ALC-Guard: An IoT-based Alcohol Intake Risk Awareness System

Jingyang Liu 02252219
Dyson School of Design Engineering
Imperial College
jl6522@ic.ac.uk

***Abstract*—Excessive or rapid alcohol consumption is closely associated with various health and behavioral risks. In real social situations, due to frequent refills and unstable rhythms, individuals often have difficulty promptly realizing their cumulative alcohol intake [1][2]. This project designed and implemented ALC-Guard, a lightweight Internet of Things alcohol consumption risk perception system for daily scenarios. The hardware is centered on ESP32, integrating FSR (liquid volume estimation), MPU6050 (angle and motion recognition), DS18B20 (temperature monitoring), and MQ-3 gas-phase alcohol sensor, and classifies events such as "drinking, sipping, refilling, and pouring out" in real time and calculates volume changes.**

The system establishes a connection with the Flask backend via Wi-Fi and uses HTTP POST to only upload events related to alcohol, achieving edge-side filtering and cloud storage. The web dashboard visually presents daily intake, British alcohol unit conversion [3], event timeline and hour distribution, and generates risk levels based on a 60-minute rolling window. The RGB LED on the cup body provides immediate feedback: it flashes when alcohol is detected, and remains red when the risk of excessive consumption is high, thereby enhancing users' risk awareness.

The prototype test results show that this system can stably identify various drinking behaviors, distinguish between alcoholic and non-alcoholic beverages, and provide timely web and physical reminders when necessary. Overall, ALC-Guard demonstrates the feasibility of combining low-cost sensors with cloud analysis for daily behavior monitoring and risk perception.

I. Introduction

A. Background and Motivation

In daily life, the drinking rhythm is often scattered and difficult to detect. Users often overlook the cumulative intake within a short period of time, and studies have shown that rapid or excessive drinking can bring significant health and behavioral risks [1][2]. The existing record tools mostly rely on manual input or rough estimation, making it difficult to capture the actual drinking behavior process and lacking timely feedback. Based on this, this project attempts to build an IoT prototype system that can operate stably in a real desktop environment, enabling it to continuously collect the interaction characteristics between the user and the cup body, and combine cloud visualization and on-site feedback to help users more intuitively perceive their drinking rhythm and potential risks.

B. Project Objectives

This project aims to develop an alcohol intake monitoring system ALC-Guard that can operate independently in a desktop environment. The main objectives of this project include:

- Continuous collection of multi-modal signals related to drinking behavior is carried out, including weight changes, inclination angles, movement patterns, liquid temperature, and vapor alcohol concentration.
- Distinguish typical events such as "drinking, sipping, refilling, and pouring out" in real time on the ESP32;
- Communicate with the server via Wi-Fi, upload alcohol-related events and generate time series visualizations;
- Provide immediate alerts and risk warnings through the web interface and RGB LEDs when the intake exceeds predefined thresholds.

C. Proposed Approach

The system structure is divided into the perception layer, the edge processing layer and the display layer. The perception layer collects the interaction features of the cup body through sensors; the edge layer identifies events through a state machine and filters out non-alcoholic data; the display layer uploads the events to the Flask backend via Wi-Fi and presents the user's drinking pattern in forms such as intake curve, time distribution and alcohol units. Meanwhile, the RGB LED flashes when drinking and keeps the red light on in high-risk situations, forming a complete process of "perception - classification - feedback".

II. System Overview

A. Physical Setup

This system prototype is built based on an ordinary desktop paper cup. The FSR is placed at the bottom of the cup to measure the pressure of the cup on the table and estimate the change in liquid volume; the MPU6050 is fixed on the cup wall to obtain the tilt angle and angular velocity in real time; the DS18B20 probe extends into the liquid from the cup lid to monitor the temperature of the drink; the MQ-3 alcohol gas sensor is installed near the cup mouth to sense the alcohol concentration in the gas phase above the liquid surface, which is used to distinguish between alcoholic and non-alcoholic beverages. All sensors are connected to the ESP32 main control board at the bottom of the cup and provide local light feedback through an external RGB LED, completing the sensor function layout without significantly changing the holding method.

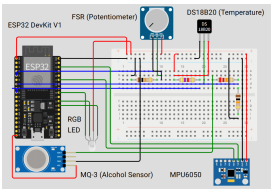

Fig. 1. Physical wiring layout of the sensing module. (The potentiometer is used as a schematic substitute for the FSR load sensor.)

B. System Architecture

This system adopts a three-layer Internet of Things architecture: the perception layer, the device-end processing layer, and the application layer. FSR, MPU6050, DS18B20, and MQ-3 are respectively installed at the bottom of the cup, the cup wall, the inside of the cup, and the cup mouth to collect information such as volume, posture, temperature, and alcohol vapor. These are connected to ESP32 through ADC, I²C, and 1-Wire. The device end is completed by ESP32 for edge computing, performing lightweight smoothing and state machine judgment, distinguishing events such as ”drinking, small sip, refill, and pouring out”. Only the volume of events judged to contain alcohol is accumulated and uploaded to the Flask server via Wi-Fi (HTTP POST), while driving the RGB LED to flash or keep the red light on for feedback. The application layer consists of the Flask backend and a web dashboard based on Chart.js: the backend writes events to CSV and provides APIs, while the front end displays the total intake for the day, the conversion to British alcohol units, the event timeline, and the recent risk progress bar, helping users understand their drinking patterns and risk levels.

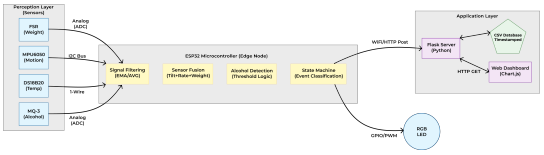

Fig. 2. System Architecture and Data Flow of the ALC-Guard Prototype.

C. Data Flow

The data flow of the system follows a linear process composed of “collection – inference – transmission – display”:

1) Data collection and preprocessing: The ESP32 continuously reads the signals from FSR, MPU6050, MQ-3 and DS18B20, and performs lightweight smoothing processing on the weight and posture data to reduce noise. Only when the current drink is identified as an alcoholic beverage will the intake amount be accumulated and the event be marked as valid.
2) Edge-side event determination: A state machine on the ESP32 monitors the smoothed signals and generates discrete events only when a complete action sequence (pick up → tilt → put down) is detected, achieving data reduction at the edge.
3) Event upload: Valid alcohol-related event is encapsulated as JSON and sent to the Flask server via HTTP POST. Upon reception, the server uniformly adds a timestamp and writes it to the local CSV.
4) Visualization and Feedback: The network dashboard will periodically request the latest alcohol incidents, update the daily intake and time distribution, and infer the risk level based on alcohol units in the last 60 minutes. At the same time, it will issue alerts on the RGB LEDs on the web page and the cups.

D. Design rationale

The design of this system is based on the limitations of the desktop usage scenario and follows the layered architecture of the Internet of Things, edge computing, and privacy protection principles in the course. In terms of sensors, the bottom FSR and the side wall MPU6050 are combined to estimate the volume of the liquid and the action mode. This project adds DS18B20 and MQ-3 on this basis to monitor the liquid temperature and the concentration of alcohol gas, thereby distinguishing between alcoholic and non-alcoholic beverages, demonstrating the idea of enhancing recognition performance through multi-sensor integration [3].

In the computing side, the ESP32 completes the event determination through a state machine locally. Only when it recognizes complete behaviors such as ”drinking alcohol, sipping, refilling, pouring out” does it output discrete events and directly discards the records that are identified as non-alcoholic, avoiding the frequent upload of raw data, and reducing latency and network burden. This ”local judgment, only uploading key events” structure is consistent with the edge processing mode introduced in the course [4].

Data storage and presentation are accomplished through a Flask server within the local network, rather than being uploaded to the public cloud, in order to reduce privacy risks related to drinking behavior and enhance controllability. On the web page, the recent intake situation is visualized through alcohol unit conversion and a timeline. The RGB LED in the cup body flashes when drinking and remains red when the cumulative intake exceeds the threshold, forming a dual-channel feedback between the web page and physical lighting, helping users perceive drinking risks without overly interrupting the rhythm.

III. Methodology

A. Sensing System

This system integrates three heterogeneous sensing channels for weight, posture and temperature, which are used to capture the multi-dimensional interaction behaviors of users during the drinking process. Since the original readings are easily affected by noise, desk vibrations and hand movements, ESP32 performs necessary smoothing, calibration and feature extraction on the data after sampling to ensure the stability of subsequent event classification.

1) Weight Channel: Mean Filtering and Segmented Linear Calibration.
   The bottom FSR is sampled by the 12-bit ADC of ESP32. To suppress short-term jitter, the `readFSR()`

function performs 20 samplings for each reading and calculates the average. To estimate the Liquid volume from the pressure, the system uses three empirical points (Empty cup, Half cup, Full cup) to construct a piecewise linear calibration model.Set the segment base points as (ADC_{base}, V_{base}), then the current volume estimate is:

$$V_{ml} = V_{base} + (ADC - ADC_{base}) \cdot k \tag{1}$$

The slope is calculated based on the adjacent measurement points k. The volume change ΔV of all events is calculated using this model.

2) Posture Channel: Angle Calculation, Calibration and EMA Filtering
The MPU6050 outputs the three-axis acceleration and angular velocity via the I²C bus. The inclination angle of the cup relative to the direction of gravity is calculated from the acceleration vector [4]:

$$\theta_{\text{raw}} = \operatorname{atan2}\big(\sqrt{a_x^2 + a_y^2}, a_z\big) \times \frac{180}{\pi} \tag{2}$$

During the startup phase, the system keeps the cup upright and samples the inclination angle 100 times to calculate the average, thereby forming the zero-point offset θ_{offset}. The relative inclination angle during operation is:

$$\theta = \theta_{\text{raw}} - \theta_{\text{offset}}. \tag{3}$$

To eliminate the high-frequency noise caused by hand tremors, the system applies exponential weighted moving average (EMA) [5] to the inclination angle:

$$\theta_{\text{filt}}[n] = 0.2 \cdot \theta[n] + 0.8 \cdot \theta_{\text{filt}}[n-1] \tag{4}$$

Among them, the smoothing coefficient is set to $\alpha = 0.2$. This parameter was determined through experiments after achieving a balance between "response speed" and "anti-jitter ability".
Subsequently, the angular velocity is calculated based on the change in the smooth inclination over adjacent time intervals and the time interval itself:

$$\omega = \frac{\theta_{\text{filt}}[n] - \theta_{\text{filt}}[n-1]}{\Delta t} \tag{5}$$

The inclination angle and angular velocity together constitute the key posture characteristics that distinguish actions such as "drinking" and "pouring".

3) Alcohol gas channel:
The AOUT output of the MQ-3 gas sensor is connected to the analog port of the ESP32. The function `readMQ3Raw()` continuously samples the current alcohol concentration in the gas phase 10 times each time and takes the average to reduce instantaneous fluctuations. The system has reserved a threshold determination mechanism for the MQ-3: when the smoothed analog value exceeds the calibration threshold, it is considered that the liquid in the cup contains alcohol, and only the cumulative volume of such events is accumulated and uploaded; when it is below the threshold, it is regarded as a non-alcoholic beverage, and corresponding events are filtered at the edge and do not participate in the intake volume statistics.

4) Temperature channel
The Liquid temperature is read by the DS18B20 digital temperature sensor inserted into the cup. The system directly obtains the current temperature during each cycle. This signal has high stability and does not require additional filtering. It is mainly used for visual display on the web page and does not participate in event classification.

5) Feature output
After the above filtering and calibration, the system obtains in each sampling period: a relatively stable weight estimation and its volume change, a smooth inclination and its rate of change, the current temperature, and the determination result of the vapor alcohol strength (alcohol/non-alcohol). These features are collectively used as input to drive the subsequent drinking event classification state machine.

B. Edge-side Event Classification

This system performs edge inference on the ESP32, compressing continuous sensor signals into discrete events related to alcoholic drinking. The process consists of a finite state machine (FSM), a volume-change based pre-classification, and a secondary posture-based judgment, and forms the core of the situational awareness capability.

1) Finite State Machine: The interaction process on ESP32 is divided into two stages:

a) ACTION stage: When any of the following conditions are met, enter the ACTION stage:

- Weight decrease: The current or previous frame's weight is lower than the threshold ($W < 400$);
- Posture tilt: Smooth inclination angle exceeds the threshold for the drinking action ($|\theta| > 20^\circ$)

From this moment the firmware records, for the current event, the maximum tilt $|\theta|_{\max}$ and maximum angular rate $|\omega|_{\max}$, and logs the initial stable volume V_{start} derived from the FSR.

b) ANALYSIS stage: When the cup is placed back on the table, the system switches to the analysis stage, detects through the following methods:

- The tilt angle returns to the upright range ($|\theta| < 10^\circ$);
- The weight rises back above the desktop support level ($W > 400$).

The firmware then waits for a brief stabilization period of approximately 3 seconds. And average value of 25 FSR samples was calculated to obtain the final volume of the state.

2) Preliminary classification based on volume change: Volume change is defined as:

$$\Delta V = V_{\text{start}} - V_{\text{end}} \tag{6}$$

The classification rules are as follows:

- $\Delta V < -10$ ml: Refill liquid;

- $\Delta V > 10$ ml: Significant liquid removal, requiring posture-based judgment;
- $1 \text{ ml} < \Delta V \leq 10$ ml: Small Sip(sip in small sips);
- $|\Delta V| \leq 1$ ml: Regarded as noise.

3) Secondary determination: For events with $\Delta V > 10$ ml, the system uses posture features collected during the ACTION stage to distinguish Drink and Pour Out. The event is classified as Pour Out if

$$|\omega|\max > 25 \quad \text{or} \quad \big(|\theta|\max > 30^\circ \ \wedge \ |\omega|\max > 12\big) \quad (7)$$

otherwise it is regarded as a normal drinking action.

4) Alcohol Filtering and Local Feedback: After the above classification, ESP32 calls the alcohol sensing function to decide whether the current liquid should be treated as alcoholic. Only when the MQ-3 reading exceeds the preset threshold (`MQ3_THRESHOLD`) is the event marked as alcoholic. In that case, for Drink and Small Sip, the corresponding volume is added to the cumulative intake `totalDrinkMl`. And for Refill and Pour Out, the event is logged but does not change the cumulative intake.

At the same time, the RGB LED provides immediate feedback on the cup body: alcoholic drinking events trigger short flashes (red for standard drink, yellow for small sip), while non-alcoholic events use blue or remain dark to avoid confusion. When `totalDrinkMl` exceeds the predefined threshold `HIGH_ALC_ML_THRESHOLD`, the function `updateRiskLED()` turns the LED to solid red, indicating that the current session has entered a high-risk alcohol level. The firmware also reserves an NTP-based `isQuietHours()` check to optionally suppress LEDs during late-night hours.

5) Event Transmission (JSON Transmission): Once an event is classified (for example, drinking or refilling), the system packages the key data (including event classification, volume change, weight reading, and liquid temperature) into a payload. Then, if the liquid is identified as an alcoholic beverage, this data will be transmitted to the server via an HTTP POST request.

C. IoT communication layer

This part describes how classified events are sent from the ESP32 to the local server and then shown on the web dashboard.

1) Device side: Event-driven HTTP POST upload: The ESP32 connects to the local router in `setup()` via `WiFi.begin()` and stays online afterwards. When the finite state machine finishes an event (e.g. `drink`, `small_sip`, `refill`, `pour_out`), the firmware calls `sendEventToServer()` to send:

```
POST http://<SERVER_HOST>:5000/event
Content-Type: application/json
```

Only events judged as alcoholic by the MQ-3 sensing function are uploaded; non-alcohol events are discarded after local logging. The JSON body contains the event type, volume change and basic context:

```
{"type": "drink | small_sip | refill |
          pour_out",
  "volume": <delta ml>,
  "startWeight": <FSR before>,
  "endWeight": <FSR after>,
  "startMl": <ml before>,
  "endMl": <ml after>,
  "temp": <liquid temperature>}
```

The device does not attach a timestamp; time is added by the server.

2) Server and web dashboard: The backend uses Flask to run the server locally, listening to the `/event` route to receive a single event from the ESP32. After each JSON is received, the event type, volume change, start and end weight and volume, as well as temperature fields are parsed. On the server side, the date, time, and Unix timestamp are uniformly generated, and the event is appended to the memory list on one hand and written to `events.csv` on the other hand, facilitating subsequent debugging and offline analysis. Based on the memory events, the `/day_data.json` route calculates the total alcohol consumption within the current session, the number of `drink` and `small_sip` occurrences, the volume distribution aggregated by hour, and the event interval, etc. These indicators are combined with the preset daily limit `DAILY_TARGET_ML` to calculate the completion ratio, which is provided to the frontend for calling in JSON format.

To reflect risks over a short period of time, the server also provides the `/api/status` interface: it filters `drink` and `small_sip` events within the recent 60-minute window, converts the volume into British alcohol units based on a fixed alcohol content, compares them with the warning threshold and danger threshold, and outputs the risk label and the number of units, which is provided to the front-end for deciding whether to highlight the alert bar. The front-end web page is provided by `index.html` and uses Chart.js to draw charts such as total, cumulative curves, and hourly distribution. The script periodically calls `loadData()` to pull the statistical results from `/day_data.json` and refresh the charts, and also calls `checkStatus()` to access `/api/status` to update the top risk banner text and color. The overall data flow is:

ESP32 - Flask(/event) - Memory/CSV - Flask(/day_data.json, /api/status) - Web dashboard.

IV. Data Presentation and Feedback

A. Visualization of Alcohol Intake Data

The front-end web dashboard is implemented based on the JSON interface provided by Flask. It periodically requests statistical results from the server through the browser and provides a brief visualization of drinking behavior. The main views include the following:

- **Event volume and cumulative trend:** The front end retrieves the event sequence within the current session from `/day_data.json`, and uses Chart.js to draw a bar chart of the volume of individual events and a line graph of cumulative intake. At the same time, the total milliliters for the day and its proportion relative to the daily recommended upper limit `DAILY_TARGET_ML` are displayed in the numeric card and progress bar, facilitating users to

quickly determine whether their overall intake is close to the threshold.

- **Time Distribution and Risk Warning:** Since the timestamps are uniformly generated on the server side, the front end can aggregate the event volume by hour and draw a bar chart to help observe whether drinking occurs mainly during specific periods (such as late at night).
- **Temperature information:** For each event, the temperature reading of the DS18B20 is recorded simultaneously. The dashboard can plot the temperature and volume as a scatter plot to describe the drinking environment (such as chilled or room temperature), providing users with additional background information that does not participate in the risk calculation.

Through these views, the system reorganizes the underlying discrete events into understandable time sequences and environmental cues, laying the foundation for subsequent risk alerts and behavioral analysis.

B. Closed-loop Feedback Mechanism

In order to form a complete behavioral intervention loop, the system achieves dual-channel feedback through ”webpage prompts and local LED indications”:

- **Web-based sedentary reminder:** The front-end script will periodically call `/api/status`. The server calculates the total alcohol intake based on the `drink` and `small_sip` events recorded in the past 60 minutes and compares it with the warning threshold and danger threshold to generate a risk level label. Subsequently, the banner at the top of the webpage will update the text and background color accordingly (for example ”Warning: 2 units in last hour – consider slowing down.”), thereby clearly alerting users to the risk of excessive alcohol consumption in a short period.
- **Local LED prompt:** At the edge end, the ESP32 controls the RGB lights on the cup body based on the event classification results. For the `drink` and `small_sip` events that are determined to contain alcohol, the firmware will trigger a brief flashing as an immediate feedback. When the cumulative intake amount `totalDrinkMl` exceeds the preset threshold `HIGH_ALC_ML_THRESHOLD`, the `updateRiskLED()` function will keep the LED in a continuously bright red state, indicating that the current period has entered a high-risk area. At the same time, the firmware reserves the `isQuietHours()` interface based on NTP time, which can suppress the red light output at night to avoid disturbing the user.

The system provides users with continuous risk awareness and lightweight intervention through two paths: ”server aggregation - network risk banner” and ”edge accumulation - cup body red light”, without forcing them to change their drinking habits.

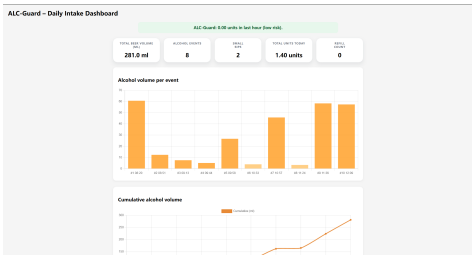

Fig. 3. System Dashboard Overview

V. Results

Based on the real usage data collected during the day, this part does a examination of the system’s output results. The performance of the Flask control panel in displaying statistical data and time patterns, as well as the dependability of event classification, are the main objectives.

A. Daily Summary and Risk Progress

All events generated during the operation of ESP32 will be added and written to the `events.csv` file through Flask. Fields like event type (`event`), starting and ending capacity (`startMl`/`endMl`), capacity change amount (`deltaMl`), temperature (`temp`), and time (`t_ms`). The server will aggregate the `drink` and `small_sip` events into the `/day_data.json` file to obtain the total intake for the current day `total_today`, and calculate indicators such as the number of events, replenishment capacity `refill_ml` and replenishment times. It will also return the preset target capacity `goal_ml` and progress ratio `progress`. The top statistics cards and progress bars on the front end directly use these fields.

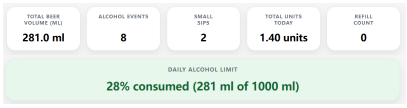

Fig. 4. Daily Summary Dashboard

The statistics cards and bar’s value is consistent with the outcome of manually accumulating based on `deltaMl` when the same dataset is compared in `events.csv`, suggesting that the front-end display and backend aggregation algorithm are in sync.

B. Temporal Patterns and Gap Statistics

Based on the summary results provided by `/day_data.json`, this webpage uses Chart.js to draw bar charts for individual event quantities and line charts for cumulative intake amounts, as well as distribution charts for intake amounts by hour, to visually display the changes in drinking rhythm throughout the day.

Fig. 5. Time-series views of beer-drinking behaviour, including per-event volume, cumulative intake, hourly distribution and drinking intervals

The server will use the timestamp of the `drink`/`small_sip` events to calculate the interval between adjacent events, thereby obtaining two metrics: ”Longest Interval” and ”Most Recent Interval”, and returning this information through the `longest_gap_min` and `last_gap_min` fields. The front end will display this information in the form of ”Today’s Longest Interval: X minutes · Most Recent Interval: Y minutes”, and compare it with the trend of the interval line chart and the actual operation time. This can better reflect the periods of no alcohol consumption for a long time and provide quantitative basis for the subsequent reminder logic. The temperature information `temp` is used to draw a scatter plot of temperature-volume, serving as a supplementary background for the drinking environment (normal temperature/iced, etc.).

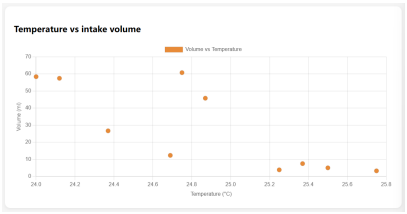

Fig. 6. Scatter plot of water temperature versus per-event volume

C. Event Recognition and System Behaviour

By analyzing the serial port debugging information and the records in the `events.csv` file, the events and overall behavior of the system are evaluated: When a complete ”pick up - tilt - drop” sequence occurs and the volume change $\Delta V > 10$ milliliters, this event will be stably marked as `drinking water`; when ΔV is between 1 and 10 milliliters, it will be classified as `sipping`; when $\Delta V < -10$ milliliters, it will be determined as `refilling` to refill the cup; and when the tilt and peak angular velocity are significantly higher than the normal drinking action, it will be identified as `pouring out`. The latter two types of events will not be included in the total intake. During multiple actual operations, no obvious incorrect judgments such as ”adding alcohol being misjudged as drinking water” were observed, which indicates that the rules based on FSR changes and IMU attitude state machine are relatively stable under the current calibration conditions. At the same time, the ESP32 can correctly trigger the RGB of the cup body after each valid drinking event and light a continuous red light when the cumulative drinking volume exceeds the threshold. The local Flask server did not experience crashes or record omissions during continuous operation for several days, and the front-end charts and statistics cards can automatically refresh with the appearance of new events. This verifies the reliability and consistency of the ”collect - classify - upload - visualize” chain in the desktop scenario.

VI. Discussion

The system has performed stably during 24-hour continuous operation, but it has also exposed some typical limitations of the prototype stage. Firstly, the FSR sensor has nonlinearity and slight zero-point drift. When the cup is stationary for a long time or its position changes, a certain volume error will occur. While piecewise linear calibration has increased the accuracy of the stimation, a more linear weighing solution is more appropriate for long-term use. Second, manually set volume changes, tilt angles, and angular velocities are used as thresholds for event classification. This approach can be used for the project’s cup type and drinking style, but is limited for other containers or user behaviors. Lastly, the system is only appropriate for restricted local area network teaching scenarios because it currently employs the unencrypted HTTP protocol and hardcodes Wi-Fi credentials in the firmware.

VII. Conclusion

This project has verified the feasibility of implementing context-aware alcohol intake monitoring under low-cost hardware conditions. By completing signal smoothing, volume estimation, finite state machine inference, and alcohol gas sensor filtering on the ESP32, the system can stably identify various behaviors such as ”drinking, sipping, refilling, pouring out” locally. It uploads these events, in the form of structured upper-level events rather than continuous raw data, via HTTP POST to the Flask server. Combined with server-side event aggregation and a web dashboard based on Chart.js, the system forms a ”perception - analysis - feedback” loop, allowing users to visually view the total daily intake, imperial alcohol units, hourly distribution, and recent one-hour risk alerts. They can also receive synchronous physical reminders through RGB LEDs, thereby enhancing their self-awareness of short-term excessive drinking risks.

Although the segmented linear analysis has improved the accuracy of pressure estimation, it still fails to meet the standards for long-term deployment. Secondly, event classification relies on manually set volume changes, tilt angles, and angular velocities as thresholds. This method is suitable for the cup type and drinking method in this project, but its universality is limited and it is not easy to be applied to different containers or user habits. Finally, the system currently uses unencrypted HTTP protocol and hard-codes Wi-Fi credentials in the firmware, which is only applicable to controlled local area network teaching scenarios. Therefore, if used in real environments, more secure configurations and encryption mechanisms need to be introduced.

VIII. Acknowledgement

I acknowledge the use of ChatGPT-5 (OpenAI, https://chatgpt.com) to improve the grammatical accuracy and stylistic flow of my original text. I provided my own original drafts and used the prompt: 'below are some sentences from my report and help me refine it to a standard of academic writing.'
I carefully reviewed and edited all the AI's suggested revisions to ensure that they accurately reflected my original ideas and judgments. The final text represents my independent academic work.

References

[1] Marks, K. R., Pike, E., Stoops, W. W., I& Rush, C. R. (2015). Alcohol administration increases cocaine craving but not cocaine cue attentional bias. Alcoholism Clinical and Experimental Research, 39(9), 1823–1831. https://doi.org/10.1111/acer.12824
[2] World Health Organization: WHO. (2024, June 28). Alcohol. https://www.who.int/news-room/fact-sheets/detail/alcohol
[3] Website, N. (2024, August 30). Alcohol units. nhs.uk. https://www.nhs.uk/live-well/alcohol-advice/calculating-alcohol-units
[4] Activity recognition from on-body sensors by classifier fusion: sensor scalability and robustness. (2007b, December 1). IEEE Conference Publication — IEEE Xplore. https://ieeexplore.ieee.org/document/4496857
[5] Edge computing: vision and challenges. (2016, October 1). IEEE Journals I& Magazine — IEEE Xplore. https://ieeexplore.ieee.org/document/7488250
[6] Digital signal processing: Hayes, M. H. (Monson H.), 1949-: Free Download, Borrow, and Streaming: Internet Archive. (2012b). Internet Archive. https://archive.org/details/digitalsignalpro0000haye

Appendix A
Hardware Prototype Components

The following figures illustrate the hardware implementation of the ALC-Guard system, detailing the sensors and microcontroller integration within the prototype.

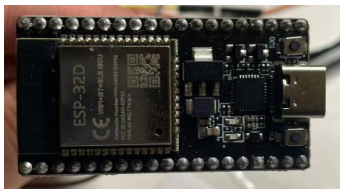

Fig. A1. **Core Processing Unit (ESP32 Microcontroller)**. The ESP32-WROOM-32D development board serves as the central unit for sensor fusion and Wi-Fi communication.

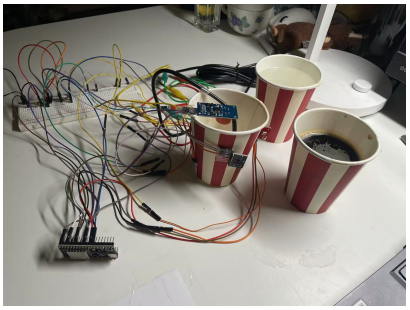

Fig. A2. **Prototype Wiring and Test Environment**. An overview of the experimental setup connecting the ESP32, breadboard, and sensors to the paper cup prototype.

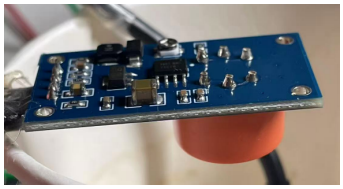

Fig. A3. **Alcohol Gas Sensor (MQ-3)**. Mounted near the cup's rim to detect ethanol vapor concentration for distinguishing alcoholic beverages.

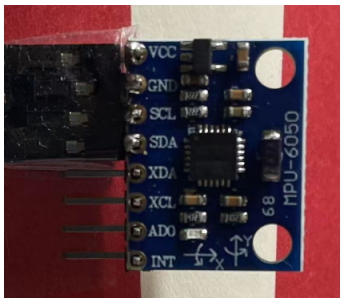

Fig. A4. **Inertial Measurement Unit (MPU6050)**. A 6-axis accelerometer and gyroscope module fixed to the cup wall to monitor tilt angles and angular velocity.

Fig. A5. **Waterproof Temperature Sensor (DS18B20)**. The probe extends into the liquid to monitor temperature changes in real-time.

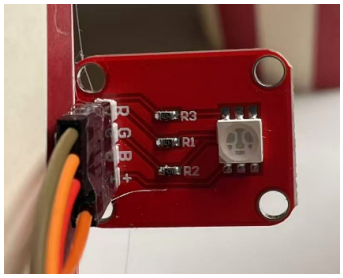

Fig. A6. **Visual Feedback Module (RGB LED)**. A common-anode LED module attached to the cup body for immediate status and risk alerts.

Appendix B
Dashboard Visual Feedback States

The following figures demonstrate the dynamic user interface states on the web dashboard. The system provides real-time feedback through two distinct mechanisms: the Risk Banner (based on alcohol units per hour) and the Daily Limit Progress Bar (based on total accumulated volume).

Fig. B1. **Dynamic Risk Banner States**. Top: **Low Risk** (< 3 units/hr), displayed in green. Middle: **Warning** ($3 - 6$ units/hr), advising the user to slow down. Bottom: **High Risk** (> 6 units/hr), clearly alerting distinct high-risk intake.

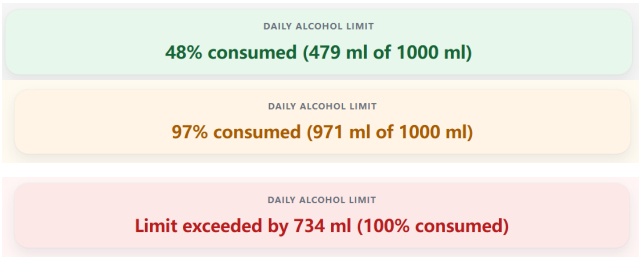

Fig. B2. **Daily Limit Progress Bar States**. Top: **Safe Zone** (Green), user is well within the daily target. Middle: **Cautionary Zone** (Amber), approaching the predefined limit. Bottom: **Limit Exceeded** (Red), visual confirmation that the daily cap has been breached.

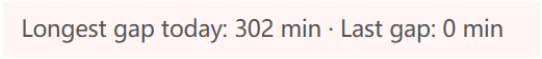

Fig. B3. **Drinking Interval Analytics Card**. This component tracks two key behavioral metrics simultaneously: **(1) Longest Gap**: The maximum rest period recorded today (e.g., 302 min). **(2) Last Gap**: The time elapsed since the most recent drink event (e.g., 0 min), providing real-time pacing feedback.