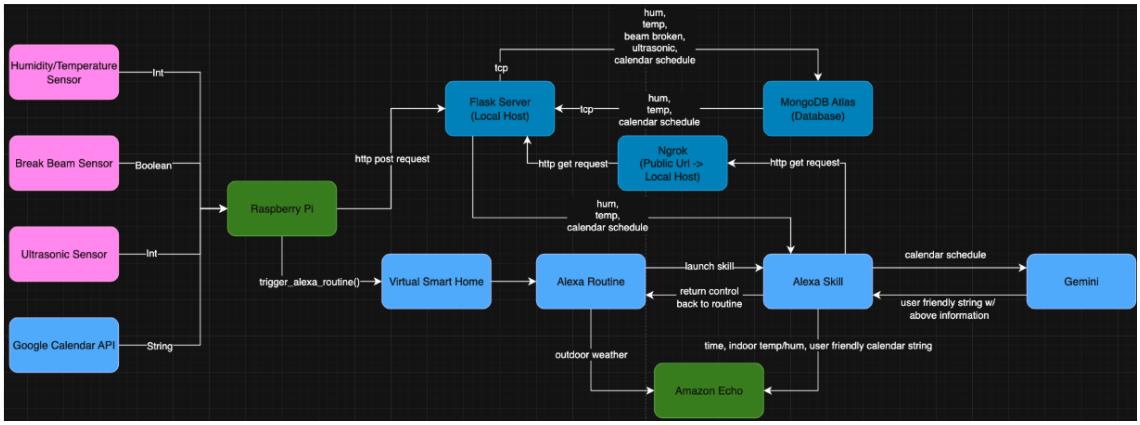

Fig. 5. Architecture Diagram

Overall, the system architecture emphasizes modularity, event-driven communication, and security-conscious design. Sensor data is processed locally, transmitted securely to a centralized backend, and retrieved on demand by the Alexa skill. This separation of responsibilities enables reliable operation within platform constraints while providing a scalable foundation for future expansion.

4.1.1 Client-Server Model. The system follows a distributed client–server architecture. The Raspberry Pi acts as a data-producing client, responsible for collecting sensor inputs and transmitting them to a backend service. The backend database, MongoDB, functions as a central server, storing the most recent system state, including door status, temperature, and humidity. The Alexa skill serves as a consumer client, retrieving this stored information to generate user-facing announcements.

In this architecture, the Raspberry Pi initiates communication by sending HTTP POST requests containing sensor data (temperature, humidity, calendar information, and door status) to the database whenever a significant event is detected, such as the user opening the door while walking through the doorway. When this event is detected, the Raspberry Pi also prompts the Alexa skill to send an HTTP GET request to retrieve the latest state from the database, enabling it to respond dynamically to changes without maintaining a persistent connection.

4.1.2 Input/Output Data Flow. The primary system inputs are sensor readings from the Raspberry Pi, including ultrasonic distance measurements, break beam sensor states, and temperature and humidity values from the DHT11 sensor. These raw inputs are first processed locally on the Raspberry Pi, where logic is applied to detect meaningful events such as door openings or user movement. By performing this preprocessing at the edge, the system reduces unnecessary network traffic and improves responsiveness.

Once an event is detected, the Raspberry Pi packages the processed sensor data into a structured JSON payload and sends it to the MongoDB backend via an HTTP POST request. This data is then stored and made available for retrieval. On the output side, the Alexa skill queries the database using HTTP GET requests. When a relevant state change is identified, Alexa synthesizes a spoken response that includes contextual information such as calendar updates, temperature, and humidity, delivering the final output to the user.

4.1.3 Networking and Communication. The Raspberry Pi and Alexa skill do not communicate directly. The Raspberry Pi communicates with a local Flask server and the Virtual Smart Home switch through HTTP API. The Virtual Smart Home Switch communicates with the custom Alexa Routine, which communicates with our Alexa skill. The Alexa skill communicates with the local Flask server through HTTP API over a public URL mapping through Ngrok. The local Flask server receives and sends data to the MongoDB database over TCP. This design improves modularity and allows each component to operate independently, simplifying debugging and future scalability.

4.1.4 Privacy and Security Considerations. Information privacy and security are handled through multiple layers. Sensitive user data, such as calendar information, is never stored directly on the Raspberry Pi. Instead, calendar summaries are generated through secure API calls and only the minimal information required for user announcements is processed. Communication with external APIs, such as Google Calendar, requires OAuth authentication, ensuring that user consent and access control are enforced.

All network requests are made over secure HTTPS connections to protect data in transit. Database access is restricted through authentication credentials. The database and code can be configured such that write access is limited to the Raspberry Pi client while read access is limited to the Alexa skill. By minimizing stored personal data and enforcing role-based access patterns, the system reduces exposure risks while maintaining functionality.

4.2 Technology Used

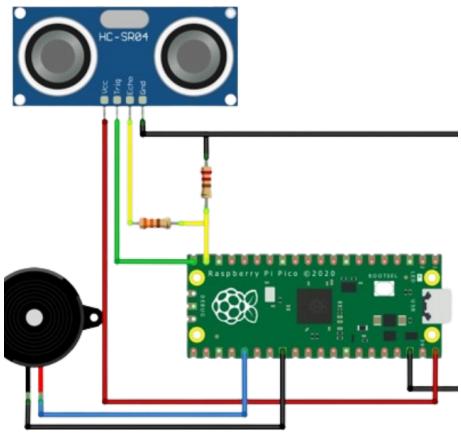

Fig. 6. Design Diagram

The two core technologies that ResRemind relies upon are the Raspberry Pi 5 and the Amazon Alexa Echo Dot devices. With the Raspberry Pi 5, we connected the breakbeam sensor and ultrasonic sensors for detecting the door opening and DHT11 temperature/humidity sensor to detect indoor temperature. Moreover, with the Amazon Alexa Echo Dot, we used Virtual Smart Home to emulate the behavior of a smart home switch. To access the user schedule, Google Calendar API was integrated. To forward and retrieve data, we set up a Flask server which interfaced with the actual database which was stored using MongoDB NoSQL database system. To parse the schedule and temperature data, the custom Alexa skill interfaces with Gemini API to create a user friendly output. Additionally we utilized the built in Alexa weather information to provide the user with an update on the outdoor weather conditions in their automated announcement.

The system integrates multiple hardware devices, software services, and cloud-based APIs to enable end-to-end functionality. At the hardware level, a Raspberry Pi serves as the primary edge computing device, responsible for interfacing with sensors, processing raw input data, and communicating with backend services. The Raspberry Pi provides the necessary GPIO access, networking capabilities, and processing power to support real-time sensor monitoring and event detection.

Several sensors were integrated to capture both environmental and user activity data. An ultrasonic sensor is used to detect door motion by measuring changes in distance over time, enabling reliable detection of door opening events. A break beam sensor is employed to detect when a user physically passes through the doorway, providing directional confirmation of movement. In addition, a DHT11 sensor collects temperature and humidity data, allowing the system to incorporate environmental context into user notifications. These sensors work in conjunction to reduce false positives and provide a more accurate understanding of user activity.

On the software and API side, the system integrates with Amazon Alexa through a custom Alexa skill. The skill enables voice-based interaction and delivers spoken announcements to the user based on system state changes. To generate concise and natural-sounding summaries of user schedules, the system also integrates the Gemini API. Gemini processes structured schedule information and returns a summarized response that Alexa can read aloud, improving clarity and user experience.

For data persistence and inter-component communication, a MongoDB database is used as the central backend. The database stores the latest sensor readings and system states, such as door status and room temperature. The Raspberry Pi sends sensor updates to the database via HTTP POST requests, while the Alexa skill retrieves information using HTTP GET requests. This approach decouples sensor data collection from voice interaction and allows each component to operate independently.

Additionally, the system explores integration with external calendar services, such as the Google Calendar API, to retrieve user schedule information. Access to calendar data requires OAuth authentication to ensure secure and user-authorized access. Although full integration remains an area of ongoing development, the system architecture is designed to support secure retrieval and processing of personal calendar data.

4.3 Features

Door Opening and User Walkthrough Detection - The system offers robust detection of door opening and the user walking through. We have the break beams detect the user walking through. The sensors are mounted to either end of the door frame. Once the beam is broken, we decide the user has walked through the door frame. Coupled with this, we used an ultrasonic sensor to detect the actual door moving/opening. The ultrasonic sensor is mounted to the top of the door. We capture a sample of distance values and check how much the standard deviation fluctuates. If it fluctuates

highly, we sense the door has opened. Combining these two sensing systems, we are able to detect if the user has both walked through the door and opened the door to step out.

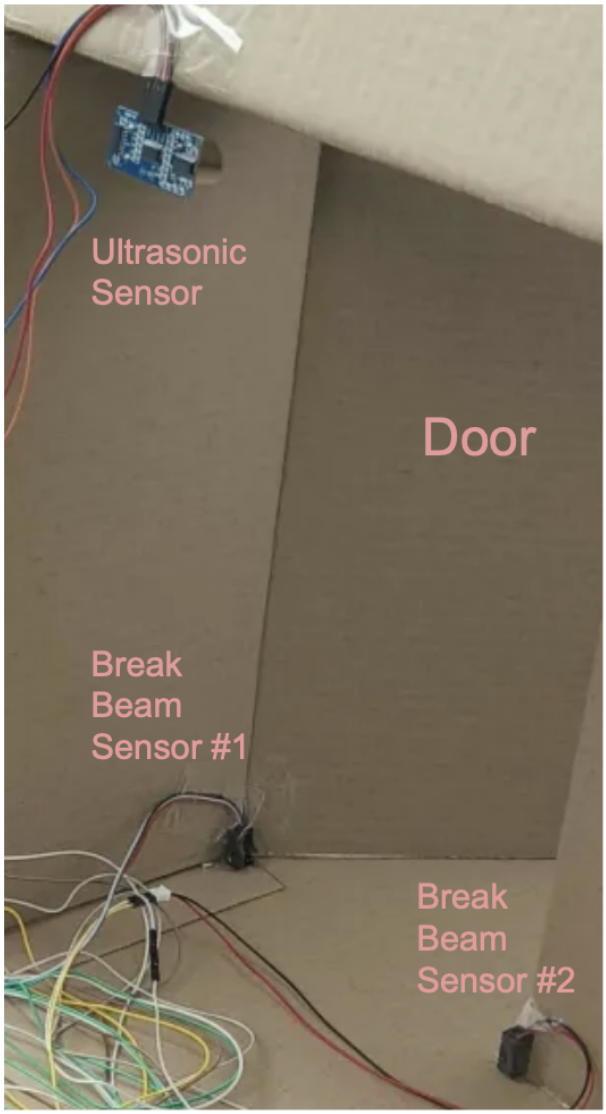

Fig. 7. Door Sensing Setup

Temperature/Humidity Detection - the connected DHT11 sensor collects the indoor temperature and humidity of the user's residential space. The associated Python script converts the temperature to Fahrenheit ensuring both Fahrenheit and Celsius readings. We took these indoor weather condition readings and added them to our database POST request in order to make the information accessible from the Amazon Alexa skill. Once they were available in the MongoDB database, we were able to make a GET request from the Amazon Alexa skill to access this information

and include it in a hardcoded structured announcement for the user including the time, temperature and humidity conditions within the user's household.

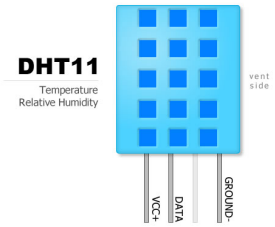

Fig. 8. DHT11 sensor

Google Calendar Integration - The Raspberry Pi interfaces with the user's google calendar through Google Calendar API. The script for this is called when the door opening and user walk through is detected. Only the schedule for the day is pulled, it becomes part of the data the Pi sends out.

Fig. 9. Google Calendar

Posting Pi Data - Once the door opening and walk through has been detected, we send a POST request with the door status, indoor temperature/humidity, and the user's schedule for the day to a Flask server which updates the MongoDB Database using the TCP networking protocol.

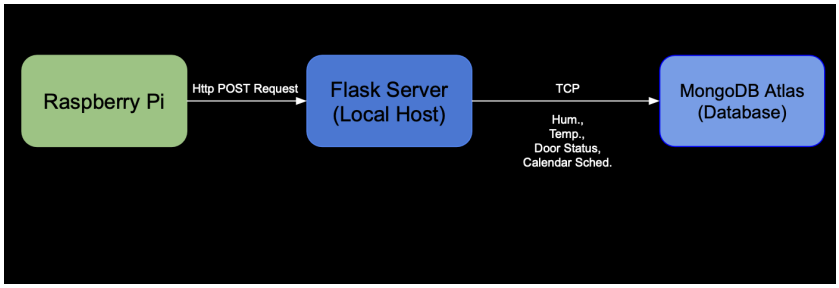

Fig. 10. Raspberry Pi Post Request to DB

Launching Alexa Skill Handsfree - On the Echo side, the Raspberry Pi launches our custom Alexa skill through triggering an Amazon Alexa Routine which relies on the Virtual Smart Home. The Amazon Alexa skill provides the

time, weather, indoor temperature/humidity and summarizes the user's schedule for the day in a friendly conversational manner using Gemini intelligence. Before issuing announcements, the skill asks the user if they actually want to hear their announcements. In order to trigger the Amazon Alexa announcement without requiring additional user interaction we looked into multiple different ways of implementing this.

The initial plan we had to begin this announcement was for the user to trigger the Alexa skill and have the skill run in a continuous loop. This would constantly check the MongoDB database for updates in the door status. In the event, that a user would walk through the doorway the Alexa would have been querying the database and found the information and begun the announcement. However, once we began this implementation we ran into a problem. We found that the Alexa skill has a runtime limit of 8 seconds so once the user triggered the skill this process would only work for 8 seconds before timing out and no longer would automate the system announcement. This also takes away from the project because it would have still required user interaction in order for the project to work with the initial user interaction to trigger the skill. Once we realized this option would not be feasible we began looking into other ways to implement this. In our research, we found that we could use a virtual smart home device in order to trigger the Amazon Alexa announcement without requiring user interaction aside from them walking through the door. In order to implement this, we created a virtual home device and set up an api call from the raspberry pi that triggered this virtual smart home. Once the virtual smart home was triggered by the api call we set that up to trigger an Amazon Alexa Routine. This Alexa Routing launches the custom skill we created.

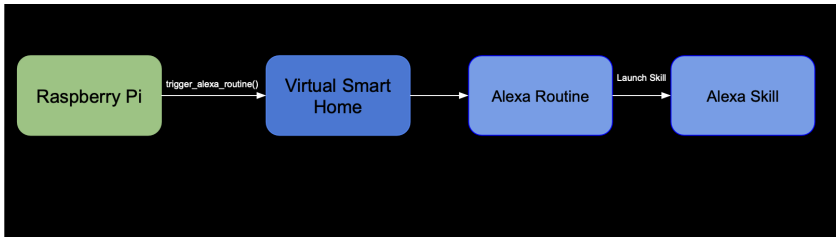

Fig. 11. Launching Alexa Skill

Retreiving Pi Data - Once the Alexa skill is launched, it sends a GET Request to Ngrok which creates a secure tunnel connection to the Flask Server which then pulls the user data(temperature, humidity, door status, and schedule) for the day from the MongoDB database.

Gemini API Integration - Once the user data is obtained from the GET request, the Alexa skill forwards the user's schedule data to Gemini API for a user friendly summary which is returned in a string back to the skill.

Fig. 12. Gemini API

Providing User Announcements - Initially, the Alexa Routine provides the outdoor weather of the user's location. Once the Alexa skill launches, retrieves and processes the user's data, the Alexa Skill announces the time, indoor temperature and humidity, and the user friendly calendar summary.

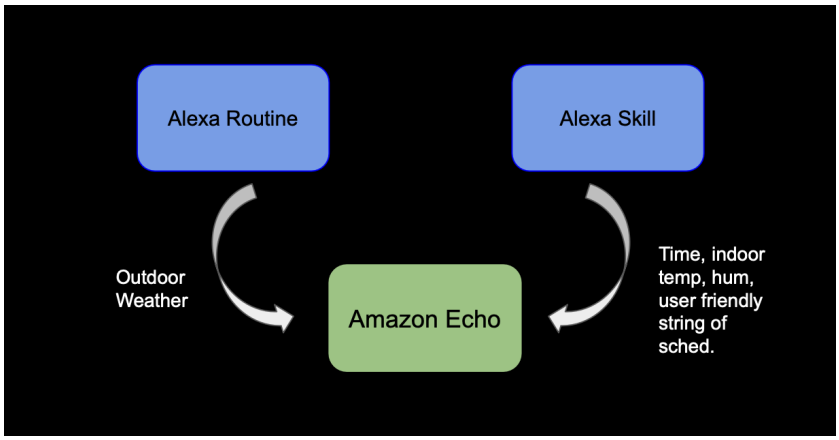

Fig. 13. Announcements through Echo

5 Testing and Evaluation

5.1 Testing

Throughout the project, there were various tests utilized to determine the success and progress of the project. Starting the project, we tested if the ultrasonic sensor was properly functioning and determined if the door was open or closed by holding it in front of a door and opening and closing it. While the door was opened and closed, we had console logs to inform us of the ultrasonic readings, and we utilized this to find a proper threshold in order to properly set the door status to either open or close.

In order to test the functionality of the full project, towards the end of the project we set up a doorway with a person holding the ultrasonic sensor in place facing the door and people holding the break beam sensors at the bottom of the doorway in order to track the person walking through the door. We tested the functionality of the project by having a user walk through the doorway and open the door which we anticipated would trigger the Alexa announcement including the indoor and outdoor weather conditions, time and calendar updates.

Another way we tested the full functionality of this project without a proper doorway was utilizing a cardboard box. We took a cardboard box and formed it into a makeshift doorway with one of the side panels opening and closing like a door. For this set up we were able to mount the sensors with tape to the box. In this way we had the ultrasonic sensor mounted facing towards the door frame and the break beam sensors were taped to either side of the door frame in order to check if there was an object moving through the door. Once we had set up the sensors around the door frame, we tested the functionality of the system by opening the cardboard panel while pushing an object through the door frame simulating a user walking through. This would then trigger the Alexa that was sitting nearby on the table which would then read out an announcement including the weather, time and calendar updates.

Fig. 14. Testing Setup

5.2 Successes

The biggest success we had throughout this project was implementing the program by triggering the Amazon Alexa without user voice input or additional action from the user aside from just walking through the doorway. We found that by using a Virtual Smart Home Device we were able to make an API request from the Raspberry Pi that triggered the Virtual Smart Home Device we created. Once the Virtual Smart Home Device was triggered, we then used that initiative to trigger an Alexa Routine which triggered our custom Alexa skill. In addition to triggering our Alexa skill, we incorporated the built in Alexa weather information to report the local weather to the user in addition to our skill which gave the user their calendar information for the day.

Another success we found through the process was pulling the Google calendar information for the user. Our original plan had been to make a Google API call from the Alexa skill. During development though we found this process to be quite difficult because it would require additional security measures. In order to create the Google API call from the Amazon Alexa skill, we were unable to import any external google libraries that would have eased the process. So we were required to set up an OAuth and take additional security measures to pull the calendar information. In order to work around this difficulty, we realized that we would be able to make the google api call from the Raspberry Pi. In doing so, we were able to use the google libraries that simplified the process significantly. Once we were able to import the Google libraries, it was quite simple to then make the Google api call form the Raspberry Pi, pull the user calendar information and then add that information to the MongoDB database. Once that information was in the database we were able to pull it from the Alexa skill in order to include the information in the announcement to the user. In order to create a clear announcement that included this information, we made a Gemini call that passed through the user schedule which was in a json format. By prompting Gemini with the calendar information, we were able to get a clear and concise summary of the user's day that would adjust and change each day along with the user's calendar.

5.3 Failures

While testing this project on a full door frame, we found there were multiple instances of false negatives. There were multiple times when we walked through the door and found that the Alexa did not begin the announcement as it should. In these instances, we believe this may have been an error in the break beam if we were unable to set them up properly. At the time the break beam sensors were being held up by people. This inconsistency in the position of the break beam could have created issues with the break beam sensor causing it to incorrectly sense the door status as not walked through even after the user had walked through.

Another idea for what may have caused the false negatives was if the user was not properly stepping in the line of the break beam sensor. In order to make it easier to line up the break beam sensor at the same height, we positioned them on the floor. Since the break beam was on the floor, we realized that it was possible for the user to step through the doorway but have moved their foot above the break beam sensor and not trigger the walkthrough sensor. In these cases, the Alexa announcement would not begin. In order to fix this issue, we would need to determine a better positioning for the break beam sensor such that it was high enough off the ground to ensure there would be no chance that the user could walk through the doorway without triggering the break beam sensor. Another solution could be to have multiple break beam sensors at various heights on the doorway in order to account for different user heights and gaits as they walk through the door. By potentially implementing this with multiple break beam sensors, it provides a more robust system in order to check a person walking through the doorway.

6 Collaboration

6.1 Team Structure

Generally, the work was split between the Raspberry Pi and the Amazon Echo. Praneeth and Farhat worked on the Raspberry Pi while Hannah and Subhon worked on the Alexa Skill. Subhon also worked on networking between the devices by setting up the MongoDB database and Flask server. On the Raspberry Pi side, Praneeth developed the algorithm employing standard deviation to detect the door opening with the ultrasonic sensor. Farhat worked on wiring the components in the Raspberry Pi and developed the scripts for the brake beam sensor and temperature/humidity sensor. Subhon helped unify these scripts and combined them with the networking logic to send a post request to the database when the door opened and user walk through was detected. Hannah critically developed the Alexa Skill and Gemini API integration which parsed the sensor data and scheduled data for user-friendly conversational output. Moreover, she retooled the Alexa skill to be launched by an Alexa Routine which enabled the Alexa skill to be launched handsfree without an app or voice command. By using the Alexa Routine, the project was able to be launched using a Virtual Smart Home Device.

6.2 Progress Breakdown

6.2.1 Week 1. During the first week of the project, the tasks were split by device such that Farhat and Praneeth worked on the Raspberry Pi and Subhon and Hannah worked on the Alexa development. For the initial steps of the project we wanted to start by setting up a baseline for each of the devices. So for the Amazon Alexa, we set up a new skill that made a call to Gemini in order to demonstrate the skill creation and the usage of Gemini which we planned to integrate into the project. On the Raspberry Pi side, we just wanted to collect the sensor information. So for the first week, Farhat and Praneeth worked on setting up the ultrasonic sensor with the raspberry pi. In addition to setting up the sensor, they worked on developing the logic to test that the door was open. For this, they tracked the last 15 measurements of

the sensor and measured the standard deviation. In this process, they worked to test different thresholds to determine the best one that would consistently indicate if the door had been opened.

One of the primary challenges we faced was developing a reliable algorithm to detect when a door was opened using the ultrasonic sensor. Raw distance measurements were often noisy due to environmental factors such as slight vibrations, sensor angle, and surrounding objects. Initially, simple threshold-based approaches produced inconsistent results and frequent false positives. To address this, we implemented a method that analyzes the latest 15 sensor readings and computes their standard deviation. Through experimentation, we found that a high standard deviation consistently correlated with door motion, allowing us to distinguish meaningful movement from background noise. Determining the appropriate threshold required iterative testing and tuning.

We also encountered difficulties when integrating the Gemini API for generating schedule summaries. A key challenge was determining the correct prompt formatting to produce concise, structured responses suitable for Alexa to speak. Early attempts resulted in overly verbose or ambiguously formatted outputs that were not ideal for voice interaction. Refining the prompt required careful experimentation with instruction wording, response constraints, and example formatting to ensure Gemini consistently returned clear, spoken-friendly summaries. This week we designed a baseline Gemini API call using a hardcoded schedule required balancing simplicity with extensibility. While the initial implementation focused on validating the end-to-end flow from schedule input to Gemini-generated summary and Alexa output we had to ensure that the prompt structure would later support dynamic user data. This required forethought in how schedule information was presented within the prompt, laying the groundwork for future integration with real user calendars.

6.2.2 Week 2. During this phase of the project, we significantly expanded the system's sensing and data integration capabilities. We added environmental awareness by developing a script on the Raspberry Pi to collect temperature (in both Celsius and Fahrenheit) and humidity data using a DHT11 sensor. This enhancement allowed the system to move beyond simple motion detection and begin incorporating contextual information about the user's surroundings. In parallel, we implemented a separate script using a break beam sensor to detect when a user walks through the doorway. This detection mechanism is designed to be augmented with logic from the ultrasonic sensor, improving reliability by combining multiple sensor inputs.

We also made progress on backend integration by connecting the Alexa skill to a MongoDB database. Through this integration, Alexa can make GET requests to retrieve stored system state information, such as the current door status and room temperature. This step marked an important milestone in enabling persistent data storage and real-time information retrieval, allowing the system to maintain continuity across interactions and better support future feature expansion.

One of the main challenges during this stage was achieving reliable sensor precision and setup. Getting the DHT11 temperature and humidity sensor to function correctly required installing specific packages on the Raspberry Pi, which proved to be nontrivial. The process involved managing dependencies and working within virtual environments to avoid conflicts with existing system packages. Troubleshooting these setup issues required careful documentation review and iterative testing before stable sensor readings could be obtained.

We also encountered difficulties with the hardware wiring of the break beam sensor. The sensor required the use of a resistor to ensure proper signal behavior, and incorrect wiring initially caused the output to remain constant, incorrectly indicating that the beam was always broken. Diagnosing this issue involved examining both the circuit design and the

GPIO configuration. Once the resistor was added and the wiring corrected, the sensor began producing the expected state changes.

In addition, integrating with the Google Calendar API introduced new complexities related to authentication and security. Making GET requests to access user calendar data required implementing OAuth authentication and handling additional security constraints. Determining the most appropriate approach for securely obtaining and managing user credentials was challenging, particularly when balancing ease of implementation with best practices for user privacy. This made it difficult to identify the optimal method for retrieving calendar information in a way that could scale beyond initial testing.

6.2.3 Week 3. This week, we focused on consolidating system functionality and improving end-to-end responsiveness. We developed a unified script on the Raspberry Pi that combines input processing for the ultrasonic sensor, the temperature and humidity sensor, and the break beam sensor. By centralizing sensor logic into a single process, we reduced redundancy and improved synchronization between sensor readings. When the system detects that a user has walked through the doorway, the script sends a POST request containing all relevant sensor data, enabling downstream components to react immediately.

On the software integration side, we enhanced the Alexa skill to continuously monitor door status updates. A loop-based approach was implemented to check for changes in the door state, and when an update is detected, Alexa initiates an announcement that includes calendar updates as well as current temperature and humidity information. In parallel, we began exploring alternative triggering mechanisms, including externally triggered routines, to improve responsiveness and reduce reliance on user-initiated interactions.

We encountered additional hardware challenges related to the break beam sensor. Achieving reliable detection required a specific resistance level that was not initially documented. Identifying the correct resistor value involved hands-on troubleshooting using a multimeter and iterative testing. Incorrect resistance caused unstable or constant readings, reinforcing the importance of precise electrical configuration in sensor-based systems.

A significant limitation arose from Alexa skill execution constraints. Alexa skills are subject to an approximately eight-second timeout, which makes it infeasible to rely on infinite or long-running while loops to continuously listen for incoming POST requests. This constraint required us to reconsider our design approach and explore event-driven or externally triggered alternatives rather than persistent polling within the skill itself.

Finally, triggering the Alexa skill externally proved to be a complex challenge. While proactive events initially appeared promising, they introduced similar limitations by requiring additional user interaction before announcements could begin. We also investigated using Alexa Routines triggered by a Virtual Smart Home device, which may provide a more viable pathway for hands-free activation. Evaluating these approaches required balancing technical feasibility, platform limitations, and user experience.

6.3 Problems Faced

One significant technical challenge we encountered was how we could launch the Alexa skill without an app or voice command once the door opening was detected. Initially, we decided to have the Alexa skill run in an infinite loop and wait for data to be posted in the database. However, the Alexa skill times out after 8 seconds making this plan unfeasible for our project as we would not be able to continue checking the database for updates on the door status. Implementing in this way would also require user interaction to first initialize the skill so that it could repeatedly query the database. Next, we explored using the Proactive event, however this also proved ineffective since proactive events

send a notification to the app which would break our notion of a hands free ubiquitous system. Using the proactive event would fix the issue of the program timing out at 8 seconds. Using a proactive event would have allowed us to no longer need to consistently check the database. Instead the program would need to make an API call to the Alexa system to trigger this proactive event which would then send a notification to the user. Once the notification appears on the Alexa as a lit up ring it would then require user interaction in order to begin the announcement. The proactive event simply prompts the user to initiate the Alexa which takes away from the automated response as it requires user interaction. Ultimately, we settled on using the Alexa Routine system to trigger the skill. Once the door walk through is detected, our script uses an api request to activate the Virtual Smart Home, essentially a virtual switch which triggers the routine which launches the skill.

Another technical issue we encountered was that the break beam sensors were initially not working correctly, showing the beam as broken no matter how much we physically lined up the sensors. We investigated different Python script implementations to see if there was an error in processing the sensor data but saw no change. Then, we re-evaluated our hardware setup. Using a multimeter, we discovered that a stronger resistance was required since the break beam was delivering too much power to the Pi. After replacing the original 3 kΩ resistor with a 10 kΩ resistor, the sensor began to work correctly showing the beam as unbroken when nothing passed between the sensors and as broken when something passed through the sensors.

7 Conclusion and Future Work

Our system posseses a strong, robust strong foundation for significant feature expansion with innumerable possibilities. If we had more time and resources, we could implement features such as giving traffic updates for the first location in the schedule by connecting with a traffic tracking api. We could also offer clothing suggestions according to weather especially if the weather is extreme with snow or rain. Also, we could create a simple, minimal web app, where users can input their google calendar. Also using our same process flow, when returning home, upon detecting the door opening and user walking through, we could turn on lights and offer updates on news, emails, and package deliveries. To achieve wireless abilities, the sensors could be divided among Raspberry Pi Picos.

7.1 Overview

Our system is designed with robustness and extensibility at its core, providing a strong foundation for future feature development. The modular architecture allows for seamless integration of new services, APIs, and hardware components. With additional time and resources, this platform could evolve into a comprehensive smart assistant that proactively supports users throughout their daily routines. This program could also become more customizable creating an easily accessible user interface or allowing users to add in additional sensors or information to their announcement.

7.2 Future Developments

One potential expansion involves enhancing the system's awareness of the user's schedule. By integrating a traffic-tracking API, the system could analyze real-time traffic conditions for the user's first scheduled destination of the day. This would allow it to provide proactive alerts, such as notifying the user when to leave earlier than usual due to congestion or accidents. This updated program could also allow for a better update for the user not only providing them with their calendar schedule for the day but also with suggestions for when they should leave to get to their next event. This integration with a traffic API would improve the information flow of the announcement. In addition, the system could incorporate weather APIs to provide contextual recommendations. For example, in extreme conditions such as

heavy rain or snow, the system could suggest appropriate clothing or accessories, ensuring users are better prepared before leaving their home. To improve accessibility and user customization, a lightweight and minimal web application could be developed. This web app would allow users to securely connect their Google Calendar, manage preferences, and view upcoming events. By centralizing configuration in a web interface, users could personalize notifications, control feature toggles, and gain better visibility into how the system operates. Using the same process flow implemented for morning routines, the system could also support arrival-based automation. Upon detecting the user returning home—through door sensors and motion detection—the system could automatically activate lights and provide a brief spoken or visual summary of relevant updates. These updates could include news highlights, unread emails, or package delivery notifications, creating a seamless transition from outside to inside the home. To enable wireless communication and improve scalability, sensor responsibilities could be distributed across multiple Raspberry Pi Picos. Each Pico could manage a subset of sensors and communicate wirelessly with a central controller. This approach would reduce wiring complexity, increase reliability, and allow the system to scale efficiently as additional sensors or rooms are added.

7.3 Long-Term Vision

Ultimately, these enhancements would transform the system from a single-purpose solution into a flexible smart-home platform. By leveraging real-time data, distributed hardware, and intuitive user interfaces, the system could continuously adapt to user needs and environments, offering meaningful assistance while remaining unobtrusive and easy to use.

8 References

1 Google Support. Automate daily routines & tasks with Google Assistant. https://support.google.com/assistant/answer/7672035?hl=en&co=GENIE.Platform%3DAndroid

2 Apple Support. September 15, 2025. Create scenes and automations with the Home app. https://support.apple.com/en-us/102313