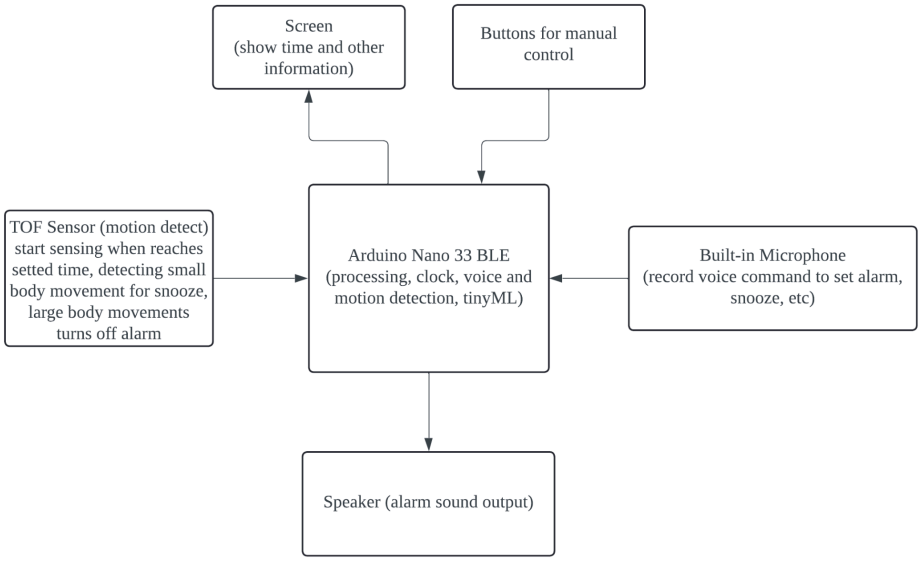
Final Report for a Smart Alarm Clock with Voice Control and Motion Recognition

Yuhong Yao (yuhongy@andrew.cmu.edu), Lai Jiang (laijiang@andrew.cmu.edu)

1. Abstract

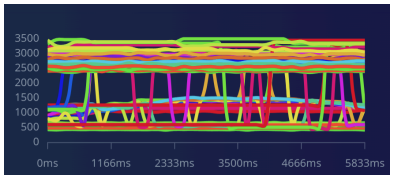
We present a smart alarm clock that uses embedded hardware and onboard inference to recognize voice commands and detect wakeful motion before disabling. The system runs compact models on a low-power platform and remains active until it confirms full wakefulness through spoken or physical cues.

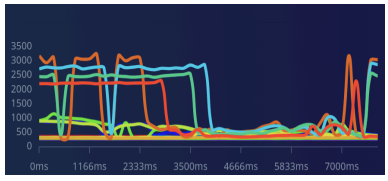


2. Dataset source and method of collection and data cleaning

The audio classification component targets voice commands for invoking, setting, and stopping the alarm, including “Zero” through “Nine,” “SetClock,” “Yes,” “No,” “Stop,” and “Other.” Except for “SetClock,” all commands were drawn from the Speech Commands dataset, with 300 randomly selected 16 kHz, 1000 ms samples each. “Other” was similarly constructed from unrelated commands, while “SetClock” was recorded in-house to produce 300 samples matching the same format.

For motion recognition, this project employs a Time-of-Flight (ToF) sensor (VL53L5CX) to collect data corresponding to four distinct motion classes, including still, idle, leave bed, and flip. We chose to use a ToF sensor multizone ranging output with 4x4 separate zones instead of 8x8, since using ToF with resolution 8x8 and 4x4, the flip data are shown: left 8x8 and right 4x4.





We use a 4x4 resolution for motion sensing to reduce noise sensitivity from ambient light and reflective surfaces while lowering computational overhead. Given that broad motion patterns remain adequately captured at this scale, we represent each motion class with 100 samples, each trimmed to a 5-second duration to ensure consistency and highlight the most relevant segments. For both the voice classification and motion detection datasets, an 80/20 train-test split is employed. The model can be improved by introducing a label going_bed for better performance.

3. Feature Extraction

For speech recognition, the system employs Mel-frequency cepstral coefficients (MFCCs) for feature extraction. The MFCC parameters include two coefficients, a frame length of 0.02 seconds, a frame stride of 0.02 seconds, 32 filters, an FFT length of 256, a normalization window size of 101, and a low-frequency cutoff at 50 Hz with no high-frequency limit. Additionally, a pre-emphasis coefficient of 0.98 is applied.

For motion recognition, each axis is scaled by a factor of ten before computing spectral features using a 64-point FFT. The resulting spectra are converted to a base-10 logarithmic scale. When multiple FFT computations are necessary to cover a window, the latter half of the previous FFT frame is reused to ensure continuity and efficient feature extraction. There was another attempt using raw data to treat each axis of the 4*4 ToF sensor output as a pixel. Think about a super low-resolution image. In this case, 4*4 resolution might not be enough for small movements such as flip, but that was also an option.

4. Classifier architecture and rationale for it

In speech recognition, we make the architecture simple to fit on an Arduino nano board while keeping the classification accuracy above 90%. Our classifier processes 1,000 input features reshaped into a 20-column format, then passes them through a sequence of three 1D convolutional and pooling layers with increasing filter counts (16, 32, 64), each followed by dropout to mitigate overfitting before flattening and classifying into 16 output classes. The initial layer uses fewer filters to capture fundamental acoustic patterns, while subsequent layers increase filter counts to learn more complex, discriminative features. Dropout layers are interspersed to reduce overfitting and ensure robustness. When the data is flattened and passed to the output layer, the network has distilled essential temporal and spectral characteristics from the input, enabling more accurate classification.

The classifier for the motion detection task is a simple feed-forward network composed of two fully connected layers (128 and 64 units) with ReLU activation functions and interspersed dropout for regularization. L1 regularization is also applied to encourage sparsity and improve generalization. A final dense layer with a softmax activation outputs the probabilities for the four motion classes. This design balances computational efficiency and representative capacity, making it practical for embedded deployment while maintaining sufficient discriminative power.

5. Confusion matrix and overall accuracy metrics

For the Motion Detection model, the confusion matrix is:

	FLIP	IDLE	LEAVE_BED	STILL
FLIP	100%	0%	0%	0%
IDLE	0%	89.5%	0%	10.5%
LEAVE_BED	0%	0%	100%	0%
STILL	0%	10%	0%	90%
F1 SCORE	1.00	0.92	1.00	0.86

As shown by its confusion matrix, the motion detection model is 95.4% accurate overall.

For the Speech Recognition model, the confusion matrix is:

	EIGH	FIVE	FOUI	NINE	NO	ONE	OTHI	SETC	SEVE	SIX	SNO(	STOP	THRE	TWO	YES	ZERC
EIGHT	95.3%	0%	0%	0%	0%	0%	2.3%	0%	2.3%	0%	0%	0%	0%	0%	0%	0%
FIVE	2.9%	91.2%	0%	0%	0%	0%	2.9%	0%	0%	0%	0%	2.9%	0%	0%	0%	0%
FOUR	2%	0%	80%	0%	0%	2%	10%	0%	0%	0%	0%	2%	0%	4%	0%	0%
NINE	0%	1.6%	0%	86.9%	1.6%	4.9%	3.3%	0%	0%	0%	0%	0%	1.6%	0%	0%	0%
NO	2.0%	0%	0%	4.1%	89.8%	0%	2.0%	0%	0%	0%	0%	0%	0%	2.0%	0%	0%
ONE	2.1%	4.2%	4.2%	2.1%	0%	87.5%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
OTHER	3.8%	19.2%	5.8%	5.8%	11.5%	13.5%	26.9%	0%	0%	0%	0%	0%	5.8%	3.8%	1.9%	1.9%
SETCLOCK	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%
SEVEN	4.5%	6.8%	2.3%	2.3%	0%	0%	0%	0%	77.3%	0%	0%	2.3%	4.5%	0%	0%	0%
SIX	0%	0%	0%	0%	0%	0%	0%	0%	1.9%	98.1%	0%	0%	0%	0%	0%	0%
SNOOZE	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%
STOP	0%	4.1%	2.0%	0%	2.0%	0%	4.1%	0%	2.0%	0%	0%	85.7%	0%	0%	0%	0%
THREE	2.3%	0%	0%	0%	0%	0%	0%	0%	0%	2.3%	0%	0%	86.4%	4.5%	4.5%	0%
TWO	3.6%	0%	0%	0%	0%	0%	1.8%	0%	0%	1.8%	0%	1.8%	7.3%	81.8%	0%	1.8%
YES	2.1%	0%	0%	2.1%	2.1%	0%	4.3%	0%	0%	0%	0%	0%	2.1%	0%	87.2%	0%
ZERO	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	7.8%	0%	92.2%
F1 SCORE	0.85	0.75	0.82	0.87	0.86	0.83	0.35	1.00	0.84	0.97	1.00	0.88	0.82	0.81	0.90	0.94

As the confusion matrix indicates, the speech recognition model achieves an overall accuracy of 85%. While the accuracy for the “other” category is somewhat lower, the primary voice commands are identified with satisfactory precision for this application.

6. Deployment method: hardware target and software techniques

The deployment of the smart alarm involved multiple hardware components and sophisticated software implementation to ensure functionality.

1. **Hardware:**

   a. **Arduino Nanos:** The system included two Arduino Nanos. The first is the central part of motion detection and alarm clock systems. It interfaces with a multizone Time of Flight sensor and manages alarm functions based on motion detection and user input. The second nano is dedicated to processing audio inputs for speech recognition. It enables users to set, snooze, and turn off the alarm through voice commands. The built-in microphone module is used.

b. **Multizone ToF Sensor:** This project uses the VL53L5CX sensor. It helps achieve motion detection by range sensing the depth of a zone, enhancing the system's responsiveness to motions such as flipping and leaving the bed.

c. **TFT Display (ST7789)**: This 240*320-pixel display visually presents real-time data, including the current time, data, and alarm status. It allows us to interact with the alarm setting and sense of time.

d. **Passive Buzzer:** An audible alarm is generated and amplified by the transistor circuit (2N2222 and 2kΩ resistor) to alert the user when the alarm is triggered. There are multiple tones we can choose to play for the alarm sound.

e. **UART Communication:** Serial Communication via TX and RX pins exchange data between speech recognition Arduino and the motion detection Arduino. Commands are transmitted through this channel to control the alarm system.

2. **Software Techniques:**

a. **Firmware architecture:** Structured around a modular design. Key modules are:

i. Motion detection Module: handles data acquisition from the ToF sensor, processes motion recognition using an Edge Impulse model, and triggers alarm actions based on the predicted motion.

ii. Alarm Management Module: This module manages the alarm state, including setting, snoozing, and turning off alarms. It interacts with the RTC module to schedule alarms and with the TFT display to provide visual feedback.

iii. Speech Recognition Module: This module processes audio inputs to recognize voice commands. Upon successful recognition, the corresponding commands are sent to the alarm management module via UART.

b. **State Machine Implementation:** Ensures orderly transitions and prevents conflicts

i. Standard display mode: In this mode, the TFT displays the date, time, and day of the week in real time, and the alarm state is OFF. If the user needs to set a clock, the speech model continuously plays messages.

ii. Set Clock Mode**:** In this mode, the set clock is detected and activated upon receiving a command from the speech recognition module. It facilitates digit-by-digit alarm settings, providing prompts and confirmations through the TFT display.

iii. Alarm Ringing State**:** In this mode, the impulse running on the central hub detects human motion and awaits the corresponding action to snooze them or turn off the alarm.

7. Significant challenges and lessons learned

During the audio classification model's development, we observed that handling the "SetClock" command could occasionally require more than one second for inference, potentially introducing

classification errors. Additionally, accuracy limitations sometimes necessitated multiple voice prompts to configure the alarm time successfully. Given the constraints of embedded deployment, maintaining a light parameter footprint dictated that users input digits one by one, which, while efficient, reduced overall convenience.

One of the primary challenges in developing the motion detection model was making motion detection work on the embedded device rather than live classification on the edge impulse. I learned that keeping up with the window size during training is necessary. The second lesson is implementing an efficient circular buffer for continuous motion detection. The optimal buffer size and strategies were investigated deeply to make the motion detection work fluently.

Despite our efforts to minimize resource usage, integrating both models proved infeasible because running the voice and motion recognition systems simultaneously on a single Arduino Nano board proved infeasible. This limitation led to deploying the models on separate boards, ultimately increasing the product's cost.

This project underscored the importance of balancing accuracy, inference speed, and parameter efficiency when implementing machine learning on embedded systems.

8. Demo Link

https://drive.google.com/file/d/1PefDenfCrIEu5YG5WElhd2q0_TBEkD16/view?usp=sharing