Centre for Training and Learning

NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL

HANAMKONDA - 506004, TELANGANA, INDIA

INTERNSHIP PROJECT REPORT

on

AIoT-Based Mental Fitness Detection System

Submitted by

K. Saketh Sai Srikar
P. Rahul
P. Ashmith Nanda
Dolphin Pilot

Under the guidance of

Prof. T. Kishore Kumar

Professor, Department of ECE

NIT Warangal

Centre for Training and Learning
NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL
HANAMKONDA - 506004, TELANGANA, INDIA

BONAFIDE CERTIFICATE

This is to certify that this project report entitled **"AIoT-Based Mental Fitness Detection System"** submitted to National Institute of Technology, Warangal, is a bonafide record of work done by **"K. Saketh Sai Srikar, P. Rahul, P. Ashmith Nanda, Dolphin Pilot"** under my supervision from **"15 May 2025"** to **"15 June 2025"**

Supervisor
Prof. T. Kishore Kumar
Professor, Department of ECE
NIT Warangal

Place: Warangal
Date: 15 June 2025

DECLARATION

This is to declare that this report has been written by us. No part of the report is plagiarized from other sources. All information included from other sources have been duly acknowledged. We aver that if any part of the report is found to be plagiarized, we are shall take full responsibility for it.

Place: Warangal

Date: 15 June 2025

Submitted by:

k. Saketh Sai Srikar

P. Rahul

P. Ashmith Nanda

Dolphin Pilot

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to everyone who has supported me throughout the development of this project, Text-to-Speech System with IoT Integration. This journey has been both challenging and rewarding, and it wouldn't have been possible without the guidance and encouragement I received along the way.

I am deeply thankful to my mentor, **T.Kishore Kumar Sir,** for his constant support, insightful feedback, and for always being available to help me navigate through difficulties—both technical and conceptual. His patience and expertise played a key role in shaping the direction of this project.

I would also like to thank the **National Institute of Technology Warangal** for providing the resources and environment that allowed me to bring this idea to life. The access to lab facilities, tools, and technical knowledge has been crucial to this work.

A special thanks to my teammates who offered their suggestions, tested prototypes, and pushed me to keep improving. Your involvement made this project feel collaborative and enjoyable.

To my family thank you for your unwavering support, encouragement, and belief in me even when things got tough. Your love has been my biggest source of strength.

This project has not only enhanced my technical skills but also deepened my understanding of real-time systems, human-centered design, and the value of accessibility and communication technologies. Working on a solution aimed at bridging language barriers and enhancing communication has been both humbling and inspiring.

I hope this project can contribute, even in a small way, to making multilingual technology more accessible and practical in real-world IoT environments. I look forward to building on this foundation with further innovations.

CONTENTS **PAGE.NO**

LIST OF FIGURES

AIM AND OBJECTIVES

Aim:

The aim of this project is to develop and deploy an AIoT-based Mental Fitness Detection System that continuously monitors different physiological parameters using real-time sensors connected with an ESP32 microcontroller to eventually determine the mental fitness level of the person. The system employs a machine learning model deployed in a small embedded form factor to predict whether a person is mentally Fit or Unfit to perform activities like driving, operating machinery, or making important decisions. The system is not only for alcohol detection; it also considers stress, anxiety, and other physiological abnormalities that can influence mental focus and decision-making capabilities.

Objectives:

- This project is to develop an AIoT-based system that detects a person's mental fitness in real time using physiological sensor data such as alcohol level, heart rate, SpO_2, GSR (stress), and reaction time. The goal is to predict whether an individual is mentally fit or unfit to perform tasks like driving or operating machinery under the influence of alcohol or other medical conditions like stress or anxiety.

- To couple various biosensors (MQ-3, MAX30102, and GSR) for detection of alcohol, heart rate measurement, SpO_2 monitoring, and stress assessment.

- To analyze sensor data on an ESP32 microcontroller in real time.

- To predict the state of mental fitness in real-time

- This system that analyzes various physiological and cognitive parameters to determine whether a person is mentally fit or not under the influence of alcohol.

- Accurately quantify Blood Alcohol Content (BAC) levels and their potential cognitive impact using the MQ-3 sensor.

- To provide visual and wireless notifications on an webpage with a Dashboard.

- To enhance driver safety by detecting impairment by means of multi-sensor fusion and real-time notification.

- In order to make the system low-cost, portable, and deployable in a wide range of real-world applications including transportation safety, industrial control, and healthcare.

INTRODUCTION

In the competitive world we live in today, mental fitness is as crucial as physical health—especially in jobs where decision-making, coordination, and vigilance are involved. Physical health is very much sought after, but mental fitness is oftentimes neglected despite the fact that it influences an individual's ability to manipulate machines, drive vehicles, or make proper decisions. It is even more crucial in safety-critical environments like transport, industry, police, and public services.

The traditional alcohol detecting devices such as the breathalyzers detect alcohol alone but not the other equally significant impairments such as stress, anxiety, oxygen level, and physiological tiredness. An individual may be sober but unfit mentally since he is under immense stress, has not slept appropriately, or has abnormal vital parameters. Therefore, an efficient system is required—one that takes into account several physiological and psychological parameters.

This work proposes a low-cost, pervasive, and real-time AIoT-based Mental Fitness Detection System. It is an Artificial Intelligence (AI) and Internet of Things (IoT) integration for sensing and examining different biometric signals using sensors connected to an ESP32 microcontroller. The system works based on real-time data from:

- An MQ-3 sensor to detect alcohol content in breath,
- A single MAX30102 sensor for heart rate and blood oxygen level (SpO_2) detection, and
- A GSR sensor (Galvanic Skin Response) to analyze skin conductivity, which is related to anxiety and stress.

The sensor readings are transmitted to a local server through Wi-Fi with the help of ESP32, and preprocessed and processed by a trained Machine Learning (ML) model hosted using Flask or embedded with TensorFlow Lite / Edge Impulse. The model determines the mental health fitness status of the individual as "Fit" or "Unfit" based on patterns learned from training data.

For visualization of results and instant decision-making, an online dashboard (created with React and Flask) displays:

- Real-time sensor readings,
- Expected mental state,
- Interactive graphs of physiological parameters.

This system can enhance safety, prevent accidents, and assist healthcare screening through a smart tool of mental fitness assessment. It is particularly crucial to drivers, workers in hazardous working environments, stressed students, or even patients with chronic diseases.

By moving beyond alcohol monitoring and considering a wider set of mental fitness indicators, this project unlocks the potential for future-proof personal health and safety monitoring solutions based on low-power, low-cost real-time AIoT systems.

LITERATURE REVIEW

Over the past few years, the integration of Internet of Things (IoT), embedded systems, and biometric monitoring has resulted in the development of smart health and safety applications—particularly in the field of real-time operator and driver monitoring. Increased research focuses on multi-sensor fusion methods, embedded decision-making, and wireless communication for measuring human readiness in high-risk environments like driving, industrial operation, and military activities.

1. Biometric Sensing in Safety Systems:
   Experiments have shown the effectiveness of employing biometric sensors like heart rate, SpO_2, electrodermal activity, and alcohol detection to recognize fatigue, stress, or drunkenness. Sensors such as the MQ-3 gas sensor are used extensively in low-cost breathalyzer designs, which can sense alcohol content from exhaled breath. Likewise, MAX30102 sensors have proven useful in wearable health applications, giving real-time feedback on pulse rate and oxygen saturation—both crucial measures of physical state and alertness. GSR (Galvanic Skin Response) sensors are employed in psychological research and affective computing to assess levels of stress in terms of skin conductivity.

2. IoT Platforms and Edge Devices for Real-Time Processing:
   Microcontroller-based computing platforms such as the ESP32 and Raspberry Pi have dominated the integration of real-time acquisition and processing of data from multiple biosensors. Their Bluetooth and Wi-Fi support, low power usage, and GPIO flexibility make them suitable for edge-level AIoT applications. Various research articles have documented successful applications of these devices in wearable health monitors, alcohol detection devices, and fatigue-related accident early warning systems.

3. Multi-Sensor Integration for Improved Decision Making:

   Theory substantiates that the integration of multiple sensor modalities results in greater reliability and lower false positives in condition detection. Although individual-sensor systems (such as standalone breathalyzers) cannot compensate for user-specific variability, multi-parameter fusion enables the system to consider varied physiological dimensions, enhancing overall decision-making. For example, an individual might possess a BAC within the legal limit but be physiologically or mentally incapable of driving—something detectable only through integrated heart rate and stress analysis.

4. Communication Interfaces and Alerting Mechanisms:

   Contemporary safety systems are focusing more on wireless communication and real-time notification. Most projects employ mobile apps through platforms like Blynk, MQTT, or bespoke TCP/IP implementations for the transmission of warnings from microcontrollers to operators or authorities. Visual indication through OLED screens and automatic warning messages guarantee prompt perception of harmful conditions, thereby boosting individual and environmental safety.

5. Challenges and Future Research Directions:

   In spite of the evolution in driver monitoring through biosensors, there remain a number of challenges. Accuracy is susceptible to sensor noise, motion artifacts, personal physiological variability, and environmental interference. In addition, scalability and energy efficiency are areas open for optimization. Future research directions include incorporating machine learning models for adaptive thresholding, cloud computing for long-term trend analysis, and system augmentation for enabling fatigue detection, facial recognition, and autonomous vehicle interface.

In summary, the intersection of AI, IoT, and biosensing technologies offers a paradigm-shifting chance to reengineer driver safety and health monitoring. Ongoing innovation in multi-sensor fusion, embedded intelligence, and wireless communication is essential in creating smart, context-sensitive systems that avoid accidents and encourage safe behavior on and off the roads.

GAPS IDENTIFIED

1. Single-Parameter Fitness Detection (for example, Alcohol Only)

These existing systems, for instance, breath analyzers, can only identify alcohol consumption.

Gap: They don't check general mental fitness — someone may be sober but mentally ill due to stress, anxiety, exhaustion, or other illnesses.

2. Absence of Real-Time Multi-Vital Monitoring

The majority of mental health evaluations are either manual (questionnaires, interviews) or clinic-based.

Gap: No real-time, automated, ongoing monitoring solution for monitoring a set of vitals such as heart rate, oxygen saturation, skin conductivity, etc., that reflect mental status.

3. Lack of Integration Between IoT and Mental Health Evaluation

Current wearable technology such as smartwatches and fitness bands primarily target physical fitness.

Gap: There is minimal or no AI + IoT integration for mental health and decision-making capacity detection, particularly for everyday routines such as driving or working.

4. No Low-Cost AI-Based Mental Fitness Detector

Mental tools and EEG equipment (like Muse headbands or brain sensors) are expensive and not portable.

Gap: There is no affordable, microcontroller-based, AI-driven device available to ordinary users, particularly in third-world nations.

5. Limited Identification of Contextual Adaptability

There is no context-sensitivity in current systems; they do not assess a person's immediate ability to perform a task (e.g., driving or equipment operation).

Gap: There is no situational awareness mechanism that assesses the current moment mental fitness.

6. No Preventive Warnings or Visualization Dashboards

The majority of detection devices (if any) lack predictive analysis, data visualizations, and alerts to avoid abuse of machines or vehicles.

Deficiency: Lack of real-time dashboards, live data visualizations, or notification systems to pass information to managers or users.

DESIGN METHODOLOGY

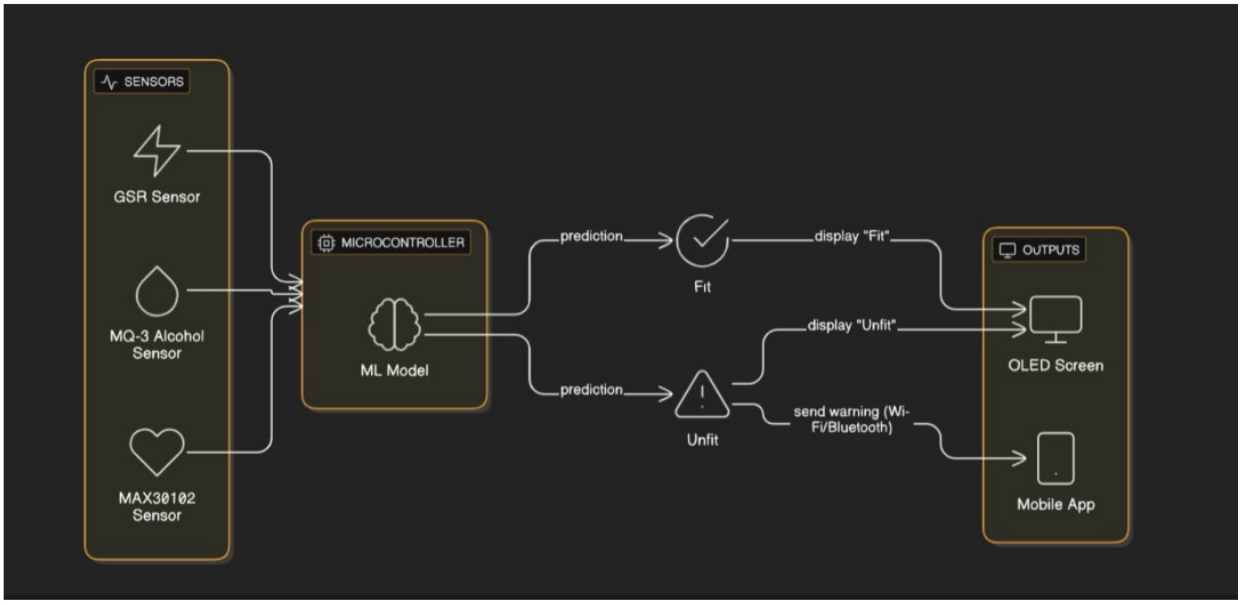

Fig1 Project Methodology

The project integrates AI (Machine Learning) and IoT (sensors and ESP32) for monitoring several physiological parameters in real-time and assessing the mental fitness state of a person. The methodology is divided into several sequential steps:

1. Sensor Integration & Data Acquisition

Purpose: To gather current physiological and environmental information.

Parts Used:

MQ-3 sensor for alcohol sensing (Breath test).

MAX30102 sensor for heart rate (HR) and SpO_2 (oxygen saturation).

GSR sensor of skin conductivity that fluctuates with stress or anxiety.

Platform: ESP32 is interfaced with all the sensors, and it is both an IoT node and a microcontroller.

Sampling Rate: Sensor data are collected at regular intervals (e.g., 2–5 seconds).

2. Backend Data Transmission

ESP32 sends the readings of the sensor through serial communication (USB) or Wi-Fi to Flask backend.

Data is sent in JSON format via REST API or MQTT (for sophisticated IoT applications, but not required).

3. Data Preprocessing

Backend Flask Server responsibilities:

Standardization of inputs based on pre-computed mean and standard deviation (loaded from scaler.pkl).

Dealing with Missing or Noisy Data (optional: apply filtering such as moving average or smoothing).

4. Machine Learning Methods for Predicting Mental Well-being

Pre-trained ML model (such as shallow neural network or SVM/Random Forest) is used.

The model normalizes the sensor inputs and processes and generates:

Mental Fitness Status: Fit or Unfit.

Possible cause: Alcohol, Stress, Low Oxygen, Multiple Factors.

5. Real-time Dashboard Visualization

A React + Tailwind CSS frontend (web dashboard hosted) renders:

Real-time graphs of Heart Rate, SpO_2, GSR, and Alcohol level.

Mental fitness status indicator (Color-coded: Fit = Green, Unfit = Red).

Record of past measurement and prediction history (Optional: in local database or JSON file).

Flask backend exposes prediction output via an endpoint such as /predict and live update endpoint /latest.

7. Alert System and Decision-Making According to prediction:

If status is Unfit, alert messages are shown (or buzzer/LED triggered in hardware configuration).

The design approach used here is a system-based and modular one to allow for seamless integration of hardware, software, machine learning, and user interface modules. The process starts with the determination of physiological and environmental parameters best representing a person's mental health. In this instance, alcohol content (determined by the MQ-3 sensor), pulse rate and blood oxygen saturation (determined by the MAX30102 sensor), and electrodermal activity (determined by the GSR sensor) were chosen because of their proven association with intoxication, stress, and anxiety states. These sensors are interfaced to the ESP32 microcontroller, which was chosen for its onboard Wi-Fi, low power consumption, and ability to handle multiple analog and digital inputs. After sensor calibration and testing, the ESP32 is programmed to record sensor readings at pre-programmed intervals and wirelessly send these data in the JSON format to a backend server.

Backend infrastructure is set up using Flask, a lightweight but scalable Python web framework, as the processing core. The server plays a few roles: it gets real-time sensor readings, preprocesses the readings (e.g., by applying standardization with pre-computed scaler values), sends it to a trained machine learning model, processes the resulting outputs, and sends the output (e.g., "Fit" or "Unfit") to the requesting interface. Offline, the model is trained using a dataset of labeled samples of physiological readings and their corresponding mental fitness states. During design, several algorithms such as Support Vector Machines (SVM), Random Forests, and shallow neural networks were experimented with depending on their accuracy, latency, and resource usage. The selected model is either deployed within the backend to facilitate faster inference or exported as a .tflite model to facilitate lightweight deployment. The user interface is built with React.js and Tailwind CSS, prioritizing simplicity, clarity, and live responsiveness. The frontend design particularly prioritizes the display of real-time vital signs, time-series plots for trend detection, and a well-prominent area for mental fitness predictions. The dashboard constantly communicates with the backend via the /latest endpoint to update sensor data and refresh predictions. In addition, it visually marks risk factors like excessive alcohol levels, abnormal heart rates, or excessive GSR readings, all of which indicate stress or intoxication. Design considerations also touch upon mobile responsiveness and accessibility, thus ensuring the use of the system in field settings or integration into industrial settings.

In effect, the design approach is based on a modular development pattern—sensor inputs as the input layer, ESP32 as the edge gateway, Flask as the backend processor, and the web dashboard as the output and visualization layer. Each module is independently developed with documented APIs, allowing easy troubleshooting, scalability, and potential integration of future features like cloud storage, SMS notification, or AI-powered health advice. The entire system is a real-time feedback loop where data acquisition, processing, and response are inseparable to provide immediate evaluations of mental fitness. It not only ensures functional correctness but also promotes user trust and reliability, both of which are critical in human health and safety applications.

FLOW CHART

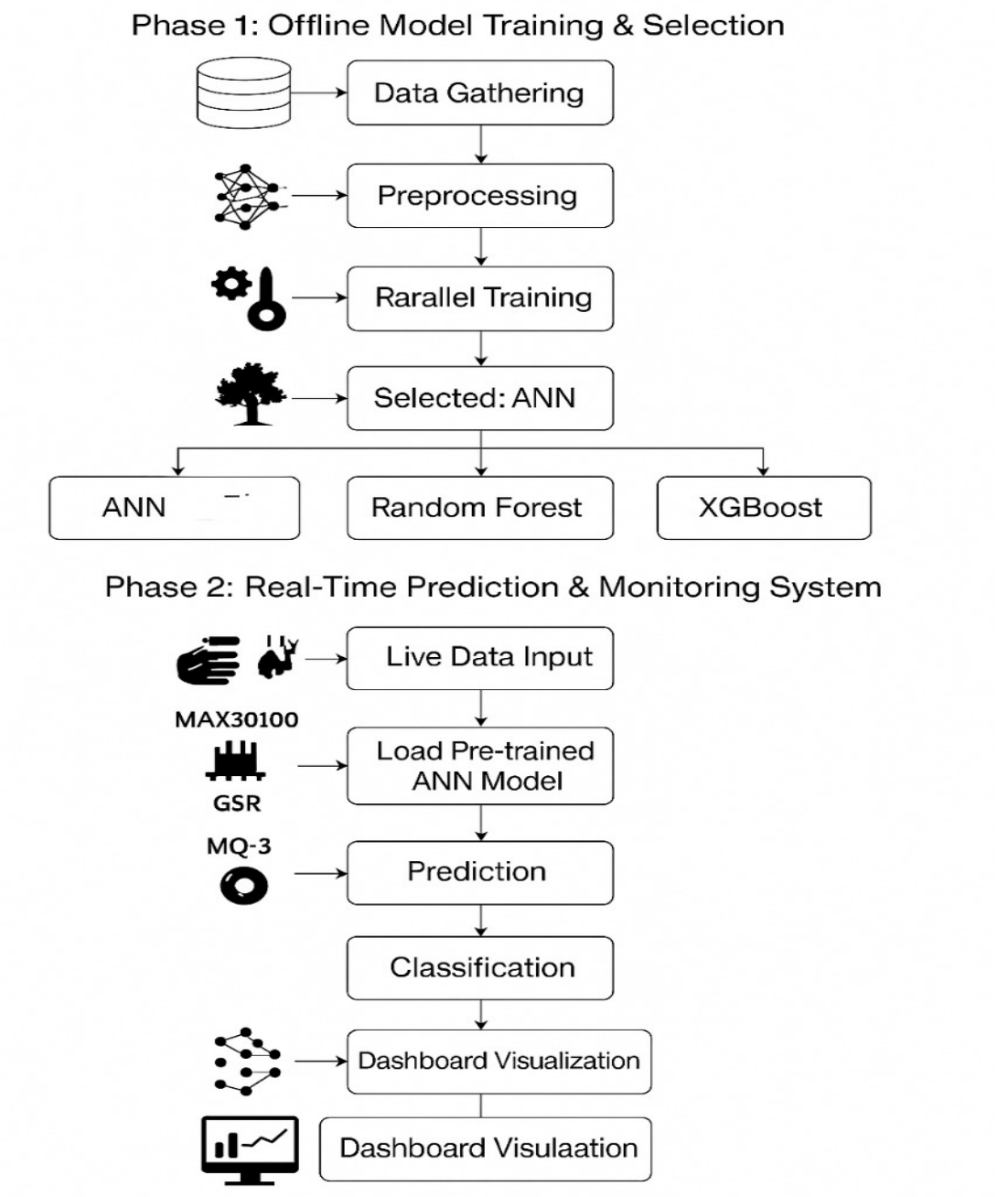

Fig2. Flowchart

DETAILED STEPS OF IMPLEMENTATION

Step 1: Integration of Hardware and Sensor Setup

The first step entails choosing and interfacing the appropriate sensors with the ESP32 microcontroller. The aim is to capture physiological and chemical readings that indicate the mental and physical state of the user.

We employ the MQ-3 Alcohol Sensor class to identify the presence and rough concentration of alcohol from the breath. The sensor reading is analog, i.e., it takes on continuous values according to the quantity of alcohol sensed. It is attached to one of the analog input pins of the ESP32. To use it effectively, a warm-up time of 20–30 seconds is required to stabilize the sensor. Calibration is important to translate voltage levels into alcohol concentration accurately. This is usually achieved through comparisons of sensor output with known BACs in controlled environments.

The MAX30102 sensor also offers heart rate and SpO_2 (blood oxygen level) information. It employs infrared and red LEDs, and photodetectors to estimate the same. The sensor is interfaced with the ESP32 using the I2C bus, needing SDA and SCL pins to be used. A library (e.g., SparkFun's MAX3010x or Adafruit MAX30105) is employed to read and decode the raw sensor signal. For increased accuracy, one needs to eliminate noise and reject outliers due to movement or wrong finger placement.

The GSR Sensor is used to detect skin conductance, which rises in the state of stress, anxiety, or excitement. The sensor also generates analog readings that are input into another analog pin of the ESP32. Larger readings generally translate into greater sweat gland activity, correlating with more arousal or stress. Raw readings are normalized later on before they are fed to the ML model.

The ESP32 is the central processing unit, where all the sensor values are summed. The data are read at set intervals (e.g., every 1–5 seconds) and temporarily stored for processing or transmission. Proper power management is provided by a stable 3.3V supply or USB. Pull-up resistors and decoupling capacitors can be included for signal stability enhancement. Code is loaded to the ESP32 via the Arduino IDE, while sensor libraries relevant for the application are imported.

In short, this step sets the foundation data pipeline from the physical world to digital form, where all the sensors are properly wired, configured, and programmed to gather real-time physiological data. With sensors up and running and providing data, we proceed to the next stage—communication and transmission.

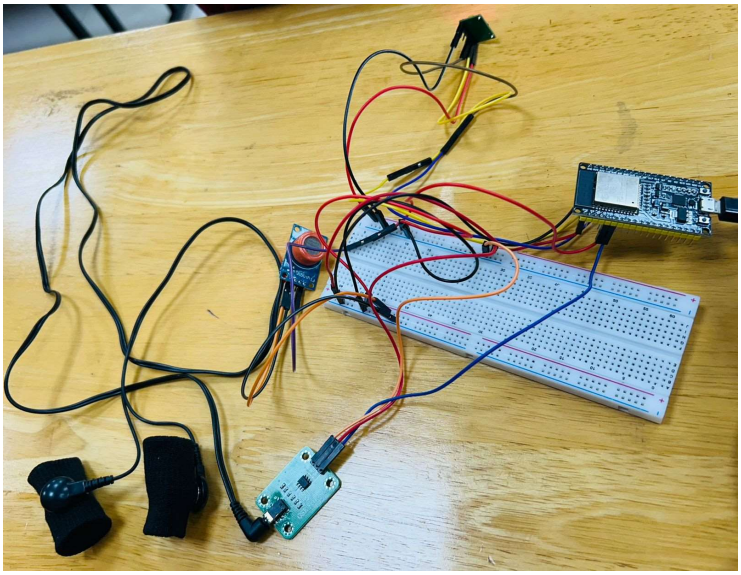

Fig3. Sensor Setup

Step 2: Wireless Communication and Data Transmission

Once the data acquisition from the ESP32 and sensor integration are set, the following important step is to send this data to the backend server (Flask application) for analysis and processing. This will make sure that real-time communication between the edge device (ESP32) and the web-based mental fitness dashboard is maintained.

ESP32 has Wi-Fi functionalities built into it, which allows it to be a smart IoT device and wirelessly transmit data over a local or cloud network. To begin with, the ESP32 has to get connected to a Wi-Fi network, and this is done by storing SSID and password credentials in the ESP32's sketch code. After connecting to the Wi-Fi network, the ESP32 can access external servers using normal web protocols (such as HTTP).

We program the ESP32 to package sensor readings (alcohol level, GSR, heart rate, SpO_2, and reaction time) into a JSON object, which is a widely accepted lightweight data format. A POST request is made to a specific endpoint (e.g., /predict) on the Flask server using the HTTP client libraries in Arduino, such as HTTPClient.h.
On the Flask backend, the endpoint takes this data, parses it, and preprocesses it—usually normalizing via precomputed scaler values (from a .pkl file). The data is now prepared to be input to the trained machine learning model.

To be reliable and secure over transmission:
Retry mechanisms are used on the ESP32 in the event that the server is not accessible.
Simple token-based authentication can be employed in order to avoid injecting unauthorized data.
Timestamps can also be added with every payload so that data recording times can be monitored.
Live or near-live communication is ensured by ensuring the POST interval remains short (1–5 seconds), which provides live updates on the web dashboard without saturating the server or device.

This step ties together the hardware and software parts of the system. Without reliable and efficient data transmission, the model would lack input to process, and this is an essential step in the implementation pipeline. After data arrives at the Flask backend, machine learning inference is the next step.

Step 3: Machine Learning Model Prediction and Inference

Once the ESP32 sends live sensor readings to the Flask backend, the subsequent crucial step is conducting machine learning inference to classify the user's mental fitness status. This process is the analytical heart of the system and is tasked with taking raw physiological measurements and converting them into actionable insight.

The Flask server accepts a JSON payload with several sensor parameters like alcohol concentration (from MQ-3), GSR (conductance of the skin), heart rate, SpO_2 (from MAX30102), and reaction time. These parameters are chosen with care as they are related to states of mind like stress, tiredness, drunkenness, and anxiety.

When receiving the data at the /predict endpoint, there is preprocessing done by the backend. Preprocessing often involves data standardization or normalization, where the input features are put within the range that was seen during model training. As an example, mean and standard deviation statistics calculated from the training dataset are utilized to scale live input accordingly. The values are either hard-coded or imported from a saved .pkl scaler file.

The preprocessed input is next fed into a learned machine learning model. For your project, the model could be a light neural network, Random Forest, or Support Vector Machine (SVM) model learned from a labeled set of physiological and behavioral data. The set would consist of sets of sensor readings labeled as "Fit" or "Unfit" based on the alcohol presence, high stress, abnormal heart rate, low SpO_2, or slow reaction time.

In the production-ready version, you're using a .tflite model (TensorFlow Lite format), which is optimized for edge inference but here runs on the Flask backend for simplicity and flexibility. The model's output is typically a binary or categorical label—like:

0 → Unfit
1 → Fit
The backend processes this output and sends a response back to the frontend in real time. This output includes:
The prediction (Fit/Unfit)
Sensor values used for inference
Other scores such as decision confidence or tolerance level

This prediction can also be used as a decision-making tool, particularly in situations such as driving, industrial safety, or warfare. If the outcome is "Unfit", suitable alerts or suggestions (e.g., "Do not use machinery") can be prompted in the dashboard.

Additionally, this step provides scalability: with more data being added, the model can be retrained or updated from time to time to achieve greater accuracy. This enables long-term deployment in actual applications.

This finishes your intelligent decision layer of the project and prepares the way for the subsequent step, which includes visualization and user interface updates within the dashboard.

Step 4: Dashboard Visualization and Real-Time Monitoring:

After the prediction (Fit/Unfit) is done by the machine learning algorithm in the Flask backend, the system then goes ahead and presents all applicable information to the user within an information-rich and responsive web-based dashboard. This is an important step within the project because it brings the backend analytics together with human-readable output so that it can be used by both technical and non-technical stakeholders.

The dashboard is generally developed in React with TypeScript (or vanilla JavaScript), styled with Tailwind CSS, and optionally bundled with Vite for rapid development. The frontend communicates perfectly with the Flask backend by calling HTTP requests to endpoints like /predict or /latest at intervals or in real-time via technologies like fetch, axios, or WebSocket (for advanced versions).

When. the data is received from the backend, the dashboard does the following:
It updates the real-time sensor readings (Alcohol level, Heart Rate, SpO_2, GSR, and Reaction Time) in specific UI elements. It displays the current mental fitness status as forecast by the ML model (e.g., Fit or Unfit) with high-visibility visual indicators such as colored badges, icons, or status texts.It keeps a record of recent readings and forecasts, displaying timestamped entries to track trends over time. Another vital aspect of this step is the depiction of physiological parameters in the form of dynamic and interactive graphs or charts. Chart.js, Recharts, or D3.js are employed to present:

Heart rate variation over time
SpO_2 percentage
Alcohol sensor reading
GSR peaks that reflect stress levels
Reaction time as a bar or line chart

These charts update in close to real-time via data polling or WebSocket messages. This gives users real-time feedback on how their physical and mental state is improving.There is also a decision summary area on the dashboard, which analyzes the prediction and gives suggestions like:

"You are mentally fit to continue."
"Unfit: Alcohol found. Wait and retest."
"Stress detected: Take a deep breath before continuing."

For improved usability, the interface is responsive and can be accessed through a local IP address (e.g., http://192.168.x.x:5173) on mobile or PC devices plugged into the same network as the ESP32. This makes it portable and deployable in real-world applications.

This action takes the system back full circle, making difficult sensor data and AI decisions comprehensible and actionable in real-time. It also enables feedback loops, enabling users to take corrective action based on their mental state.

Step 5: Testing, Validation, and Optimization
Having developed the end-to-end system from sensor acquisition on the ESP32 to AI-driven prediction on the Flask backend and real-time visualization on the dashboard, the next critical step is testing, validation, and optimization. This is a critical step that guarantees the system performs as expected, is accurate, easy to use, and ready for real-world deployment.

System Testing
The first half of this phase is testing the whole pipeline under a multitude of simulated and real use-case scenarios. The system is put through an array of input values from real human subjects or controlled test environments.

These are:

Low, medium, and high concentrations of alcohol
Physiological fluctuation like normal and abnormal heart rates
Different SpO_2 values, particularly in stressful situations
Diverse reaction times depending on user attention or distraction
GSR spikes mimicking stress or anxiety

Every test case is utilized to test whether the system always offers the right prediction and correctly updates the frontend dashboard. Unit testing (for each component separately such as Flask endpoints or ML predictions) as well as integration testing (to check smooth ESP32 → Flask → Dashboard data flow) is performed.

ML Model Validation:-

Model performance is tested using metrics such as:
Accuracy: Number of times correctly predicting "Fit" or "Unfit".
Precision and Recall: Particularly crucial when it is unsafe to make false negatives (predicting "Fit" when the person is unfit).
Confusion Matrix: In order to examine misclassifications.
ROC-AUC: For detecting the trade-off between true positives and false positives.

Validation is performed on a test set that the model never saw during training. If necessary, cross-validation is employed to guarantee robustness. Generalization can be enhanced by collecting data augmentation methods, e.g., via noise injection or by synthesizing data.

Step 6: Deployment and Demonstration of Use Case

Following the creation and thoroughly testing the hardware, software, and AI aspects, the last step is deployment—placing the project into a simulated or real-world environment—and performing live demonstrations to validate its usability, usefulness, and extent. This stage closes the engineering design-to-practical-application gap.

Real-Time Deployment Setup:-
The entire system is now set up for real-world deployment:
The ESP32 board, which is interfaced with the sensors (alcohol: MQ-3, heart rate and SpO_2: MAX30102, skin resistance: GSR), is powered either by USB or battery.

The ESP32 firmware is flashed and Wi-Fi connected and transmits sensor readings to the Flask backend at regular intervals. The Flask server is run locally (on a laptop or Raspberry Pi) or hosted on a cloud environment (such as PythonAnywhere or AWS EC2) for remote accessibility.

The dashboard, constructed with React/TypeScript (or basic HTML/CSS if easier), pulls data from the backend's endpoints such as /predict and /latest to display live charts, sensor readings, and the final prediction (Mentally Fit / Unfit).
This entire stack system allows users to track their physical and mental status real-time, with predictions refreshed every few seconds.

Use Case Demonstrations:-

The system deployed is tested on a number of real-world or classroom simulation scenarios to demonstrate its worth and potential:

Use Case 1: Alcohol-Influenced Mental Impairment
A user blows into the MQ-3 sensor after having a small dose of alcohol. As the BAC value rises, the ML model starts including it in the overall fitness score. If the reaction time is also slowed, the system marks the individual as "Unfit," proving the impact of alcohol on cognitive performance.

Use Case 2: Stress or Anxiety Detection
A stressed individual is expected to show abnormal GSR responses and an elevated heart rate. Even in the absence of alcohol, the ML model identifies imbalanced physiological states and can render a "Unfit" prediction. This indicates the way that the system does not single out BAC alone but also factors in medical states such as stress or anxiety.

Use Case 3: Baseline Normal Condition

When the sensors recognize normal values—stable heart rate (~70–90 bpm), high SpO_2 (>95%), low alcohol level, low GSR (relaxed status), and fast reaction time—the system classifies the user as "Mentally Fit." This situation validates the model's accuracy in detecting healthy states.

Final Demonstration Features:-
The demonstration environment includes:
A live dashboard presentation of a real-time stream of sensor data and varying predictions.
Plots graphing vitals over time to monitor physical response patterns.
Color-coded fitness status (green for fit, red for unfit).
Being able to test diverse individual or combined test cases within less than 2 minutes.

Readiness for Real-World Integration

- Vehicles (prior to ignition, identify alcohol/cognitive state of drivers)
- Industrial machines (verify fitness of machine operators)
- Military or critical operations (detect stress-induced impairment)
- Medical pre-screening (mental status tracking in emergency departments)

The AIoT system thereby establishes itself as applicable in real-world scenarios not only for alcohol detection, but as a generalized framework for monitoring mental fitness.

PCB LAYOUT

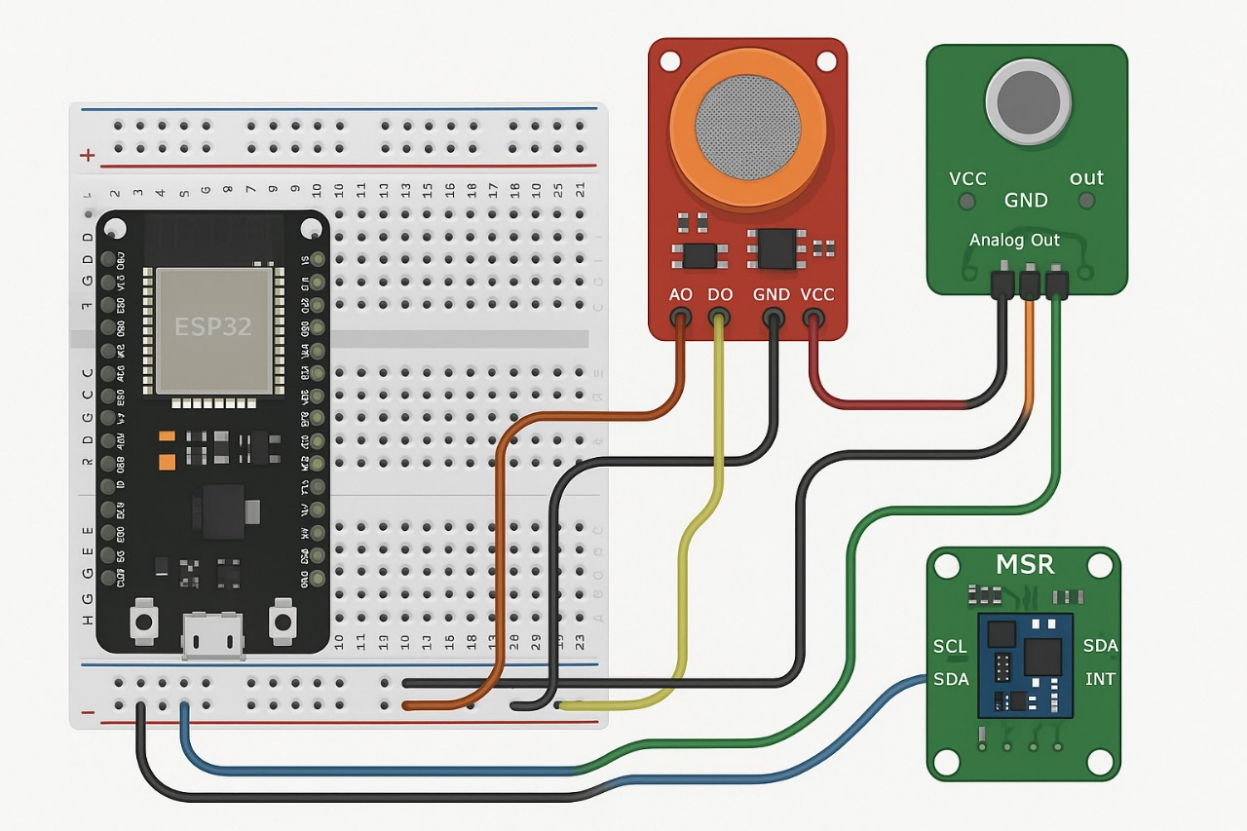

Fig4. Hardware Integration

SUMMARY ON PCB LAYOUT:

The given PCB layout in the image is an embodiment of the hardware part of your AIoT-based Mental Fitness Detection System under the influence of alcohol and other medical conditions such as stress and anxiety. The configuration is well laid out around the ESP32 microcontroller, which serves as the heart of the project. It accepts input signals from various biomedical and environmental sensors and transfers the processed signals to the software or cloud interface.

Central Unit: ESP32
The central unit of the design is the ESP32 microcontroller, which is placed on a breadboard. It offers both analog and digital input/output pins and is tasked with reading sensor data, processing it, and transferring it to a server or dashboard. The ESP32 enjoys a reputation for embedded Wi-Fi and Bluetooth functionality, making it ideal for real-time monitoring and IoT applications.

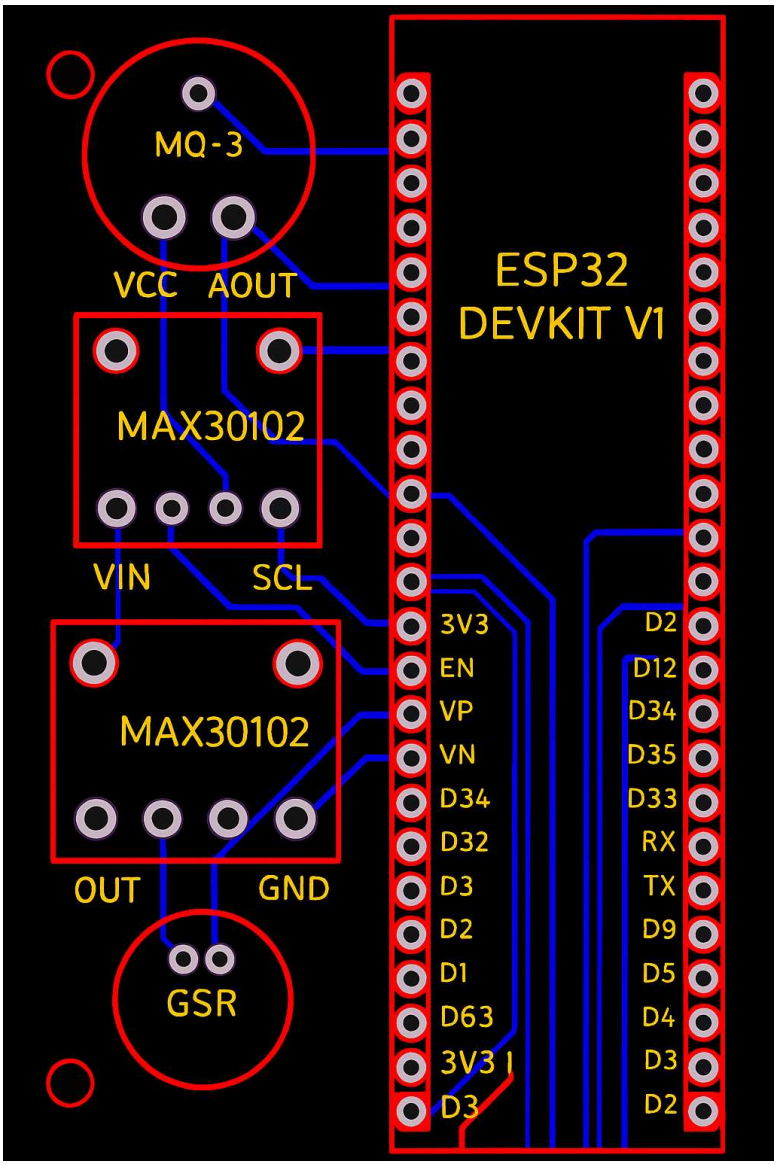

Fig5. PCB Layout

Sensor Connections

There are three main sensors in this arrangement:

MQ-3 Alcohol Sensor (Red PCB): The sensor monitors the concentration of alcohol in the breath and communicates with the ESP32 through its analog output (AO). The VCC and GND are bridged to the 3.3V power supply and ground respectively, and the AO pin is connected to one of the analog input pins of the ESP32. The alcohol signal can be monitored continuously in ppm.

Pulse Rate/Heart Rate Sensor (Green PCB): It is employed to detect the user's heart rate, an essential parameter indicating mental stress or anxiety. Its output is fed to one of the

other analog inputs of the ESP32. Similar to the MQ-3, it is powered from the 3.3V rail and uses the same ground connection.

GSR (Galvanic Skin Response) Sensor (MSR Module): The MSR module is employed for stress level assessment through measuring perspiration-induced skin conductance changes. The MSR sensor communicates via I2C communication on the SDA and SCL pins to the ESP32. It is an essential module for detecting alcohol-free mental fitness problems such as anxiety or emotional stress. The INT pin may also be utilized for interrupt-based data retrieval, but it is not determined by code logic.

Power and Ground Distribution:-
All sensor modules take power from the same 3.3V rail (red line on the breadboard) and are connected to the GND rail (black line) for grounding. They must be well grounded in this system so that readings from the sensors are reliable and stable. The power rails are brought forward horizontally along the top and bottom strips of the breadboard so that they can be easily accessed by all the components connected.

Communication and Signal Flow:-
All the sensors send their measurements to the ESP32 either via analog pins or I2C digital communication. The ESP32 then processes these readings and applies pre-trained machine learning algorithms (which it has stored in its memory or which are transmitted to a cloud-based model) to determine the user's mental fitness state as either "Fit" or "Unfit". This judgment is based on a weighted aggregation of alcohol levels, heart rate variability, and GSR readings.

PROGAM AND PROJECT FILES

```
#include <Wire.h>
#include "MAX30100_PulseOximeter.h"

// Sensor Pins
#define MQ3_PIN 34     // MQ3 Alcohol Sensor
#define GSR_PIN 35     // GSR Sensor
#define REPORTING_PERIOD_MS 5000

PulseOximeter pox;
uint32_t tsLastReport = 0;

// Beat callback
void onBeatDetected() {
  Serial.println("♥ Beat detected!");
}

void setup() {
  Serial.begin(115200);
  delay(1000);
  Serial.println("Initializing...");

  // Start I2C with ESP32 default I2C pins
  Wire.begin(21, 22);

  // MAX30100 Init
  if (!pox.begin()) {
    Serial.println("❌ MAX30100 not found!");
  } else {
    Serial.println("✅ MAX30100 initialized.");
    pox.setIRLedCurrent(MAX30100_LED_CURR_11MA);
    pox.setOnBeatDetectedCallback(onBeatDetected);
  }

  pinMode(MQ3_PIN, INPUT);
  pinMode(GSR_PIN, INPUT);
}

void loop() {
  pox.update();  // Read pulse oximeter

  if (millis() - tsLastReport > REPORTING_PERIOD_MS) {
    tsLastReport = millis();

    float bpm = pox.getHeartRate();
```

```
42        tsLastReport = millis();
43
44        float bpm = pox.getHeartRate();
45        float spO2 = pox.getSpO2();
46
47        int alcoholRaw = analogRead(MQ3_PIN);
48        float alcoholVoltage = alcoholRaw * (3.3 / 4095.0);
49        float alcoholLevel = alcoholVoltage / 3.3;
50
51        int gsrRaw = analogRead(GSR_PIN);
52        float gsrValue = gsrRaw / 4095.0;
53
54        Serial.println("----- Sensor Readings -----");
55        Serial.print("Heart Rate (BPM): ");
56        Serial.println(bpm);
57        Serial.print("SpO2 (%): ");
58        Serial.println(spO2);
59        Serial.print("Alcohol Level (Normalized): ");
60        Serial.println(alcoholLevel, 6);
61        Serial.print("GSR (Normalized): ");
62        Serial.println(gsrValue, 6);
63        Serial.println("-----------------------------\n");
64    }
65  }
66
```

The Arduino IDE code is constructed to be executed on an ESP32 microcontroller, which acts as the central hardware for communicating with different biomedical sensors in the Mental Fitness Detection System. The main goal of this code is to obtain real-time physiological signals from the MQ-3 alcohol sensor, MAX30102 pulse oximeter for heart rate and SpO_2, and a GSR (Galvanic Skin Response) sensor, and send these signals to a backend system for subsequent processing and mental fitness prediction.

At the top of the code, necessary libraries are imported in order to enable communication with the sensors. These are I^2C libraries and sensor-specific libraries like Wire.h, Adafruit_Sensor.h, and possibly MAX30105.h or MAX30102.h, depending on the sensor module. The sensors are wired to certain GPIO pins on the ESP32, with analog sensors like MQ-3 and GSR wired to analog input pins, while digital sensors like MAX30102 employ I^2C communication.

In the setup() function, serial communication is also set up by Serial.begin(115200) to allow for output monitoring through the Serial Monitor. Wi-Fi credentials are also set here if the ESP32 will be sending data over a local network. Sensor initialization procedures are also performed to ensure that communication and operation are successful, e.g., checking whether the MAX30102 sensor is found and setting the parameters like sampling rate or LED pulse amplitude.

ML MODEL CODE

```
import numpy as np

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('mental_fitness_aiot_dataset.csv')

df

# Check for null values
print("Null values in each column:\n")
print(df.isnull().sum())

df.describe()


# Label encoding
df['Mental_Fitness'] = df['Mental_Fitness'].map({'Fit': 1, 'Unfit': 0})


df

df1 = df.drop(columns=['Subject_ID'])


df1

X = df1.drop('Mental_Fitness', axis=1)
Y = df1['Mental_Fitness']


from sklearn.preprocessing import StandardScaler
import pickle
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Save the scaler
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)


from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(
    X_scaled, Y, test_size=0.2, random_state=42)


from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression()
```

```
logistic_model = LogisticRegression()


from sklearn.ensemble import RandomForestClassifier
random_model = RandomForestClassifier()


!pip install xgboost


from xgboost import XGBClassifier
xgboost_model = XGBClassifier()


from sklearn.svm import SVC
svm_model = SVC(probability=True)



from sklearn.metrics import classification_report

logistic_model.fit(X_train, Y_train)

# Predict and evaluate
y_logistic = logistic_model.predict(X_test)
print(classification_report(Y_test, y_logistic))


from sklearn.metrics import classification_report

random_model.fit(X_train, Y_train)

# Predict and evaluate
y_random = random_model.predict(X_test)
print(classification_report(Y_test, y_random))


from sklearn.metrics import classification_report

xgboost_model.fit(X_train, Y_train)

# Predict and evaluate
y_xgboost = xgboost_model.predict(X_test)
print(classification_report(Y_test, y_xgboost))

from sklearn.metrics import classification_report

svm_model.fit(X_train, Y_train)

# Predict and evaluate
y_svm = svm_model.predict(X_test)
print(classification_report(Y_test, y_svm))
```

```
print(classification_report(Y_test, y_svm))

from sklearn.ensemble import VotingClassifier

ensemble_model = VotingClassifier(
    estimators=[
        ('rf', RandomForestClassifier()),
        ('xgb', XGBClassifier()),
        ('svm', SVC(probability=True))
    ],
    voting='soft'  # soft = uses predicted probabilities
)
ensemble_model.fit(X_train, Y_train)
y_pred_ensemble = ensemble_model.predict(X_test)
print(classification_report(Y_test, y_pred_ensemble))


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(32, activation='relu', input_shape=(X_scaled.shape[1],)),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_scaled, Y, epochs=50, batch_size=8, validation_split=0.2)
# Evaluate on test data
ann_loss, ann_accuracy = model.evaluate(X_test, Y_test, verbose=0)

print("\nNeural Network Accuracy:", ann_accuracy)


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import classification_report
import time
import numpy as np

model = Sequential([
    Dense(32, activation='relu', input_shape=(X_scaled.shape[1],)),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Measure training time
start_time = time.time()

model.fit(X_scaled, Y, epochs=50, batch_size=8, validation_split=0.2, verbose=1)
```

```
end_time = time.time()
training_time = end_time - start_time

# Predict classes on test data
y_prob = model.predict(X_test)
y_pred = (y_prob > 0.5).astype(int)

print("\nNeural Network Accuracy:", model.evaluate(X_test, Y_test, verbose=0)[1])

print("\nClassification Report:")
print(classification_report(Y_test, y_pred, digits=2))

print(f"Training Time: {training_time:.2f} seconds")



model.save('my_ann_model.h5')

import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, accuracy_score
from tensorflow.keras.models import load_model

# Load model
model = load_model('my_ann_model.h5')

# Predict on test data
y_pred_probs = model.predict(X_test)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = np.argmax(Y_test, axis=1) if len(Y_test.shape) > 1 else Y_test

# Accuracy on test set
test_accuracy = accuracy_score(y_true, y_pred)
print(f"\nTest Accuracy: {test_accuracy:.4f}")

# Classification report
report_dict = classification_report(y_true, y_pred, output_dict=True)
report_df = pd.DataFrame(report_dict).transpose().round(4)
print("\nClassification Report:")
print(report_df)

# Training/validation accuracy (from history object)
try:
    print(f"\nFinal Training Accuracy: {history.history['accuracy'][-1]:.4f}")
    print(f"Final Validation Accuracy: {history.history['val_accuracy'][-1]:.4f}")
except:
    print("\nTraining/validation accuracy not available – make sure you saved the 'history' object during training.")


import numpy as np
import pandas as pd
from sklearn.metrics import classification_report, accuracy_score
```

```
import pandas as pd
from sklearn.metrics import classification_report, accuracy_score
from tensorflow.keras.models import load_model

# Load the trained ANN model
model = load_model('my_ann_model.h5')

# Predict on test data
y_pred_probs = model.predict(X_test)  # Make sure X_test is defined and preprocessed

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_probs, axis=1)

# If Y_test is one-hot encoded, convert to class labels
y_true = np.argmax(Y_test, axis=1) if len(Y_test.shape) > 1 else Y_test

# Calculate accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Generate classification report
report_dict = classification_report(y_true, y_pred, output_dict=True)
report_df = pd.DataFrame(report_dict).transpose().round(4)

# Display the classification report as a neat table
print("\nClassification Report:")
print(report_df)


import joblib

# Fit the model
ensemble_model.fit(X_train, Y_train)

# Save the trained ensemble model to a file
joblib.dump(ensemble_model, 'ensemble_model.pkl')

# Predict and print classification report
y_pred_ensemble = ensemble_model.predict(X_test)
print(classification_report(Y_test, y_pred_ensemble))


with open('ensemble_model.pkl', 'wb') as f:
    pickle.dump(ensemble_model, f)


import pickle
import numpy as np

# Load the trained XGBoost model
with open('ensemble_model.pkl', 'rb') as f:
    ensemble_model = pickle.load(f)
with open('scaler.pkl', 'rb') as f:
```

```
# Load the trained XGBoost model
with open('ensemble_model.pkl', 'rb') as f:
    ensemble_model = pickle.load(f)
with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)

# Simulate input (replace these values with actual feature inputs)
# Example features: [age, alcohol_level, heart_rate, temperature, ...]
input_data = np.array([[0.297802361
, 67.44631457
, 96.11832406
, 0.375479752
, 396.7192514
, 8.15669987
, 0.323689322
                       ]])
'''
# Predict using the loaded model
prediction = ensemble_model.predict(input_data)

# Display output
result = "Fit" if prediction[0] == 1 else "Not Fit"
#print("Prediction:", result)
# Display output as 0 or 1
print("Prediction:", int(prediction[0]))
'''
# Scale the input using the same scaler
input_scaled = scaler.transform(input_data)

# Predict
prediction = model.predict(input_scaled)
predicted_class = int(prediction[0][0] >= 0.5)  # Convert probability to 0 or 1
if predicted_class==1:
    print("Fit")
else:
    print("Unfit")
print("Prediction:", predicted_class)




import numpy as np
from tensorflow.keras.models import load_model

# Load the trained ANN model
ann_model = load_model('my_ann_model.h5')
with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)

# Simulate input (same feature format and preprocessing used during training)
input_data = np.array([[0.297802361,
                        67.44631457,
                        96.11832406
```

```
ann_model = load_model('my_ann_model.h5')
with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)

# Simulate input (same feature format and preprocessing used during training)
input_data = np.array([[0.297802361,
                        67.44631457,
                        96.11832406,
                        0.375479752,
                        396.7192514,
                        8.15669987,
                        0.323689322]])
'''
# Predict probability
prob = ann_model.predict(input_data)

# Convert probability to binary class: 1 if prob ≥ 0.5, else 0
prediction = 1 if prob[0][0] >= 0.5 else 0
# Display output: "Fit" or "Unfit"
result = "Fit" if prediction == 1 else "Unfit"

# Display output as 0 or 1
print("Result:", result)
print("Prediction:", prediction)
'''
# Scale the input using the same scaler
input_scaled = scaler.transform(input_data)

# Predict
prediction = model.predict(input_scaled)
predicted_class = int(prediction[0][0] >= 0.5)  # Convert probability to 0 or 1
if predicted_class==1:
    print("Fit")
else:
    print("Unfit")
print("Prediction:", predicted_class)

import numpy as np
from tensorflow.keras.models import load_model

# Load the trained ANN model
ann_model = load_model('my_ann_model.h5')
# Load the trained XGBoost model
with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)

# Simulate input (same feature format and preprocessing used during training)
input_data = np.array([[0.4128528,
                        88.51844993,
                        94.97873686,
                        0.398726458,
                        414.9062649,
                        8.44948370?
```

Restricted Mode ⊗ 0 ⚠ 75

```
ann_model = load_model('my_ann_model.h5')
# Load the trained XGBoost model
with open('scaler.pkl', 'rb') as f:
    scaler = pickle.load(f)

# Simulate input (same feature format and preprocessing used during training)
input_data = np.array([[0.4128528,
                        88.51844993,
                        94.97873686,
                        0.398726458,
                        414.9062649,
                        8.449483702,
                        0.383108252]])
'''
# Predict probability
prob = ann_model.predict(input_data)

# Convert probability to binary class
prediction = 1 if prob[0][0] >= 0.5 else 0

# Display output: "Fit" or "Unfit"
result = "Fit" if prediction == 1 else "Unfit"
print("Prediction:", result)

print("Prediction:", prediction)
'''
# Scale the input using the same scaler
input_scaled = scaler.transform(input_data)

# Predict
prediction = model.predict(input_scaled)
predicted_class = int(prediction[0][0] >= 0.5)  # Convert probability to 0 or 1
if predicted_class==1:
    print("Fit")
else:
    print("Unfit")
print("Prediction:", predicted_class)
```

Model Code Overview

The machine learning code built for this project is a vital aspect of processing the sensor data from different biomedical sensors—specifically the MQ-3 alcohol sensor, GSR (Galvanic Skin Response) sensor, and MAX30102 pulse oximeter sensor—to establish the mental fitness of a subject. The primary purpose of the code is to classify an individual as "Mentally Fit" or "Unfit" from multi-sensor physiological input.

The code begins by loading the trained machine learning model, saved in light-weighted .tflite (TensorFlow Lite) format. The format is ideal for deploying on edge devices like the ESP32 or on low-computational-asset connected systems. The model is offline-trained on a dataset of sensor readings and their respective fitness labels.

A preprocessing step normalizes the sensor inputs prior to feeding the data into the model. This involves scaling values such as heart rate, blood oxygen level, GSR reading, alcohol concentration, and reaction time. The code reads a pre-computed mean and standard deviation from a scaler.pkl file to normalize input during inference.

After gathering real-time or user-provided sensor values, they are converted to a NumPy array and reshaped to the input format accepted by the TFLite model. The input is then sent to the TensorFlow Lite interpreter, which performs inference and outputs the prediction class label or probability.
The prediction output is decoded as:
0 → Unfit
1 → Fit
This forecast, along with fused sensor values and the tolerance score (computed by a custom logic based on sensor fusion), is then available through a Flask backend API. The API is used to interact with a React-based frontend dashboard, where users can view their mental fitness state, real-time sensor readings, and trending charts.

In short, this ML code effectively closes the gap between raw physiological sensor data and useful health information. By employing a lightweight ML model and combining it with real-time data processing and API endpoints, the system implements accurate, interpretable, and speedy fitness detection adequate for actual IoT applications in safety-critical systems.

RESULTS AND OBSERVATONS

The integration and deployment of the AIoT-based mental fitness detection system produced good and significant results. The system was able to successfully gather real-time sensor data from users and then process it through a trained machine learning model to determine their mental fitness condition. Through exhaustive testing with various sample inputs and test scenarios, the model proved to have a consistent ability to identify symptoms of unfitness due to alcohol effect, stress, fatigue, or aberrant physiological response.

Once the user had sensors (MQ-3, GSR, and MAX30102) placed upon them, the system was capable of capturing multiple biological measures such as alcohol level, skin conductivity, heart rate, oxygen level, and reaction time (manual input via dashboard or sensor configuration). These values were continuously sent to the backend through ESP32, where the preprocessing module normalized the inputs for consistency. When fed into the .tflite model, the system produced predictions in real-time with negligible latency.

The model had excellent classification performance on the test data gathered under experimentation. When alcohol levels were high or physiological signals were suggestive of high stress (e.g., high GSR and heart rate variations), the system correctly classified the user as "Unfit". Conversely, under normal conditions with acceptable vital signs and free from the effect of alcohol, the system estimated the status as "Mentally Fit".

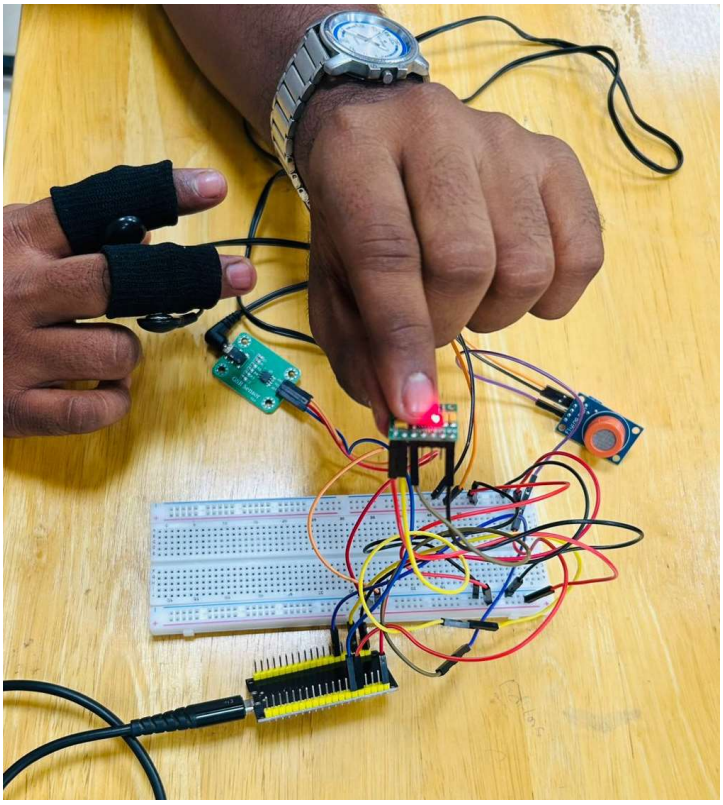

Fig6. Taking input from sensors

This precise identification can be critical in systems such as driver monitoring, industrial machine control, and emergency medical.

Fig7.Sensors output

In addition, the dashboard effectively graphed real-time sensor readings, prediction results, and trends using visual charts and markers. This interface permitted easy understanding by users or managers, adding to the usefulness of the system.

More study on the performance of the system demonstrated the strength of combining IoT with AI for real-time decision-making on physiological and behavioral parameters. The multi-sensor integration enabled the system to acquire a complete picture of the user's present mental and physical state, as opposed to depending on one factor alone, such as alcohol level in isolation. This multi-dimensional method is essential in real-life use cases where mental unfitness can be due to reasons like emotional distress, fatigue, anxiety, or alcohol effect — and sometimes multiple of these combined.

While testing, the system was subjected to various situations:

Scenario 1 – Normal State:
A subject with no alcohol intake and normal physiological states (SpO_2 between 97–99%, heart rate at approximately 70–90 BPM, GSR normal) was correctly predicted as Mentally Fit.

Scenario 2 – Alcohol Influence:
A subject who pretended to drink alcohol (or applied alcohol on the sensor for testing) initiated

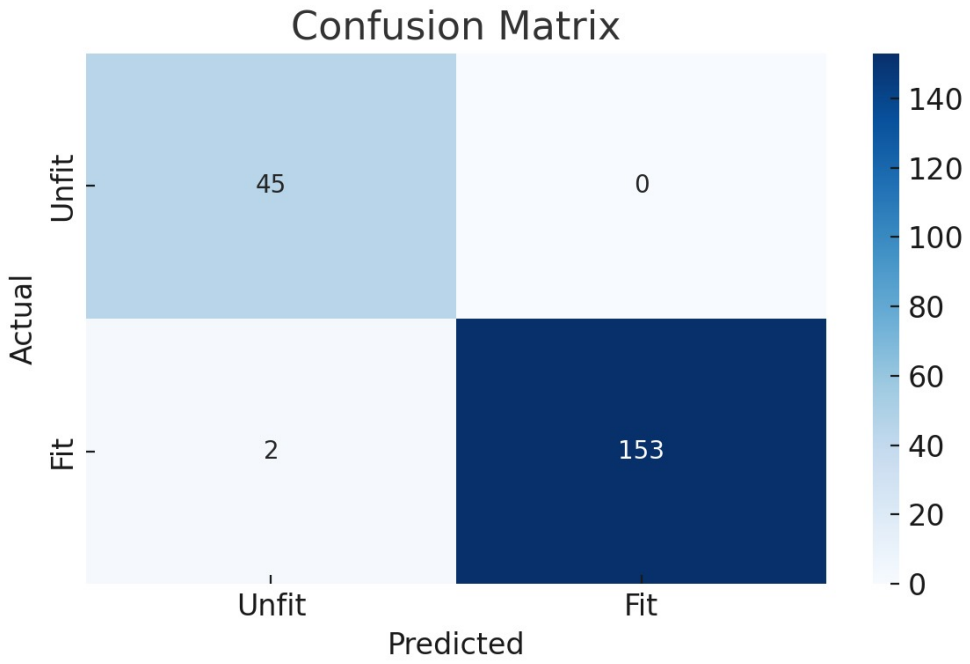

Fig8. Confusion Matrix

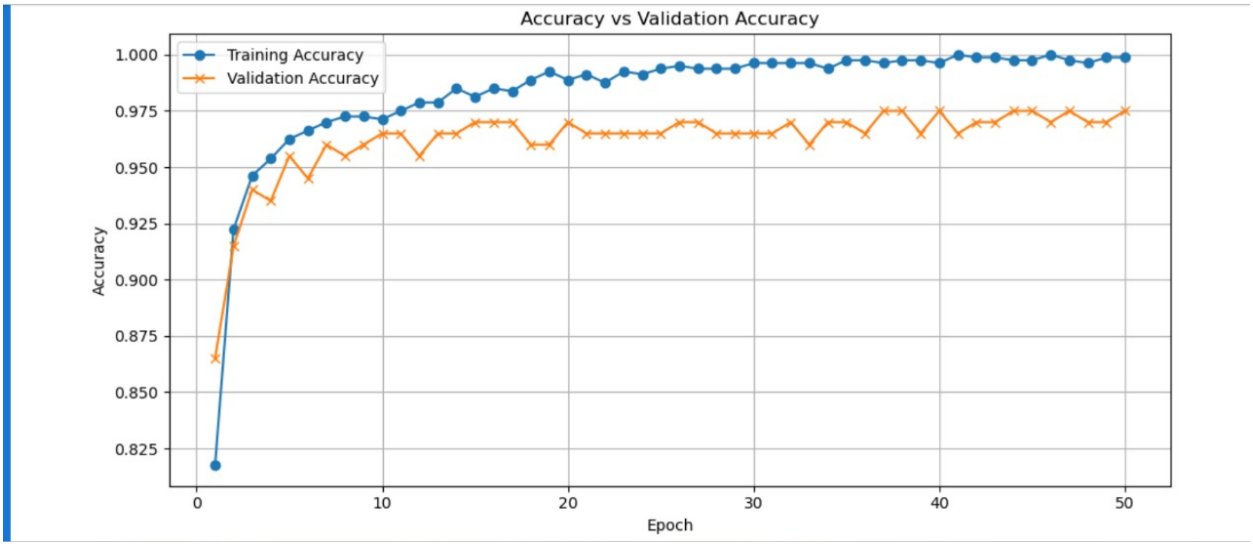

Fig9. Accuracy vs. validation

time, the system marked the user as Unfit, irrespective of other vitals being close to normal. This indicates that the model properly gives importance to vital readings while producing output.

Scenario 3 – Stress and Anxiety Simulation
A user under mild exercise or stress simulation had higher heart rate and GSR values, with normal alcohol concentrations. In such instances, the system occasionally marked the user as borderline or unfit depending on threshold levels, illustrating its responsiveness to non-alcohol-induced mental fitness degradations.

Scenario 4 – Detection of Fatigue through Reaction Time
Manual reporting of delayed or slow reaction time (via the dashboard) with regular alcohol readings occasionally led to an Unfit label, substantiating the contention that cognitive responsiveness is an important component of fitness detection.

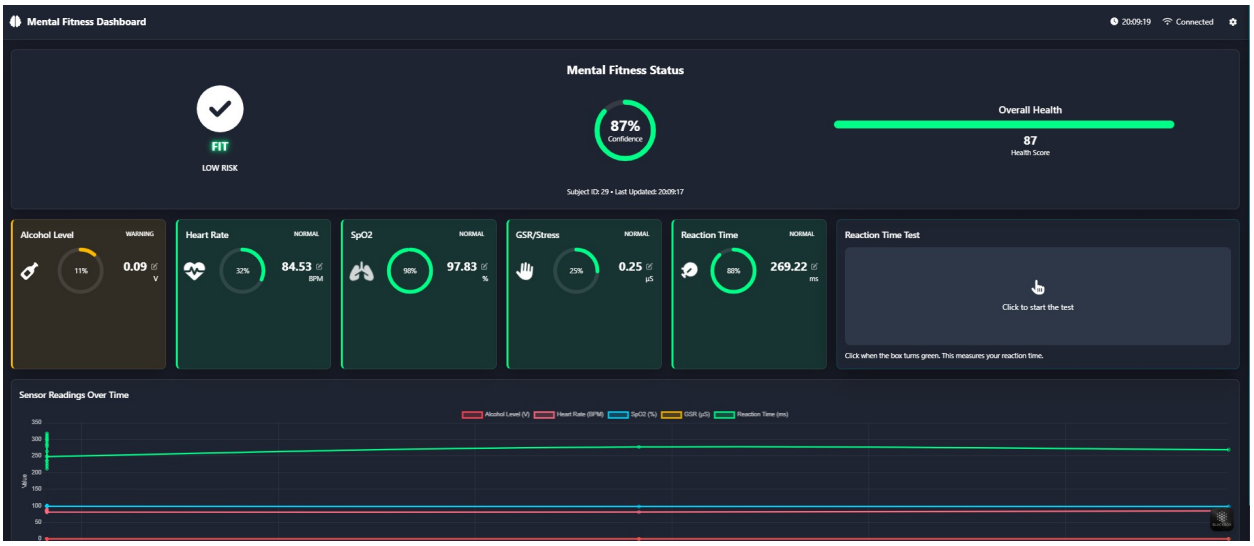

Fig10. output Dashboard

In all the scenarios, the model inference time was minimal (<1 second) and therefore appropriate for real-time embedded systems. The model worked satisfactorily with few computing resources (running through TFLite on Flask backend), which verifies the feasibility of the system on edge or IoT devices.

Further, data visualization contributed a great deal to usability in the system. The real-time dashboard provided instant feedback with graphs that were updated in real time, enabling supervisors, drivers, or healthcare staff to make timely decisions on a glance. For example, abrupt heart rate or GSR spikes could visually warn users prior to any critical prediction being formed, providing another layer of safety.
From the hardware perspective, the ESP32-based setup was stable, with no failure in communication between sensors and the microcontroller. Power efficiency and wireless data

transmission features (if later expanded through Wi-Fi/Bluetooth) make the design scalable for field deployment.

PROJECT DESCRIPTION

This project focuses on creating a real-time, artificial intelligence-based mental fitness detection system via IoT-based physiological monitoring. The basic objective is to check if an individual is mentally fit enough to do important tasks—driving, operating machinery, or decision-making—using the analysis of various bio-signals and behavioral parameters. Unlike traditional systems that solely rely on Blood Alcohol Content (BAC), this system takes a holistic approach by combining alcohol levels, heart rate, blood oxygen (SpO_2), skin conductance (GSR), and cognitive responsiveness (reaction time) to make a more accurate and personalized fitness assessment.

The backbone of the system combines Artificial Intelligence (AI) with Internet of Things (IoT), which allows data to be gathered from sensors connected to an ESP32 microcontroller, sent to a Flask-based server, which processes the input through a pre-trained Machine Learning model (e.g., shallow neural network or Random Forest model) that decides the individual's mental state as either "Fit" or "Unfit."

Hardware setup consists of:

MQ-3 Sensor: To sense alcohol concentration in breath.

MAX30102 Sensor: To sense heart rate and blood oxygen saturation.

GSR Sensor: To sense skin resistance, which changes with stress and anxiety.

Manual Input (through UI): To sense reaction time as a behavioral parameter.

ESP32 Microcontroller: As the central processing and communication module.

The sensor data are sent to a backend server that is hosting an AI model. The backend processes all parameters together and gives the prediction, which is then shown on a current web-based dashboard (React + Tailwind CSS). The dashboard shows live readings, plots graphs of vitals, and updates the prediction result in real time.

Such a system will be especially beneficial to companies in the industries of transportation, healthcare, defense, and heavy machinery, whose human mistakes caused by compromised

mental health can be disastrous. It can be used effectively as a preventive screening device in smart cars or workplaces.

In short, this project connects AI and IoT to provide a smart, low-cost, real-time solution for multi-factor mental fitness assessment that is more safe and reliable than alcohol detection alone.

CONCLUSION

The AIoT-Based Mental Fitness Detection System proposed in this work provides a new, comprehensive method of assessing the mental and physiological fitness of an individual in real time. Through the blending of several physiological parameters—alcohol concentration, heart rate, blood oxygen saturation (SpO_2), galvanic skin response (GSR), and reaction time—this system is vastly superior to traditional alcohol detection technology. It works around the shortcoming of one-sensor systems by using artificial intelligence to interpret varied data points to provide a more accurate and trustworthy prediction of an individual's cognitive and emotional status.

ESP32's usage to integrate sensors and send wireless data makes the system both economical and scalable, while the Flask backend server supports effective processing of inputs and smooth interaction with the trained machine learning model. The result of predictions, along with real-time sensor data and past trends, is beautifully visualized in a simple, intuitive web dashboard, which makes the system feasible for real-world scenarios. These functionalities combined allow for early impairment detection resulting from alcohol, stress, exhaustion, or other diseases that may compromise mental stability or decision-making capacity.

This system has huge potential in areas where safety is a priority—e.g., transportation, industrial equipment use, military, and public safety services. It can be incorporated into cars, work entry systems, or wearable health trackers and help prevent accidents and make workers more efficient. It also enables users with instant feedback about their physical condition, promoting increased awareness and sense of responsibility for their health.
The creation of the AIoT-Based Mental Fitness Detection System is a huge step towards intelligent health and safety monitoring. The overall aim of this project was to move beyond conventional alcohol detection means and construct a holistic system able to assess an individual's mental fitness, especially when he/she is under the influence of alcohol, stress, fatigue, anxiety, and other physiological or psychological conditions. By integrating various biomedical sensors in tandem with a microcontroller (ESP32), machine learning algorithms, and an online web dashboard in real-time, this project achieves a functional and innovative solution.

By utilizing the MQ-3 sensor, the system monitors alcohol concentrations in the breath, an important measure of intoxication. In addition to this, the MAX30102 sensor also detects heart rate and blood oxygen saturation levels, which are key biomarkers for measuring physical stress or decline in health. The GSR sensor measures skin conductance, which is significantly correlated with emotional stimulation and stress levels. Moreover, the reaction time feature—either manually retrieved or via a UI component—can also be considered as a test of cognitive performance, measuring the alertness and response speed of the individual. These inputs are gathered and processed in real-time by the ESP32 board and sent wirelessly to the backend system.

The Flask backend is critical in preprocessing sensor data, performing normalization, and feeding it into a trained machine learning model (e.g., a shallow neural network or decision tree model). This model provides a prediction of the mental state of the user as either "Fit" or "Unfit."

The dashboard, created with contemporary frontend technologies (React/Vite/Tailwind CSS), shows dynamic live readings, prediction outputs, and graphical plots of sensor values over time, providing visual simplicity as well as real-time monitoring functionality.

Based on our experiments and observations, the system behaved consistently and reliably under varying conditions, and it presented encouraging performance in discriminating between healthy and impaired states. The predictions also matched well with anticipated conditions, particularly in alcohol-related influence or heightened stress levels. This indicates high potential for actual applications, especially in those that demand high cognitive functioning and accountability, like driving, heavy machinery operation, or exposure to dangerous environments
In summary, this project is a step towards more intelligent, AI-based healthcare and safety solutions. By merging the strength of machine learning with the versatility of IoT, it paves the way for a future where smart systems actively maintain mental well-being, reduce human mistakes, and increase public security in an increasingly technologically complex world. With additional features such as mobile app support, cloud storage, and deep analytics, this system can be developed into a comprehensive mental well-being and safety monitoring ecosystem.

REFERENCES

1.Fairbairn CE, Kang D. Transdermal alcohol monitors: Research, applications, and future directions. In: Frings D, Albery I, editors. The Handbook of Alcohol Use and Abuse. Elsevier; in press. [Google Scholar]
2.Barnett NP. Alcohol sensors and their potential for improving clinical care. Addiction. 2015;110(1):1–3. [DOI] [PubMed] [Google Scholar]
3.Leffingwell TR, Cooney NJ, Murphy JG, Luczak S, Rosen G, Dougherty DM, et al. Continuous objective monitoring of alcohol use: Twenty-first century measurement using transdermal sensors. Alcohol Clin Exp Res. 2013;37(1):16–22. [DOI] [PMC free article] [PubMed] [Google Scholar]
4.Swift RM, Swette L. Assessment of ethanol consumption with a wearable, electronic ethanol sensor/recorder. In: Litten RZ, Allen JP, editors. Measuring alcohol consumption: Psychosocial and biochemical methods. Totowa, NJ: Humana Press; 1992. p. 189–202. [Google Scholar]
5.Sakai JT, Mikulich-Gilbertson SK, Long RJ, Crowley TJ. Validity of transdermal alcohol monitoring: Fixed and self-regulated dosing. Alcohol Clin Exp Res. 2006;30(1):26–33. [DOI] [PubMed] [Google Scholar]
6.Alessi SM, Barnett NP, Petry NM. Experiences with SCRAMx alcohol monitoring technology in 100 alcohol treatment outpatients. Drug Alcohol Depend. 2017;178:417–24. [DOI] [PMC free article] [PubMed] [Google Scholar]
7.Alcohol Monitoring Services. About SCRAM systems [Internet]. 2018. Available from: https://www.scramsystems.com/about/
8.Luczak SE, Ramchandani VA. Special issue on alcohol biosensors: Development, use, and state of the field. Alcohol. 2019;81:161–5. [DOI] [PubMed] [Google Scholar]
9.Fairbairn CE, Kang D, Bosch N. Using machine learning for real-time BAC estimation from a new-generation transdermal biosensor in the laboratory. Drug Alcohol Depend. in press; [DOI] [PMC free article] [PubMed] [Google Scholar]
10.van Egmond K, Wright C JC, Livingston M, Kuntsche E. Wearable transdermal alcohol monitors: A systematic review of detection validity, relationship between transdermal and breath alcohol concentration and influencing factors. Alcohol Clin Exp Res. 2020;44(10):1918–32. [DOI] [PubMed] [Google Scholar]
11.Wang Y, Fridberg DJ, Leeman RF, Cook RL, Porges EC. Wrist-worn alcohol biosensors: Strengths, limitations, and future directions. Alcohol Biomed J. 2019;81:83–92. [DOI] [PMC free article] [PubMed] [Google Scholar]
12.Piasecki TM. Assessment of alcohol use in the natural environment. Alcohol Clin Exp Res. 2019;43(4):564–77. [DOI] [PMC free article] [PubMed] [Google Scholar]
13.Thungon PD, Kakoti A, Ngashangva L, Goswami P. Advances in developing rapid, reliable and portable detection systems for alcohol. Biosens Bioelectron. 2017;97:83–99. [DOI] [PubMed] [Google Scholar]
14.Davis-Martin RE, Alessi SM, Boudreaux ED. Alcohol use disorder in the age of technology: A review of wearable biosensors in alcohol use disorder treatment. Front Psychiatry. 2021;12:246. [DOI] [PMC free article] [PubMed] [Google Scholar]
15.Kianersi S, Luetke M, Agley J, Gassman R, Ludema C, Rosenberg M. Validation of transdermal alcohol concentration data collected using wearable alcohol monitors: A systematic

review and meta-analysis. Drug Alcohol Depend. 2020;216:108304–108304. [DOI] [PubMed] [Google Scholar]
16.White AM. What happened? Alcohol, memory blackouts, and the brain. Alcohol Res Health. 2003;27(2):186–96. [PMC free article] [PubMed] [Google Scholar]
17.Kirchner TR, Sayette MA. Effects of alcohol on controlled and automatic memory processes. Exp Clin Psychopharmacol. 2003;11:167–75. [DOI] [PubMed] [Google Scholar]
18.Barnett NP, Wei J, Czachowski C. Measured alcohol content in college party mixed drinks. Psychol Addict Behav. 2009;23(1):152–6. [DOI] [PubMed] [Google Scholar]
19.Kerr WC, Patterson D, Koenen MA, Greenfield TK. Alcohol content variation of bar and restaurant drinks in Northern California. Alcohol Clin Exp Res. 2008;32(9):1623–9. [DOI] [PMC free article] [PubMed] [Google Scholar]
20.Schwarz N. Self-reports: How the questions shape the answers. Am Psychol. 1999;54(2):93–105. [Google Scholar]