ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

The Smart Fitness Coach

Team #11

YUXUAN LIN
(yuxuan42@illinois.edu)
XINGYU LI
(xingyul6@illinois.edu)
LISHAN SHI
(lishans3@illinois.edu)
TIANHENG WU
(tw44@illinois.edu)

TA: Xiaoai Wang

May 18, 2025

Abstract

The Smart Fitness Coach is an AIoT-based real-time exercise tracking system designed to provide accurate, low-cost, and accessible fitness guidance without the need for wearable devices or human supervision. Leveraging a compact PCB camera module and a server-side pose estimation pipeline powered by MediaPipe, the system analyzes user posture and delivers dynamic feedback via a cross-platform mobile frontend developed with UniApp.

The project addresses key shortcomings of existing solutions—namely, the discomfort of wearable sensors and the passivity of pre-recorded videos—by enabling skeleton-based tracking of body movements such as squats and push-ups. Core components include a Wi-Fi-enabled image acquisition module, a Flask-based backend with REST and WebSocket interfaces, and a SQLite-integrated session tracker. The entire pipeline operates with latency under 200ms in most conditions.

Extensive validation was conducted across various lighting, distance, and motion scenarios to ensure robustness. Results show consistent recognition accuracy and real-time responsiveness. The final implementation meets design goals in portability, affordability, and performance. Broader impacts include potential integration with fitness equipment in gym environments and the promotion of safe, interactive home exercise practices.

Contents

1 Introduction

Smart Fitness Coach is an AIoT platform that offers instant camera-based exercise tracking and feedback to home fitness workout users. The concept of this project is founded on growing demands for accessible and engaging fitness gadgets that can support users comfortably without the need for wearables or trainers. While existing products are either founded upon intrusive attachment sensors or passive video instructions, the suggested design addresses the gap in that it offers a camera-based, light feedback mechanism with the potential to detect and evaluate large movements such as squats, push-ups, and lunges.

As illustrated in Figure 1, the system captures photos via a Wi-Fi-enabled PCB camera module and then transfers them to a Flask-built cloud-based backend fueled by MediaPose for skeleton pose estimation. Posture evaluation is performed server-side, and personalized feedback is returned to the user through a UniApp-built[1] responsive mobile frontend. The backend also makes use of a SQLite database for session logging and tracking user progress over time. This hybrid architecture offers high accuracy and adaptability with low cost and portability. Compared to wearable systems, our approach does away with issues of discomfort and incomplete anatomical coverage. Unlike static tutorial videos, Smart Fitness Coach offers real-time and dynamic feedback enabling users to adjust their form and reduce the risk of injury. With RESTful APIs and WebSocket channels, real-time data exchange between front-end interface and back-end inference engine offers sub-200ms latency in most situations.

Systematic testing under conditions of lighting, motion, and camera location demonstrates the robustness and reliability of the system. The final implementation yields consistent performance and achieves the original design requirements of affordability, simplified deployment, and user-centric interaction. This document records the motivation, engineering effort, and verification plan that brought Smart Fitness Coach to completion.

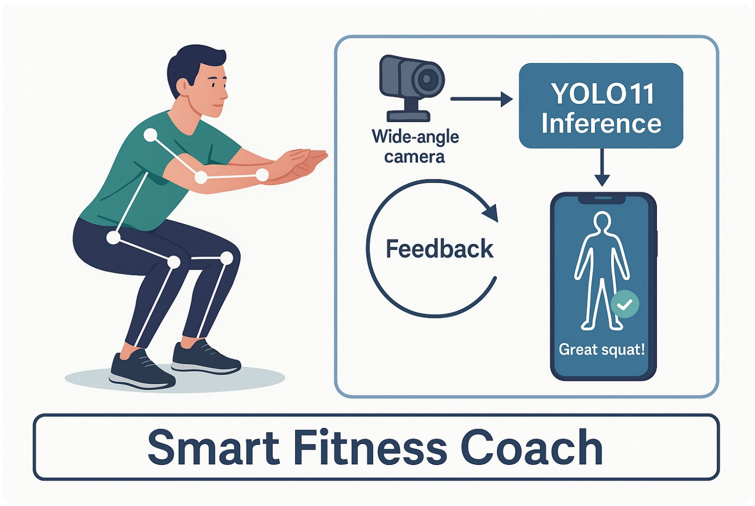


Figure 1: The Smart Fitness Coach

2 Deisgn

2.1 Design procedure

2.1.1 Hardware

The system architecture integrates five key subsystems—power supply, control unit, camera, PCB, and angle regulator—through a structured design methodology balancing performance, cost, and reliability. The power supply employs a USB-Mini input with a low-dropout linear regulator (LDO) to eliminate switching noise, prioritizing stable 5V output over the higher efficiency of buck converters, which was deemed unnecessary for low-current IoT applications. The control unit centers on the AI-M61-32S microcontroller, selected for its integrated Wi-Fi and camera interface, avoiding the complexity of other module designs. The angle regulator utilizes two SG90 servo motors (0–180° range) controlled via PWM pulses generated by the MCU, ensuring precise angular positioning without the complexity of stepper motor drivers.

2.1.2 Software

The software development process followed a progressive shift from embedded local inference to a cloud-assisted, modular architecture. Initially, we aimed for on-device inference using YOLO-based[2] models, but later moved to a hybrid deployment model, balancing low-power front-end data acquisition and robust server-side computation.

At the early development stage, we verified the feasibility of YOLO-based posture recognition using PC simulations. Although the model produced accurate results, we observed significant latency and FPS degradation when attempting embedded inference. This prompted us to test alternative solutions.

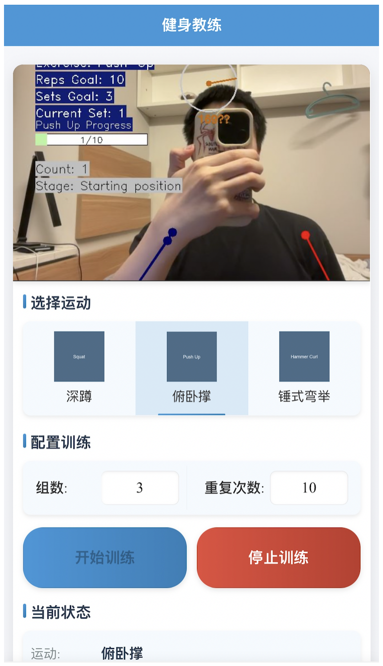
Next, we introduced MediaPipe as a potential inference engine. We tested MediaPipe Pose locally on the server and found its lightweight nature and excellent keypoint resolution (33 landmarks) well-suited for our fitness action recognition tasks (see Table 1). MediaPipe's Python API made it easier to perform keypoint extraction and extend functionality with custom angle-based rule logic.

Table 1: Comparison of YOLOv7, YOLO11, and MediaPipe

Attribute	**YOLOv7**	**YOLO11**	**MediaPipe**
Release Year	2022	2025	2020
Keypoints Supported	17	17	33
Model Size	70 MB	45 MB	7.5 MB
Primary Use-case	Object detection	Pose estimation	Pose tracking
Platform Support	GPU, edge	NPU, GPU, edge	CPU, Web, Native

Once the backend model was stabilized, the front-end acquisition module was switched to our PCB camera module. This low-cost, Wi-Fi-enabled microcontroller was programmed to periodically capture images and transmit them via HTTP POST to the backend server.

Server development then focused on building a RESTful API using Flask, with endpoints to accept uploaded frames, return keypoints and action status, and support future logging extensions. The `POST /upload` endpoint became the primary communication channel, handling image decoding, preprocessing, and inference.



Reps Goal: 10
Sets Goal: 3
Current Set: 1
Push Up Progress
1/10
Count: 1
Stage: Starting position

组数: 3
重复次数: 10

运动: 俯卧撑

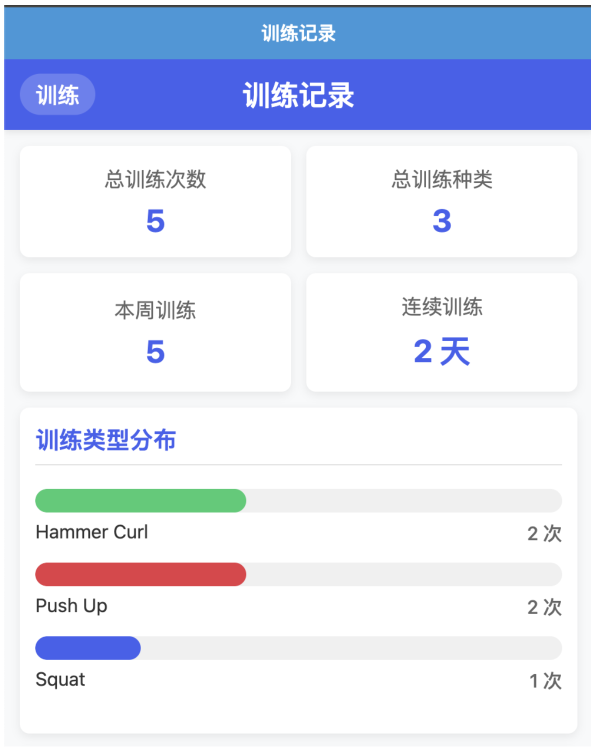
Figure 2: App Index

Finally, we implemented the front-end application using UniApp, a Vue3-based cross-platform framework. This decision allowed us to target Android, iOS, and H5 with a single codebase. The frontend displayed feedback such as posture correctness, repetition count (see Figure 2), and historical records (see Figure 3 and Figure 4) by polling the backend or processing real-time responses.

This procedure was iterative and collaborative, involving multiple cycles of testing, integration, and debugging across all software modules.

2.1.3 Physical Design

Initially, we designed with the goal of multi-functional integration, hoping to create the best product. The initial design adopts a composite structure of waterproof ABS plastic and anti-slip silicone coating. The surface of the shell is covered with an IP52 protection level, making it suitable for high-humidity environments such as gyms. The back is designed with a foldable stand that supports angle adjustment from 0° to 45°. It integrates


训练记录

训练 训练记录

总训练次数
5

总训练种类
3

本周训练
5

连续训练
2 天

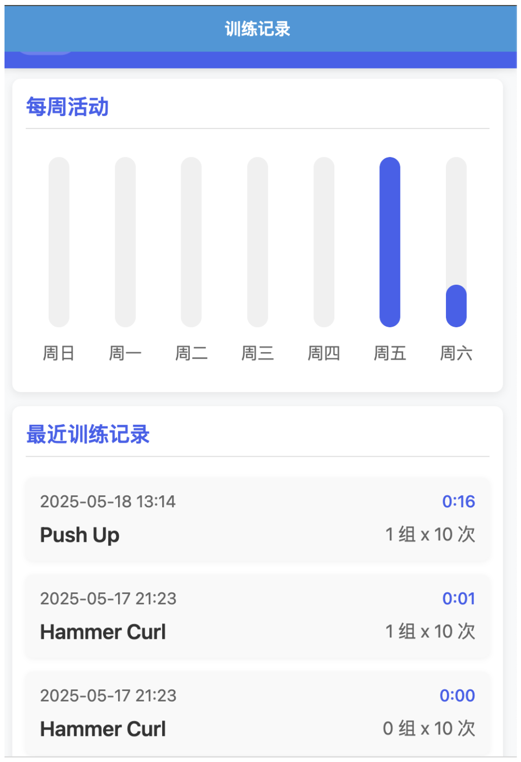
训练类型分布

Hammer Curl 2 次

Push Up 2 次

Squat 1 次

Figure 3: App Dashboard I


训练记录

每周活动

最近训练记录

2025-05-18 13:14 0:16
Push Up 1 组 x 10 次

2025-05-17 21:23 0:01
Hammer Curl 1 组 x 10 次

2025-05-17 21:23 0:00
Hammer Curl 0 组 x 10 次

Figure 4: App Dashboard II

a 5-inch IPS touch screen and physical buttons inside, allowing users to operate the device directly. The camera module adopts a 1080P wide-angle lens and is equipped with a sliding lens hood to reduce the interference of ambient light. However, this complex design exposed the problems of high cost and high manufacturing complexity in actual tests. This version of the design is difficult to implement.

Based on the verification results, we decided to simplify the design. The waterproof design was removed, and the shell material was ordinary ABS plastic. The foldable bracket is changed to a fixed mortise and tenon structure, and the stable connection of the pan-tilt base is achieved through the geometric self-locking feature, avoiding the risk of loosening during movement. Touch screens and physical buttons have been eliminated, and user operations completely depend on mobile phone apps. This move not only reduces hardware costs but also decreases circuit complexity and failure points. The camera module is simplified to a basic configuration, with the lens hood design removed. Through simplification, we successfully achieved the basic functions required by the project while saving a significant amount of costs.

2.2 Design details

2.2.1 Subsystem of Hardware Setup

Below are the Hardwares required for our project. Figure 5 shows the block diagram for the project.

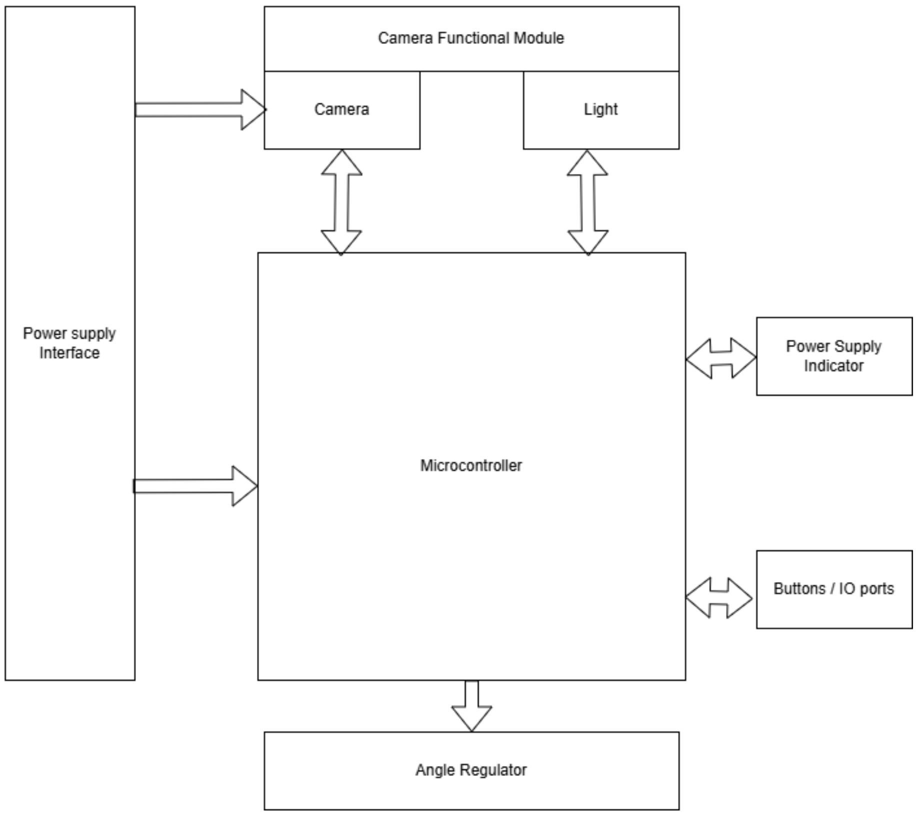


Figure 5: System block diagram of PCB

- Power Supply
  This power supply module is designed to accept power from a USB-Mini connector and deliver a stable 5V DC output.

- Control Unit
  The control unit is responsible for coordinating the operations of all components and executing application-level logic. It sends the signal to control the angle regulator.

- Camera
  A mounted RGB camera captures user movements during workouts. Stable frame acquisition is the key to realize accurate attitude key point recognition.

- PCB
  A PCB is designed to integrate all modules combining AI-M61-32S which integrates Wi-Fi radio transceiver with a camera and can be programmed through Arduino IDE. // The camera interface connects to OV2640 camera to captures the postures of users. Figure 6 shows the schematic of the camera in PCB.

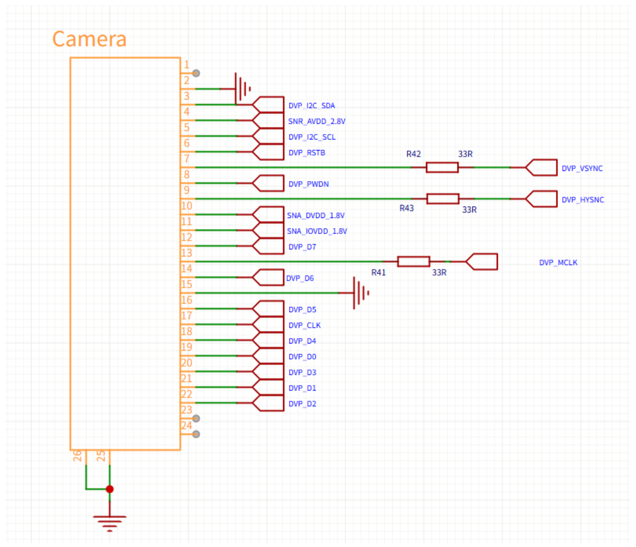


Figure 6: Camera

The Processor serves as the central control logic processor for peripheral communication. Besides, it accepts the video data captured by send the video stream to the website through Wi-Fi. Figure 7 shows the schematic of the camera in PCB.

- Angle Regulator
  The motor unit adjusts the camera angle using two SG-90 servo motors (see Figure 8) to ensure optimal field of view during workouts. The servos are controlled by PWM signals from the microcontroller, enabling precise angular positioning. This subsystem includes the servo motors and a motor control interface to coordinate smooth adjustments.
  We need the servo motors to operate within a 0–180° range.The SG90 servo motor

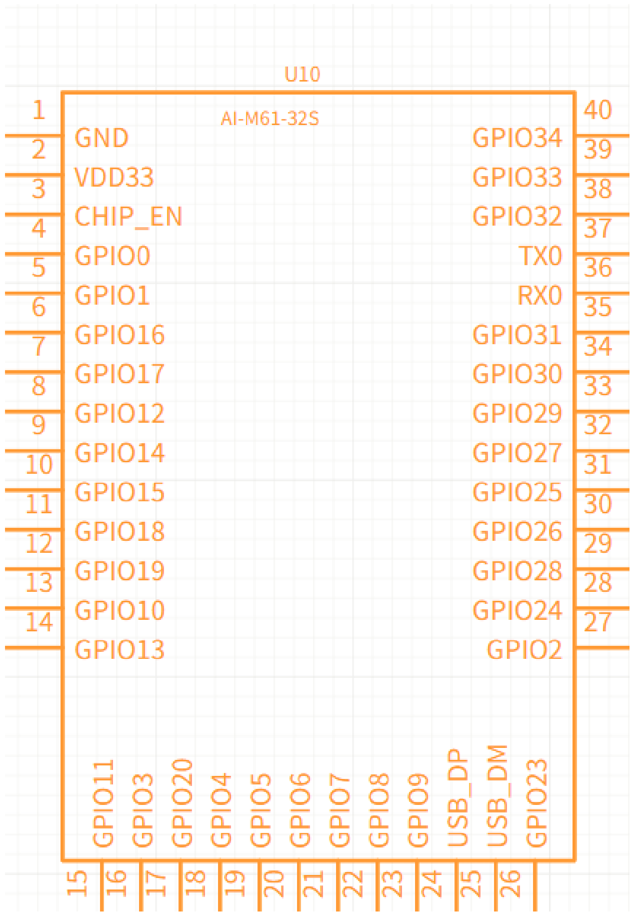


Figure 7: Processor

position is controlled by a PWM signal, where the pulse width directly determines the angular position.
Given the signal frequency is 50HZ. The pulse width (t_{pulse}) is linearly proportional to the target angle (θ)[3]:

$$\begin{aligned} t_{\text{pulse}}(\theta) &= 500\,\mu\text{s} + \left(\frac{\theta}{180^\circ}\right) \times 2000\,\mu\text{s} \\ &= 500 + 11.11\theta \quad (\mu\text{s}) \end{aligned}$$

So, when the pulse width is 0.5ms, the angle is 0° and when the pulse width is 2.5ms, the angle is 180°.

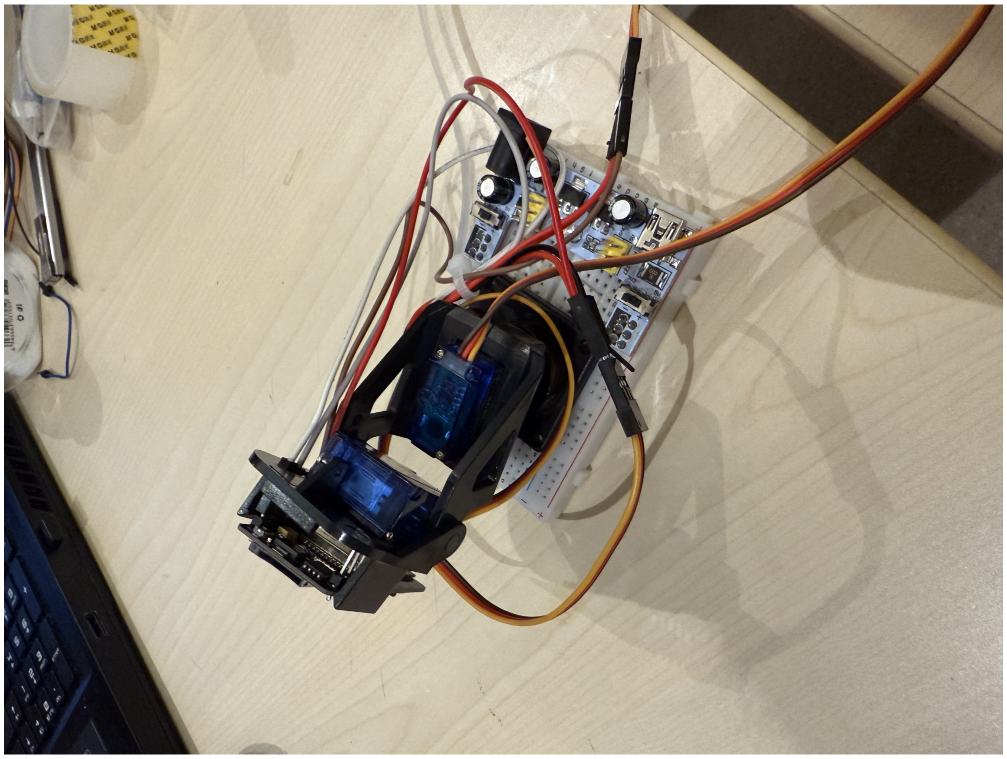


Figure 8: Servo Motors

2.2.2 Software Overview

The Smart Fitness Coach system adopts a decoupled architecture composed of three core components: a Wi-Fi-enabled PCB camera module for image acquisition, a Flask-based backend with MediaPipe for pose estimation and SQLite for session storage, and a UniApp (Vue 3) cross-platform frontend. This design enables real-time posture analysis via REST and WebSocket APIs, ensures responsiveness across Web and mobile platforms, and supports independent development of sensing, inference, and user interface layers.

2.2.3 Software Components

The software system consists of three major components: the stream uploader, the backend inference server, and the UniApp mobile frontend.

PCB Image Upload System The PCB is programmed with Arduino C/C++ to periodically capture images (default resolution 640×480) and transmit them via HTTP POST in JPEG format. It includes features such as retry-on-failure logic, local buffering to prevent frame loss, and dynamic LED control for low-light compensation.

Backend Flask Inference API The backend, built with Flask and Python, hosts a MediaPipe Pose estimator. Its core endpoint `POST /upload` handles incoming images, applies preprocessing (decoding/resizing), performs inference, and responds with 33 keypoints, posture classification (e.g., `squat_good`), and joint angles. Additional endpoints like `GET /status` and `GET /data` are used for system monitoring and result retrieval. CORS is enabled to support cross-origin requests from frontend clients.

UniApp Mobile Frontend The frontend, built with Vue 3 and UniApp, provides a responsive, cross-platform interface. Users interact through screens like: `Index` (navigation hub), `Live Workout` (real-time feedback with skeleton overlay), `Profile`, `History`, and `dashboard`. It uses axios for API communication, localStorage for caching, and includes animations and progress rings for user engagement. A modular design ensures maintainability across platforms.

2.2.4 Implementation Specifics

The implementation of the Smart Fitness Coach software is divided into two primary sections: the Python-based backend server and the Vue 3-based UniApp frontend. Table 2 summarizes the core backend responsibilities and modules, while Table 3 outlines the structure and logic of the major frontend components.

Backend implementation The backend centers around a single entry file, `app.py`, which initializes the Flask application, manages image ingestion, invokes pose inference, and handles communication with clients. As shown in Table 2, image frames are received through the `POST /upload` endpoint in JPEG format. Each frame is validated and stored in an in-memory queue before being processed by a worker thread running the MediaPipe Pose model. Keypoints and angles (e.g., knees, elbows) are extracted and published to a Redis-backed message bus.

For real-time visual feedback, the server exposes a `/video_feed` MJPEG endpoint and a WebSocket interface `exercise_status`, which transmits inference results at approximately 20 frames per second. The system also maintains persistent workout summaries, stored in `data/workouts.json`, and generates dashboard metrics from historical logs. Additional reliability features include CORS handling, structured logging, health checks via `GET /health`, and automatic camera re-initialization on failure.

Table 2: Backend Functional Modules and Responsibilities

Module / Responsibility	Description
app.py	Main entry point; initializes Flask application and registers routes.
Image Ingestion	POST /upload receives JPEG frames, validates content-type, and queues them for inference.
Pose Inference	MediaPipe Pose model runs in a worker thread; calculates joint angles (e.g., knees, elbows) and sends results to a Redis-backed message queue.
Real-Time Streaming	/video_feed serves MJPEG stream; exercise_status WebSocket emits real-time JSON packets at 20 FPS.
Persistence	Inference results are logged in workouts.json; dashboard aggregates weekly stats from logs.
Reliability Features	Supports CORS, exposes GET /health for monitoring, enables structured logging and auto camera re-initialization on failure.

Frontend implementation The client is developed using Vue 3 and UniApp, enabling deployment to multiple platforms including H5, Android, and WeChat Mini Programs. Table 3 presents the main components of the frontend and their associated responsibilities.

The index.vue file serves as the primary live workout interface, where the video stream is displayed, WebSocket messages are parsed, and skeleton overlays along with repetition counters are rendered in real time. The dashboard.vue component retrieves exercise history from /api/dashboard_data and visualizes trends using the uCharts library. The exercise.vue component provides an interface for configuring workout parameters, such as set size, rest interval, and repetition sensitivity.

The frontend further employs axios interceptors to log HTTP traffic and display toast notifications upon network failure using global mixins. Pinia is used for state management, allowing local caching and synchronization of user preferences across sessions.

API Surface & Environment Table 4 summarises the public REST endpoints; WebSocket events mirror the same semantics for live sessions.

As shown in the table 5, the back-end relies on the combination of Flask-SocketIO and MediaPipe to achieve real-time pose reasoning and WebSocket state push. The same table also lists the front-end Vue 3 and UniApp, explaining the role of each package in cross-platform interface rendering.

Table 3: Frontend Pages and UI Logic (UniApp Vue 3)

Component File	Functionality Description
`index.vue`	Main workout interface. Opens MJPEG video stream, subscribes to WebSocket feedback, overlays skeletons, and tracks repetitions in real time.
`dashboard.vue`	Fetches data from `/api/dashboard_data` and visualizes statistics using uCharts, including weekly frequency, exercise distribution, and activity streaks.
`exercise.vue`	Provides session configuration options such as set size, rest intervals, and auto-count sensitivity.
Global logic	Axios interceptors log all HTTP traffic. Global mixins display fallback toasts when disconnected. Pinia is used for local state persistence and synchronization.

Table 4: Essential REST API endpoints

Route	Method	Description
`/api/exercises`	GET	List available exercise types
`/api/start_exercise`	POST	Begin a workout session
`/api/stop_exercise`	POST	Terminate the current session
`/api/status`	GET	Poll current workout state
`/api/dashboard_data`	GET	Aggregated training statistics
`/api/health`	GET	Liveness / readiness probe

Table 5: Core tool-chain with roles

Layer	Package / Version	Primary purpose
Backend	Flask 2.0.1	HTTP routing and REST API framework
	Flask-SocketIO 5.1.1	WebSocket transport for real-time status push
	Werkzeug 2.0.1	WSGI utilities powering Flask under the hood
	flask-cors 3.0.10	Cross-origin resource sharing for mobile clients
	MediaPipe 0.8.9.1	Pose estimation (33 landmarks)
	OpenCV 4.5.3.56	Frame capture / image preprocessing helpers
	NumPy 1.21.2	Numeric backend for angle computation
Frontend	Vue 3.4.21	Reactive UI core for all pages
	UniApp 3.0.0	Cross-compile Vue code to H5 / WeChat Mini-App
	Vite 5.2.8	Fast dev server & production bundler
	uCharts 3.x	Canvas-based charts for dashboard analytics
	socket.io-client 4.8.1	WebSocket client that mirrors Flask-SocketIO events
	vue-i18n 9.1.9	Internationalisation for multi-language UI
	Sass 1.89 (+ loader 16.0.5)	Pre-processing for modular, theme-able styles

Current strengths and future work Planned enhancements include streamlining the **/video_feed** path with HLS, introducing JWT-based authentication, and extending the exercise library to cover lunges and plank postures.

2.2.5 Physical Components

The physical design is shown in Figure 9 and 10. The design has two parts. One part is the top, the other is the bottom. The shell is modeled with SolidWorks[4] and made by 3D printing[5], which is low-cost. There two openings on the top for connecting the gimbal base[6]. The small openings forms a mortise and tenon structure with the base of the pan-tilt for fixation[7]. And the bigger one is used to fix the camera. The camera is fixed on the housing, and the housing is fixed on the base of the pan-tilt. We can use the APP to remotely control the rotation of the pan-tilt, thereby adjusting the angle of the camera. For the bottom, there is a bigger hole to place the pan-tilt.

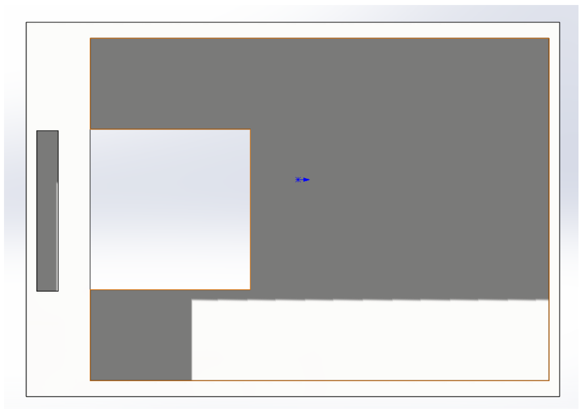


Figure 9: Top Component

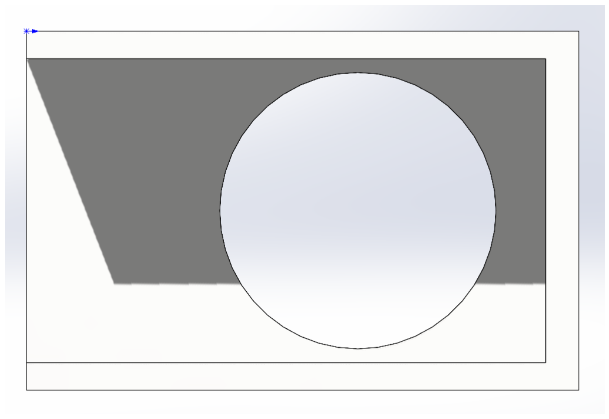


Figure 10: Bottom Component

3 Verification

3.1 Control Unit

The control unit is based on the Ai-M61-32S module and is programmed using the Arduino framework. Verification was done by uploading the compiled sketch via USB and observing the system's runtime behavior. Upon powering the device, the module successfully initialized the onboard DVP camera and the Wi-Fi access point. The detailed verification process is summarized in Table 6.

Table 6: Control Unit Verification Summary

Verification Item	**Method / Observation**
System Boot	Upload sketch via USB and observe serial logs
Camera Initialization	Initialized using `esp_camera` library; successful config confirmed
Wi-Fi AP Creation	Access point `Team11` visible to client devices
HTML Interface Access	Web interface loads successfully in browser
WebSocket Communication	Sliders control servos and LED with no delay
Frame Streaming	JPEG frames sent via `wsCamera.binary()`
Pose Tracking	Movement feedback correct in web interface

3.2 WiFi Module

The Wi-Fi module, integrated within the Ai-M61-32S, was verified by uploading the firmware via Arduino IDE and observing the system's behavior. Upon powering the device, the module successfully initialized and established a Wi-Fi access point named `Team11` with the password `12345678`. This access point was detectable and connectable by various client devices.

The firmware includes an embedded web server that serves an HTML interface accessible through a web browser. This interface lets users view live video streams from the camera and control pan, tilt, and lighting functions via WebSocket connections. The stability and responsiveness of the Wi-Fi connection were verified by observing uninterrupted video streaming and real-time control without noticeable delay or connection loss.

3.3 Camera Module

The camera module, connected to the Ai-M61-32S, was verified by initializing it using the `esp_camera` library within the Arduino framework. Upon system startup, the camera was successfully configured with the specified settings, including resolution and frame size.

The firmware captures JPEG frames using `esp_camera_fb_get()` and transmits them to connected clients via WebSocket. The live video feed was accessible through the web interface, displaying real-time images with acceptable quality and frame rate. The module's capability to perform pose-based activity tracking, such as repetition counting and form correction, was tested and confirmed to function as intended.

3.4 Frontend

The frontend was developed using UniApp, built on a Vue 3-based architecture, and supports deployment to both H5 and WeChat Mini Program platforms. Verification was performed by compiling the project for each platform and testing it on physical devices.

Real-time posture and motion feedback were successfully displayed by consuming RESTful APIs and WebSocket streams provided by the backend. Core features—including exercise status display, live frame updates, and training result visualization — were tested across multiple screen sizes to ensure a responsive layout and consistent user experience.

The stability of the WebSocket connection was also verified under various network conditions. Users were able to view training progress, receive real-time feedback, and interact with the session through the frontend interface without noticeable lag. IP auto-configuration and camera permission handling were confirmed to function correctly.

All frontend features were validated through end-to-end testing on supported platforms, confirming full compatibility and seamless integration with backend services.

3.5 Backend

The backend system, built with Flask and Python, integrates pose estimation and fitness logic modules. Verification involved launching the server and testing the full workflow from initialization to client interaction.

The RESTful API endpoints (e.g., `/get_exercise`, `/start_session`, `/stop_session`) were tested using HTTP clients and frontend requests. Each route returned expected JSON responses with valid structure and values under both normal and edge conditions. Error handling was tested for invalid inputs and malformed requests.

WebSocket functionality was verified by pushing real-time posture estimation and session updates. Using sample video inputs and webcam streams, the server was confirmed to emit consistent posture frames and classification results. Frame latency remained within acceptable bounds, and multiple simultaneous connections were handled without packet loss.

Internal modules, including `pose_estimation`, `exercises`, and `feedback`, were tested individually and as part of the full pipeline. Unit tests and manual simulations confirmed correct rep counting, form correction feedback, and session timing logic.

4 Costs

The hardware and development cost of the Smart Fitness Coach prototype consists of three main components: labor, parts and materials, and an estimate for scaled manufacturing.

4.1 Labor Cost Estimation

The total labor cost is calculated using the formula:

$$\text{Cost} = \text{Hourly Rate} \times \text{Hours Worked} \times 2.5$$

Assuming an hourly rate of ¥60 and that each of the four team members contributed approximately 30 hours, the total labor cost is:

$$60 \times 30 \times 2.5 \times 4 = \textbf{¥18,000}$$

4.2 Parts and Materials

The project's shell was fabricated using 3D printing with lab resources provided by the university, and therefore incurred no material cost. Table 7 summarizes the hardware expenditures.

Table 7: Hardware Cost Summary (Ai-M61-32S-Based Design)

Part/Service	**Description**	**Unit Price (CNY)**	**Qty**	**Total (CNY)**
Ai-M61-32S Module	Wi-Fi, camera, and SoC combo	¥35.0	1	¥35.0
Micro Servo Motors	SG90 servo for pan/tilt	¥6.0	2	¥12.0
PCB Fabrication	Custom circuit board	¥30.0	1	¥30.0
Digital Components	Resistors, capacitors, connectors	¥10.0	1 set	¥10.0
OV2640 Camera	Camera to capture images	¥18.0	1	¥18.0
3D Printed Case	FDM print (university lab)	¥0.0	1	¥0.0
Total Hardware Cost				**¥105.0**

5 Conclusion

The Smart Fitness Coach system successfully demonstrates an affordable, scalable, and accessible real-time exercise monitoring system with an AIoT framework. The combination of a Wi-Fi-enabled PCB camera module, server-side MediaPipe inference, and interactive cross-platform UniApp frontend causes the system to render wearables obsolete while delivering impactful posture feedback to home users. The design meets main goals of portability, accuracy, and price, providing real-time capability at sub-200ms latency and continuous detection across a range of exercise types such as squats and push-ups.

As it matured during development, the project evolved into a feature-complete and modular system that cleanly separates sensing, inference, and user interface layers. Major engineering feats include the implementation of MediaPipe-based pose estimation as a component within a Flask server, real-time integration with the ESP32-CAM camera module using HTTP and WebSocket protocols, and development of an aesthetically engaging UniApp frontend that performs perfectly on Web, Android, and WeChat platforms. The system was thoroughly experimented upon across various lighting and motion conditions, with results indicating proper frame capture, posture classification, and user feedback.

Nevertheless, a few challenges and uncertainties remain. The posture detection remains lighting-, camera-, and orientation-dependent. These limitations can be mitigated in future work through temporal smoothing techniques or sensor fusion, longer support for additional action types, and more advanced pose estimation logic for broader user bases. Furthermore, on-device inference with more capable edge processors such as RK3588 can potentially deliver full decentralization with real-time performance.

5.1 Ethical Considerations

The Smart Fitness Coach system was developed in alignment with the IEEE Code of Ethics[8] and ACM Code of Ethics and Professional Conduct[9], with careful attention to user safety, data privacy, and inclusivity. The camera lens is protected by a manual plastic clamshell cover which users must open before each session. Over time, mechanical wear such as hinge loosening or lens scratching may occur, and the lithium battery compartment lacks a humidity sensor. To ensure safe operation, it is recommended that users avoid prolonged usage (over 2 hours) in damp conditions and periodically clean the device with a dry cloth.

From a data protection perspective, the system is designed to avoid transmitting any raw video or skeletal pose data to external cloud services. All processing is performed locally on the backend server. Meanwhile, the frontend application explicitly requests camera access permissions from users at runtime, providing transparency and user control. Additionally, the data storage interface supports fine-grained deletion, allowing users to manage their session records with minimal friction and maximum autonomy.

In terms of fairness and inclusivity, the posture recognition algorithm has been validated for equity across different body types or abilities. Users with disabilities may still be recognized for adaptive movements. However, this remains an open area for future improvement.

5.2 Broader Impacts

Smart Fitness Coach can have a significant impact globally and in society. It provides a low-cost and portable fitness solution to consumers in rural or resource-constrained environments, where it may not be possible to have access to gyms or personal trainers. The small hardware mitigates power consumption and environmental effect, and the flexible software architecture allows for localized deployment irrespective of cloud infrastructure. In an age of post-pandemic, when home fitness remains a more and more relevant topic, the project is a timely contribution to safe, interactive, and intelligent exercise technologies.

From a commercial standpoint, the Smart Fitness Coach system offers strong potential for integration into gym environments. By embedding the camera module directly into fitness equipment—such as treadmills, squat racks, or cable machines—and pairing it with dedicated companion software, fitness centers can provide members with real-time posture feedback, automated form correction, and personalized workout tracking without the need for trainers to be present at all times. This enhances the overall value of gym services while improving user engagement and safety. Additionally, such integration could enable data-driven fitness plans and progress dashboards, offering gyms a competitive edge in delivering intelligent, tech-enhanced experiences to their clientele.

References

[1] DCloud Technologies. “UniApp Quickstart (CLI Mode).” (2025), [Online]. Available: https://uniapp.dcloud.net.cn/quickstart-cli.html (visited on 05/18/2025).

[2] Ultralytics. ““yolo11”.” Ultralytics Documentation. (2025), [Online]. Available: https://docs.ultralytics.com/models/yolo11/ (visited on 04/13/2025).

[3] WZH2014825. ““the stm32f103 controls the rotation of the servo”.” Servo Blog. (2020), [Online]. Available: https://blog.csdn.net/WZH2014825/article/details/107360318 (visited on 02/10/2025).

[4] D. Systèmes. “Solidworks user guide.” (2025), [Online]. Available: https://www.solidworks.com/support/user-guide (visited on 04/10/2025).

[5] I. Gibson, D. Rosen, and B. Stucker, *Additive Manufacturing Technologies: 3D Printing, Rapid Prototyping, and Direct Digital Manufacturing*. Springer, 2015. [Online]. Available: https://doi.org/10.1007/978-1-4939-2113-3 (visited on 04/10/2025).

[6] R. L. Norton, *Machine Design: An Integrated Approach*. Pearson Education, 2020. [Online]. Available: https://www.pearson.com/store (visited on 04/10/2025).

[7] X. Chen and Y. Zhang, “Mortise-tenon structures in modern mechanical design,” *Journal of Mechanical Engineering*, 2018. [Online]. Available: https://doi.org/10.1016/j.jmech.2018.03.007 (visited on 04/10/2025).

[8] IEEE. “Ieee code of ethics.” (2016), [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (visited on 04/13/2025).

[9] Association for Computing Machinery. “ACM Code of Ethics and Professional Conduct.” (2018), [Online]. Available: https://www.acm.org/code-of-ethics (visited on 04/13/2025).