Interactive Desktop Companion Robot for Stress Relief

By

Jiajun Gao (jiajung3)

Yuchen Shih (ycshih2)

Zichao Wang (zichao3)

Final Report for ECE 445, Senior Design, Spring 2026

TA: Haocheng Bill Yang

May 4, 2026

Project No.6

Abstract

Jarvis is an ESP32-S3 desktop companion robot designed to provide short, low-effort stress-relief interactions during desk work. The final system integrates offline wake-word detection, streamed voice communication with a local Mac server, ASR, Gemini-based response generation, Edge TTS playback, MCP tool calls for robot actions, a geometric LCD face, Wi-Fi human-control mode, and local motor-safety overrides. Verification showed that the voice pipeline met the main latency requirement, with direct robot-command turns reaching first TTS audio at about 2.2 seconds and conversational turns at about 2.7 seconds median latency. MCP, LLM, and TTS stages operated successfully in the recorded test sessions. The main remaining limitation is short-command ASR word accuracy during battery-powered operation, which appears to be related to analog microphone power integrity. Overall, the project demonstrates a complete embedded AI robot prototype while identifying clear hardware improvements for a second PCB revision.

Contents

1. Introduction

1.1 Problem

Prolonged seated work and continuous screen exposure can contribute to mental fatigue, accumulated stress, and reduced cognitive efficiency. However, many existing solutions address this problem only partially. Passive desk toys are convenient and easy to use, but they do not respond to the user's context or provide meaningful interaction. Consumer robots, on the other hand, are often too large, mechanically complex, or not designed with desktop safety constraints in mind. Software-based wellness applications may also be counterproductive, since they often require additional screen time at the exact moment when the user needs a simple and low-friction break.

This creates an engineering opportunity for a compact embedded system that can remain on a desk, provide short conversational interactions, move in a controlled manner, and communicate its current state without becoming another distracting productivity application. To be effective, the system needs to feel responsive during interaction, operate safely within the limited space of a desktop environment, and remain transparent about how voice input is processed.

The desktop environment introduces an important engineering constraint that is often underestimated. A robot operating on a desk has very little room for error if it moves in the wrong direction. A fall could damage the robot itself, nearby electronics, or the user's workspace. Unlike a floor robot, a desktop robot must treat edge detection as a core safety function rather than an optional navigation feature. It also cannot assume that the user will constantly watch a screen to understand its status. For this reason, Jarvis was designed around three main requirements: it must be interactive, physically conservative, and able to communicate its state through both audio and visual feedback.

Overall, this project sits at the intersection of embedded systems, human-robot interaction, real-time safety control, and cloud-assisted AI. The final report focuses not only on the final set of features, but also on the engineering tradeoffs that made the working prototype possible. These include local wake-word detection, a persistent WebSocket connection, a server-side voice processing pipeline, device-side authority over motor control, and a simple visual face that makes the robot's state easy to understand without becoming visually distracting.

1.2 Project Overview and Final Solution

Jarvis is a compact interactive desktop companion robot built around the ESP32-S3. The system is designed to provide low-effort conversational interaction and simple physical feedback through voice input, dynamic visual expressions, wireless connectivity, and controlled motion. In the final prototype, Jarvis listens for the wake word "Jarvis" locally, streams compressed audio to the backend only after activation, receives streamed text-to-speech audio in response, and exposes robot actions as MCP tools that can be triggered by either fast-path intent handlers or the language model. The completed robot includes a geometric animated face on an ST7735 LCD, differential-drive motion, front and bottom TCRT5000 infrared safety sensors, and a Wi-Fi manual-control mode for browser-based driving.

The final build differs from the initial proposal in two main ways. First, the working prototype uses a local Mac server instead of a fully cloud-based server. This made the ASR, LLM, and TTS pipeline easier to test, observe, and tune during development. Second, the safety system uses digital TCRT5000 infrared sensors rather than the originally proposed ToF modules. Although the sensing hardware changed, the core safety objective remained the same: the robot must be able to detect forward obstacles and desk-edge conditions quickly enough to override unsafe motor commands.

A typical interaction begins when the user says the wake word. The ESP32-S3 detects the wake word locally and then starts an audio session with the backend. The backend receives the streamed audio, performs voice activity detection, generates a final ASR transcript, and routes the command either through a fast-path intent handler or through the language model. For motion-related commands, the backend does not directly control the motor GPIO pins. Instead, it requests a named MCP tool, such as robot.move. The ESP32-S3 then checks the request against the current mode and safety state before allowing any motor command to reach the motor driver. This separation is important because language model responses can be slower or less predictable, while motor safety requires deterministic local control.

Jarvis is intentionally limited in both physical behavior and interaction scope. The robot is not designed for full home navigation or continuous autonomous operation. Instead, it focuses on short conversational exchanges, small motion responses, readable visual expressions, and safe operation on a desk. By narrowing the scope in this way, the team was able to build a complete end-to-end prototype rather than only demonstrating separate subsystems.

1.3 High-Level Requirements

The seven high-level requirements were selected to evaluate the full user experience rather than isolated technical functions. HR1 and HR2 focus on whether the robot can connect, detect activation, and respond quickly enough to feel interactive. HR3, HR4, and HR5 evaluate whether the robot can operate safely on a desk without creating a physical risk to itself or the surrounding workspace. HR6 measures whether the face display can serve as a clear communication channel for robot state and feedback. HR7 evaluates whether the integrated hardware can operate from battery power without unexpected resets. Together, these requirements shaped the design around a balance of conversational interaction, real-time safety control, and electrical reliability.

Table 1 High-Level Requirements

ID	Requirement	Final Evidence
HR1	Wi-Fi/cloud connection within 5s	Persistent WebSocket backend connection enabled live voice sessions on campus Wi-Fi.
HR2	Voice response latency under 6s	Average end-to-end latency was 2503 ms across 24 turns; most responses were Fast (< 2s) or OK (2-3.5s).
HR3	Lift and edge detection through bottom IR	Bottom IR cliff detection latches emergency stop and blocks motion until floor is restored.
HR4	Front obstacle detection with 90 percent stop success	Front IR obstacle state vetoes forward-like motion and was validated in functional obstacle tests.

HR5	Emergency stop within 50 ms of hazard detection	High-priority IR/safety tasks continuously override motor PWM outputs.
HR6	Correct LCD expressions across all operating states	FaceLcdDisplay maps listening, thinking, speaking, moving, music, idle, and error states to animated expressions.
HR7	Stable battery operation for 30 minutes without reset	Battery operation was functional but exposed microphone rail-sag limitations; treated as partial.

2. Design

The system is organized as a modular robot paired with a local voice backend. On the hardware side, the Face and Body boards separate the conversation and user-interface functions from the drive and power subsystems. On the software side, the ESP32-S3 is responsible for wake-word detection, audio streaming, display control, and safety-critical motion authority, while the local Mac backend handles the more computationally intensive tasks, including VAD, ASR, LLM processing, and TTS generation.

The most important design principle is that the backend can request actions, but the embedded robot has final authority over whether those actions are safe to execute. The backend is optimized for semantic interpretation and speech generation, while the ESP32-S3 is optimized for deterministic responses to local hazards. This separation helps protect the system from network delay, LLM latency, and ASR errors. Even if the backend misinterprets a command or the language model requests motion at an unsafe time, the local safety manager still has the final opportunity to block the motor command.

A second design principle is that long-latency operations should either be streamed or decoupled from safety-critical control. Audio is streamed to the backend instead of being uploaded as a complete file, and TTS audio is returned in chunks rather than waiting for a full speech file to be generated. Behaviors such as dance and follow mode are started asynchronously so that the tool call can return quickly. These design choices reduce the user's perceived delay and help keep the robot responsive even when a cloud model or external API takes longer than expected.

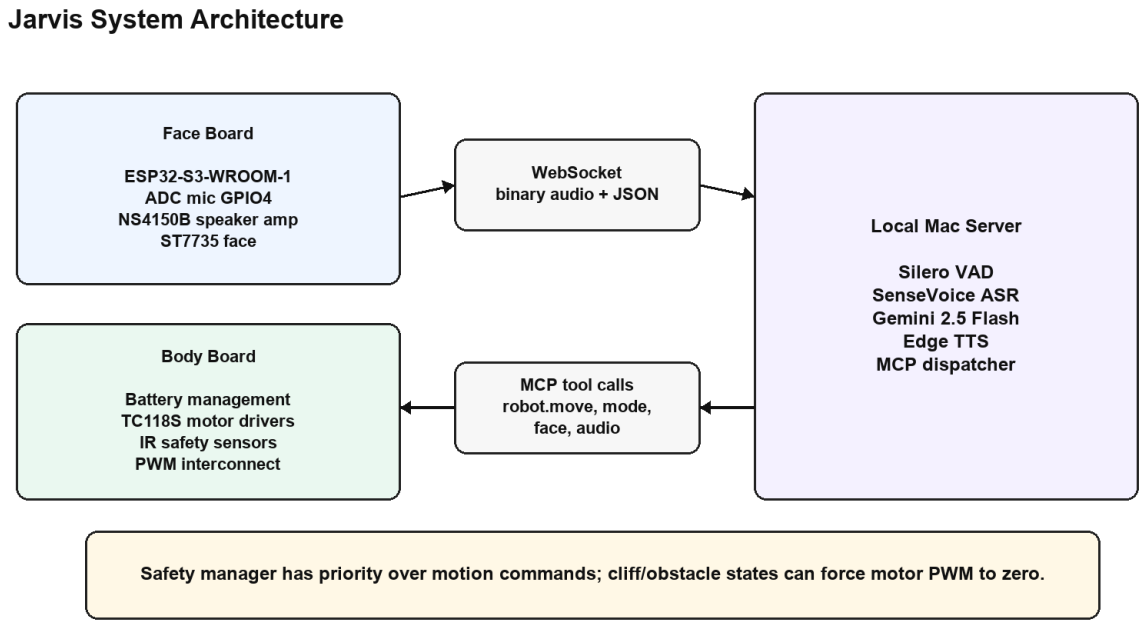

Figure 1 Overall Jarvis system architecture.

2.1 Hardware Architecture

The system uses a dual-PCB modular architecture. The Face Board integrates the ESP32-S3-WROOM-1 module, the ADC microphone connected to GPIO4, the speaker power amplifier, and direct control of the ST7735 80 × 160 SPI LCD. The speaker amplifier uses an NS4150B with an approximate gain of 7.2, which provides sufficient local volume for text-to-speech output while keeping the audio circuit

compact. The Body Board functions as the drive and power hub. It handles motor control, power isolation, battery management, and the physical connections to the drivetrain and infrared sensors.

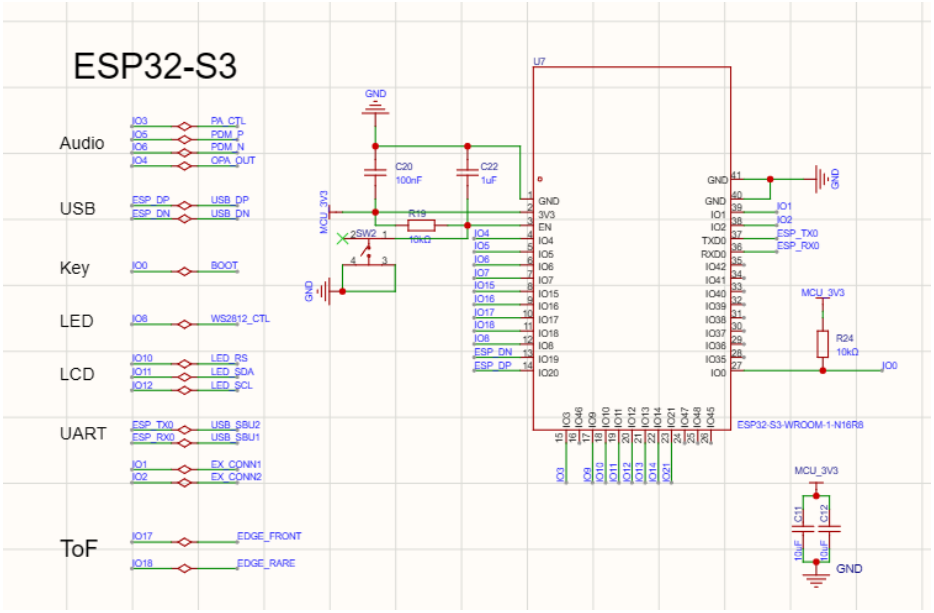

Fig 1. ESP32-S3 (Face Board)

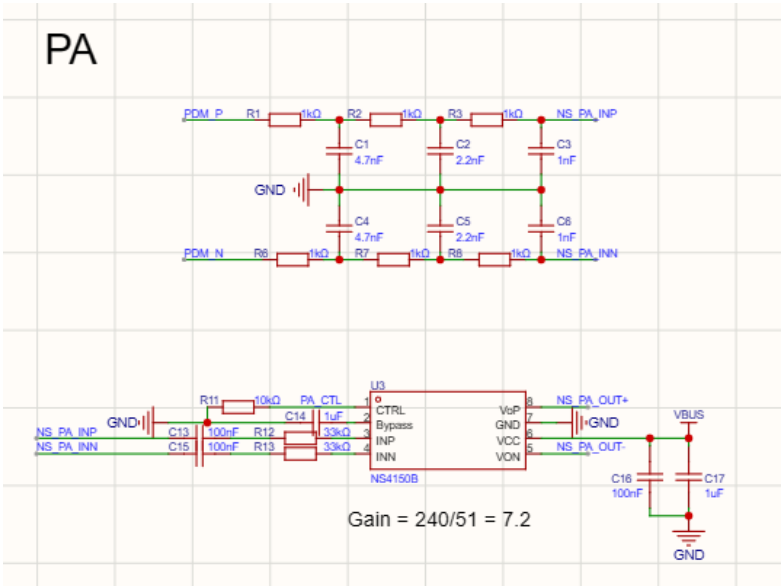

Fig 2. Power Amplifier (Face Board)

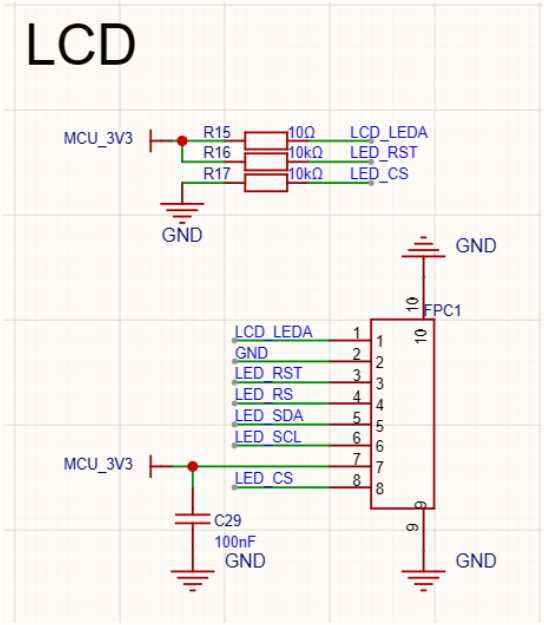

Fig 3. LCD (Face Board)

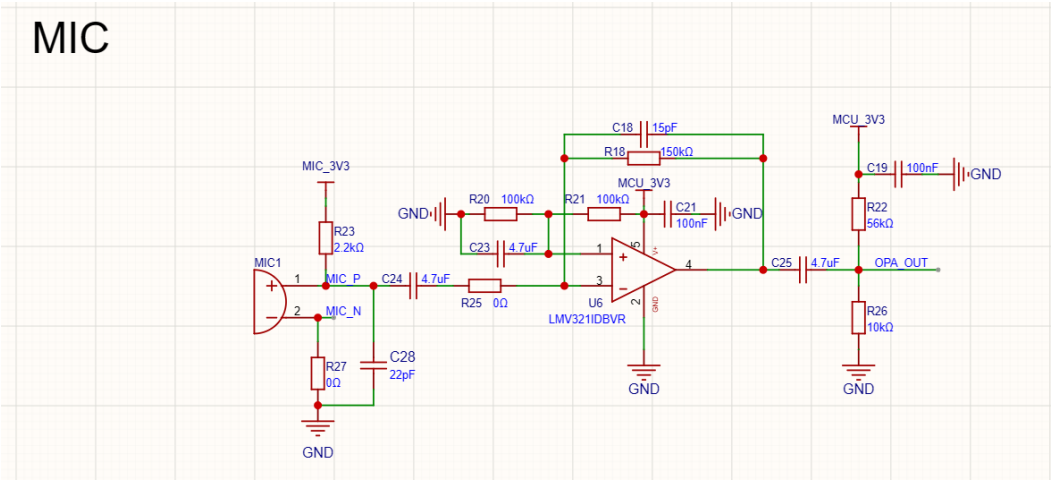

Fig 4. MIC (Face Board)

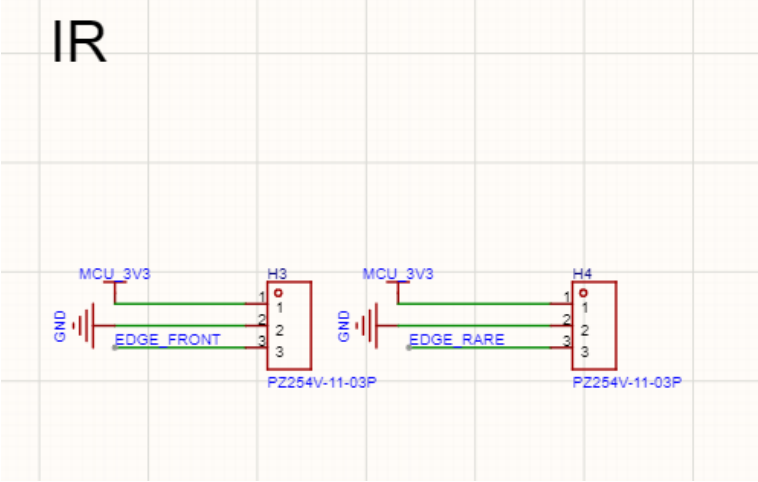

Fig 5. IR senor (Face Board)

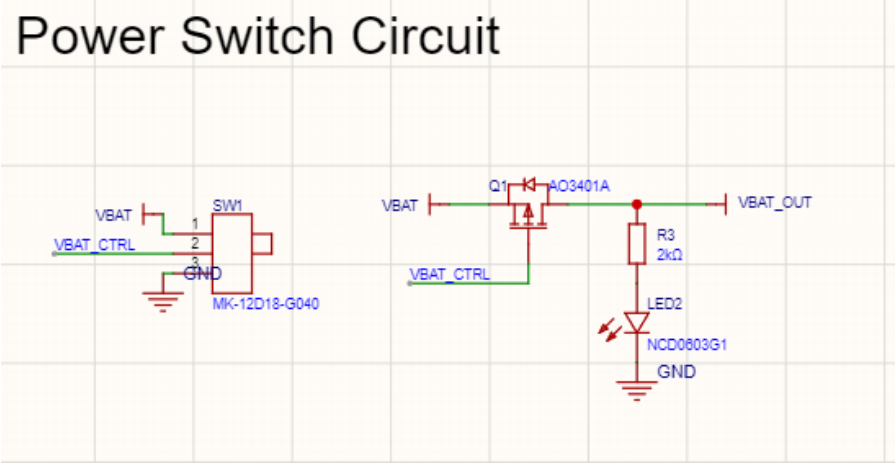

Fig 6. Power Switch Circuit (Body Board)

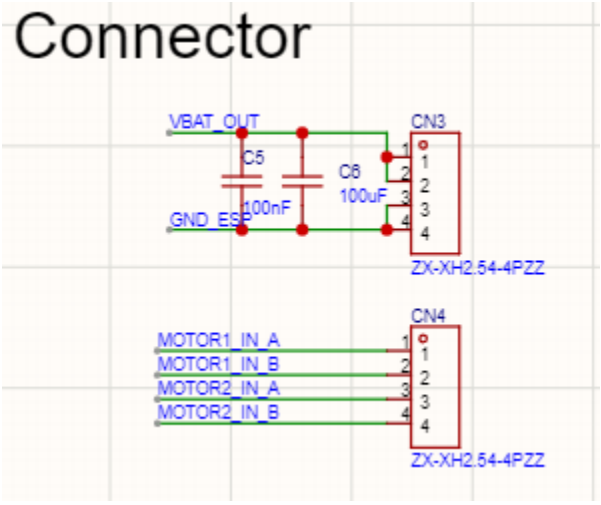

Fig 7. Dupont Cable Connector (Body Board)

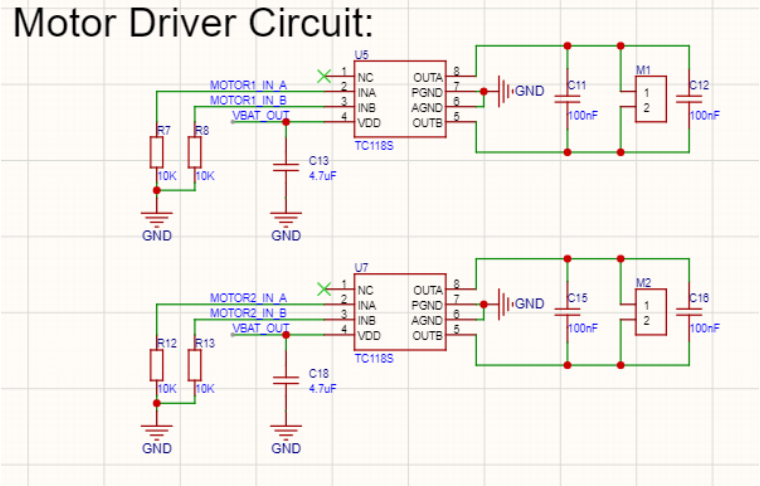

Fig 8. Motor Driver Circuit (Body Board)

The two boards are connected using Dupont cables. This connection method made the boards easier to test independently and allowed the Face Board to send PWM motor-control signals to the Body Board without placing the audio and display circuitry directly next to the highest-current motor-drive path. The firmware also keeps an optional OV5640 camera pin mapping behind the ECE445_ENABLE_CAMERA flag, but camera operation was not included in the verified final demonstration.

The modular board split was especially helpful during bring-up and debugging. The Face Board could be validated first using the wake-word detection, microphone, speaker, LCD, and Wi-Fi functions before the Body Board was fully integrated. Similarly, the Body Board could be tested using motor outputs and IR sensor thresholds without requiring the full voice pipeline to run. This reduced debugging ambiguity. For example, if audio failed, the issue was more likely related to the Face Board or backend. If motion failed, the issue was more likely related to the Body Board, motor wiring, or safety logic.

The Dupont interconnect is not ideal for a final commercial product because loose wires can introduce resistance, noise pickup, and mechanical reliability concerns. However, for a senior design prototype, it provided useful flexibility. The team could reassign signals, probe PWM lines, and isolate the two boards during testing. In a future revision, the loose interconnect should be replaced with a keyed board-to-board connector or a short cable harness with defined ground returns.

Table 2 Final Embedded Interfaces

Block	Final Implementation	Purpose
Face Board controller	ESP32-S3-WROOM-1 with custom ece445-custom-s3 firmware profile	Coordinates audio, display, Wi-Fi, MCP tools, and safety authority.
Microphone	ADC microphone on GPIO4	Captures wake-word and speech audio.
Speaker	Speaker path with NS4150B power amplifier, gain about 7.2	Plays streamed TTS and local prompts.
Display	ST7735 80x160 SPI LCD	Shows geometric state-based face.

Body Board motor drive	Dual TC118S motor drivers driven through PWM interconnect	Differential-drive movement with board-level power isolation.
Safety sensors	Front and bottom TCRT5000 digital IR sensors on GPIO15 and GPIO16	Detect obstacle and floor/cliff state.

2.2 Software Architecture

The software stack is organized into four main layers. First, the device firmware listens for the wake word and streams compressed audio to the backend through a persistent WebSocket connection. Second, the local Mac backend performs voice activity detection to separate user speech from background noise. Third, the detected speech is converted to text and routed through either Fast-Path ASR handling or Gemini 2.5 Flash for command understanding. Fourth, the selected response is converted back into speech using streaming TTS and sent to the robot in small audio chunks to reduce perceived latency.

The Model Context Protocol bridge connects the language model to the robot hardware through WebSocket-based JSON-RPC style tool calls. Motion-related tools include robot.move, which accepts direction and speed inputs and returns a structured response that includes infrared safety status. Other tools handle UI updates, mode switching, and Wi-Fi browser-based manual control. This allows the system to switch between AI control and human control when manual driving is safer or more useful than autonomous motion.

The firmware is divided into smaller managers rather than being implemented as one large control loop. The audio path manages microphone sampling, wake-word detection, audio streaming, and speaker playback. The robot layer manages motion control, operating modes, IR sensor sampling, safety filtering, dance behavior, follow behavior, and Wi-Fi manual control. The display layer manages the LVGL face and updates it based on the current application state. This structure helped keep timing-sensitive logic easier to read and made it easier to test individual behaviors such as cliff latching, front-obstacle vetoing, and rejection of unsafe commands during human-control mode.

The backend follows a similar staged structure. VAD determines when a user utterance has ended. ASR converts the completed utterance into text. Fast-path handling captures simple commands that do not require a full language model call. The LLM is used for open-ended conversation and ambiguous requests. Finally, TTS converts the response into audio for playback on the robot. The backend also records timing information for each stage, allowing latency to be analyzed after a live session rather than estimated only through observation.

2.3 Motion, MCP Control, and Human-Control Mode

Robot actions are exposed through MCP tools registered by the firmware. The primary motion command is robot.move, which supports forward, backward, left, right, spin_left, spin_right, and stop. Each command returns the current mode, safety flags, and a response message, allowing the language model to explain whether the requested motion was accepted or blocked. The robot.set_mode tool switches the robot between AI-control mode and human-control mode. When human-control mode is active, the Wi-Fi web controller becomes available, while voice-driven motion is blocked except for stop commands or recovery-safe behavior.

The differential-drive motor layer uses PWM signals sent from the Face Board to the motor drivers on the Body Board. Before the wheel outputs are updated, each motion command is checked by the mode manager and safety manager. This ensures that slow or uncertain AI decisions never directly control the motors. Instead, the ESP32-S3 retains final authority over motion, and watchdog-style safety logic prevents unsupervised or unsafe movement.

The return value of robot.move is more than a simple acknowledgement. It includes the current operating mode, front-obstacle flag, bottom-floor flag, and safety reason. This allows the backend or language model to respond with grounded explanations, such as “I cannot move forward because the cliff safety is latched,” rather than producing a generic failure message. In practice, this also made debugging easier because the same status information used to protect the robot was visible through the conversational interface.

Human-control mode was included because not every safe action should be mediated by the language model. During demos, manual driving from a phone browser gives the operator more precise control while still preserving the same local safety overrides. The mode separation also prevents competing command sources from interfering with each other: AI motion is rejected during human-control mode, and browser-based motion is ignored when AI mode is active.

2.4 Safety Sensing and Override Logic

The safety subsystem uses infrared sensors for both front obstacle detection and downward cliff or edge detection. IR sensing was selected because it provides a fast response, a simple digital GPIO interface, and a low-cost implementation. However, it also has limitations, including sensitivity to ambient lighting and reduced reliability on certain matte or low-reflectivity surfaces. These limitations are one reason that ToF sensors remain part of the future improvement plan.

The safety timing budget is based on the robot’s maximum speed. At $v_{max} = 0.1$ m/sand a 20 ms safety polling interval, the maximum distance traveled between safety checks is $d = v_{max} \times t = 0.1 \times 0.020 = 0.002$ m, or 2 mm. This means the robot checks for hazards frequently enough that it should only move a very small distance between safety updates. The IR safety flags run at high RTOS priority and continuously override motor PWM outputs when hazards are detected. Cliff detection is given higher priority than front obstacle detection. When the bottom sensor loses floor reflection, the robot treats it as a cliff condition and triggers backoff followed by an emergency latch.

The safety logic intentionally handles different hazards in different ways. A front obstacle is mainly dangerous during forward motion, so the robot can still move backward or rotate in order to recover. A cliff condition is more serious because continued motion near a desk edge could cause the robot to fall. For that reason, cliff detection places the robot into an emergency stop state after the backoff behavior. The latch is released only after the bottom sensor reports a safe floor condition for the required clear interval.

The digital IR modules also require threshold tuning. Each module includes an onboard potentiometer that sets the comparator trip point. During bring-up, the front sensor was adjusted to trigger when an object was close enough to be relevant at desktop driving speed. The bottom sensor was adjusted to report floor-present during normal operation and floor-lost when positioned over an edge. The software also applies debounce logic so that a single noisy reading does not immediately change the stable safety state.

2.5 Visual Expression and User Feedback

The face display uses a minimalist geometric design built from circles, arcs, and lines instead of complex graphics or emoji-style assets. The display subsystem is implemented through FaceLcdDisplay, a full-screen LVGL renderer with double buffering. Its animations transition over roughly 200–500 ms and target 30–60 FPS, allowing the face to remain responsive and visually active while the backend is listening, thinking, or speaking.

The face is driven by system state, including listening, thinking, speaking, moving, music, idle, and error modes. This gives the user immediate visual feedback about what the robot is doing and reduces the need to rely on audio feedback alone.

The visual design was intentionally kept abstract. A realistic face would require more graphical assets and could make expression transitions appear unnatural on a small 80 × 160 display. In contrast, a geometric face is easier to render with consistent timing while still communicating attention, activity, and emotion-like states. For example, the listening state can be represented with an alert open expression, the thinking state with a calmer processing expression, the speaking state with mouth movement, and the moving state with a more active pose.

This approach also reduces storage and asset-management complexity. Since the face is generated from LVGL primitives, the firmware does not need to load image packs or manage large sprite sheets. This keeps memory usage predictable on the ESP32-S3 and allows state transitions to be controlled through parameters rather than fixed image assets.

2.6 Power and Integration Considerations

The project uses a compact battery-powered embedded architecture, which makes power integrity an important design concern. Although the final software pipeline is stable, battery-powered operation revealed a weakness in the analog microphone path. Voltage rail sag during Wi-Fi transmission, speaker playback, and motor activity can reduce ASR word accuracy, especially for short commands. A future hardware revision should add stronger bulk capacitance, improve regulator sizing, shorten sensitive analog signal paths, and better isolate high-current motor and audio amplifier activity from the microphone supply and reference.

A second integration challenge was acoustic echo caused by the close physical placement of the speaker and microphone. Since the prototype does not include a dedicated acoustic echo cancellation DSP, the team used a software-based mitigation strategy. This includes a server-side time-windowing Echo Guard filter and a firmware grace period that prevents the microphone from reactivating too aggressively immediately after TTS playback.

The power issue is important because the latency pipeline can appear successful even when the user experience is degraded. If the ASR system receives noisy or clipped audio, the backend may respond quickly but respond to the wrong transcript. For this reason, the final analysis separates pipeline completion from word-level accuracy. The backend can reliably complete ASR, call the LLM, trigger tools, and synthesize TTS, but the quality of the physical analog front end still determines whether the recognized text matches what the user actually said.

The echo issue has a similar system-level impact. A loud speaker and a nearby microphone are convenient for a small desktop robot, but they also create feedback paths. Echo Guard reduces false ASR triggers by ignoring suspicious audio windows after TTS playback, while the firmware grace period delays

aggressive listening after speech output. These mitigations were sufficient for the prototype, but they also reduce full-duplex interaction because the robot is less able to listen naturally while it is speaking.

3. Design Verification

Verification focused on integrated end-to-end behavior rather than only isolated component demonstrations. The robot needed to wake, listen, interpret commands, respond through speech, execute actions when appropriate, and remain safe throughout the interaction. For this reason, the most important verification artifacts are the live backend metrics report, MCP tool-call logs, IR safety behavior, and final functional demonstration results. The metrics figures in this section are based on a recorded 24-turn test session and show both average system performance and tail-latency behavior.

3.1 Voice Pipeline Verification

The main voice requirement was that the robot should begin speaking within 6 seconds during normal interactions. Two recorded sessions support this requirement. In a 16-turn session, direct MCP turns reached first TTS audio at about 2.2 seconds, while conversational LLM turns reached first TTS audio at about 2.7 seconds median ASR-final-to-first-TTS latency, excluding one external RSS plugin outlier. A later 24-turn metrics report showed a 2.01-second median ASR-to-first-TTS latency and a 2.50-second mean latency, with p95 latency at 5.29 seconds. The p95 value is higher because external tool calls and longer conversational turns still create tail latency. Functional testing on campus Wi-Fi also produced consistent sub-4-second end-to-end latency for most normal voice interactions.

The latency distribution shows why streaming was necessary. The robot does not need to wait for a complete paragraph of LLM output or a full TTS audio file before the user hears a response. Instead, the first useful audio chunk is sent as soon as the response begins to become available. This makes the system feel faster than a purely sequential pipeline using the same backend components. The timeline plot also shows that latency is not constant across all turns. Short commands and direct MCP actions are faster, while plugin-backed queries and longer conversations increase tail latency.

LLM time-to-first-token should be interpreted carefully. Only turns that actually require an LLM call have TTFT values. Direct robot commands can bypass the LLM path, so the 14 out of 24 TTFT count does not mean the LLM failed on the other turns. Instead, it shows that fast-path handling successfully avoided unnecessary LLM calls for some interactions.

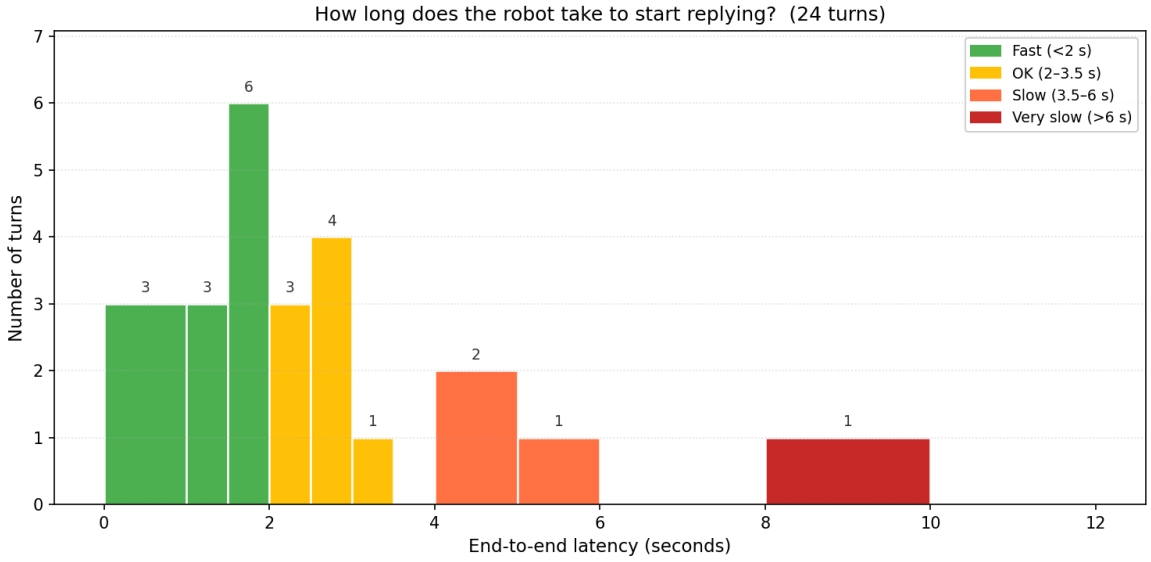

Figure 2 End-to-end ASR-to-first-TTS latency distribution from the recorded metrics report.

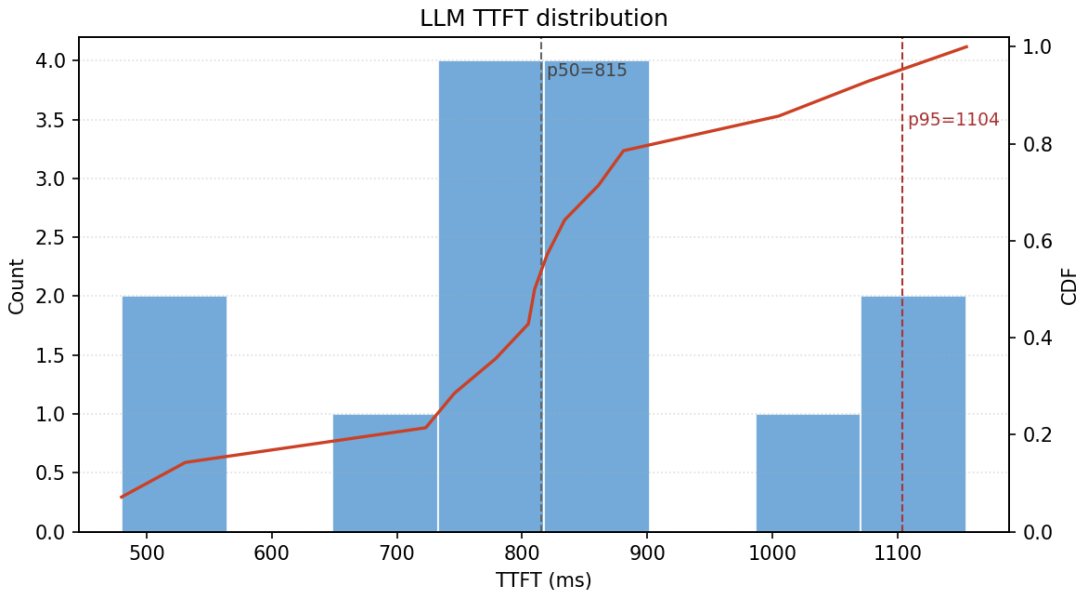

Figure 3 LLM time-to-first-token distribution from the metrics report.

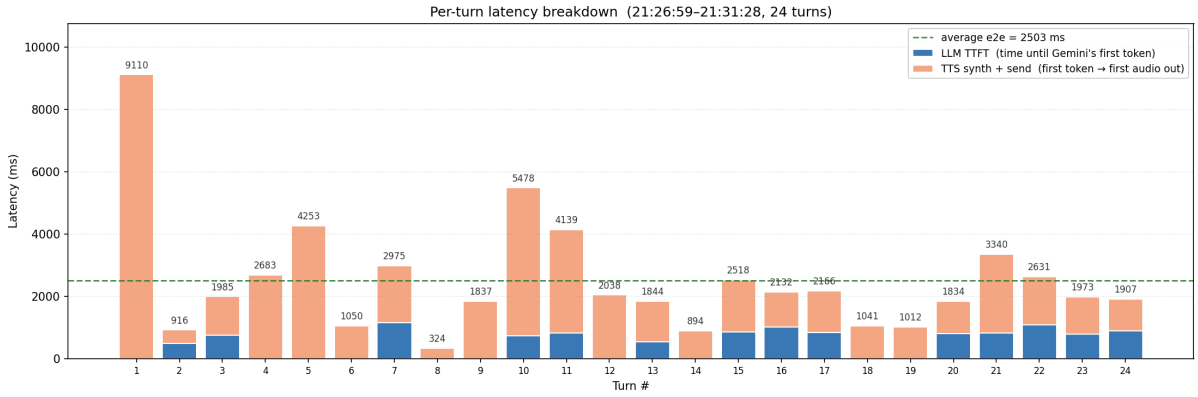

Figure 4 End-to-end latency timeline across the recorded 24-turn session.

Table 3 Voice Pipeline Verification

Metric	Measured Result	Verdict
Wake greeting	Cached greeting sent within the same second as wake detection; server-side latency under 100 ms.	Pass
Direct robot command latency	About 2.2 s median for direct-MCP robot commands.	Pass
Conversational latency	About 2.7 s median in the 16-turn session; 2.01 s median in the 24-turn metrics report.	Pass
LLM first-token time	Observed for 14 of 24 turns (58 percent, because direct-tool turns do not use an LLM TTFT); p50 was 815 ms.	Pass

Per-stage completion	ASR commit, end-to-end TTS, MCP tool calls, and TTS synthesis completed at 100 percent in the metrics report.	Pass
ASR word accuracy	About 95 percent on wall power but about 55 percent on battery because rail sag reduced analog-front-end SNR.	Partial

3.2 MCP and Robot Tool Verification

MCP tool dispatch was verified through repeated robot command sessions. The 16-turn session reported 100% MCP success, and the later 24-turn metrics report logged 13 tool calls with zero timeouts. The most important robot tool, robot_move, averaged 49 ms in the 24-turn report, making device-side command execution much faster than ASR or TTS startup. Tool execution latency was mainly affected by external API calls, such as get_weather at 2740 ms, while internal robot tools remained fast. For example, robot_move averaged 49 ms and robot_follow_start averaged 28 ms.

This result supports the MCP architecture for controlling the robot's physical actions. A tool call that completes in tens of milliseconds is short enough that it does not dominate the user's perceived interaction time. It also shows that future optimization should focus more on ASR quality, TTS startup, and external plugin delays before optimizing the robot_move implementation itself. The tool-frequency plot is also useful because it shows which tools were actually exercised during testing, preventing the average latency results from being misread based on rarely used paths.

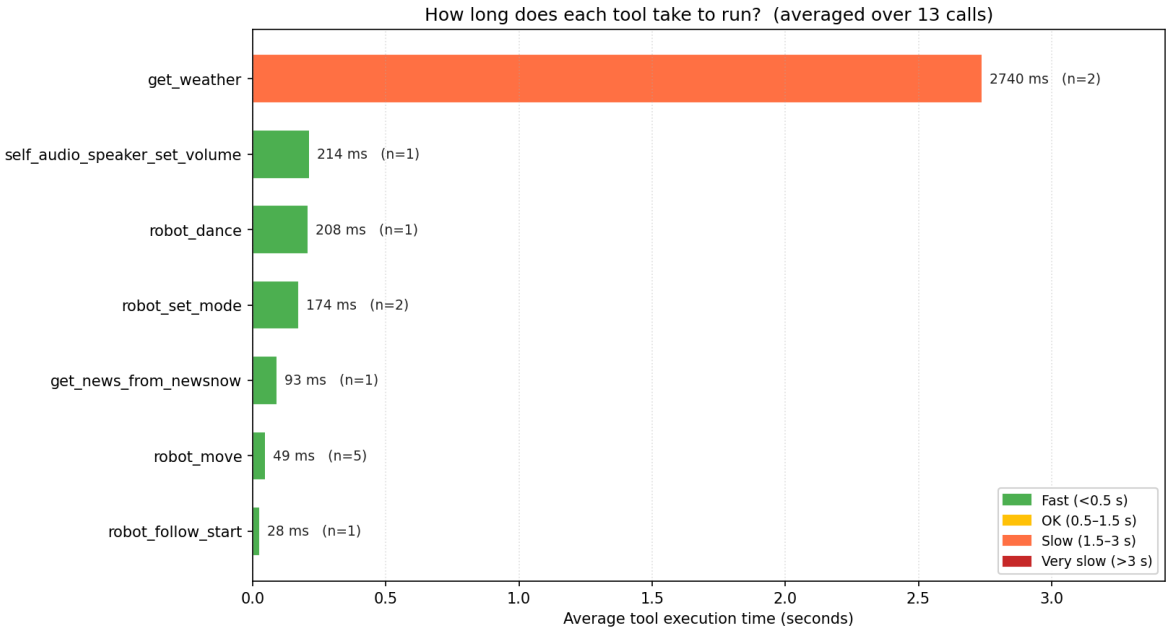

Figure 5 MCP and plugin tool latency by tool type.

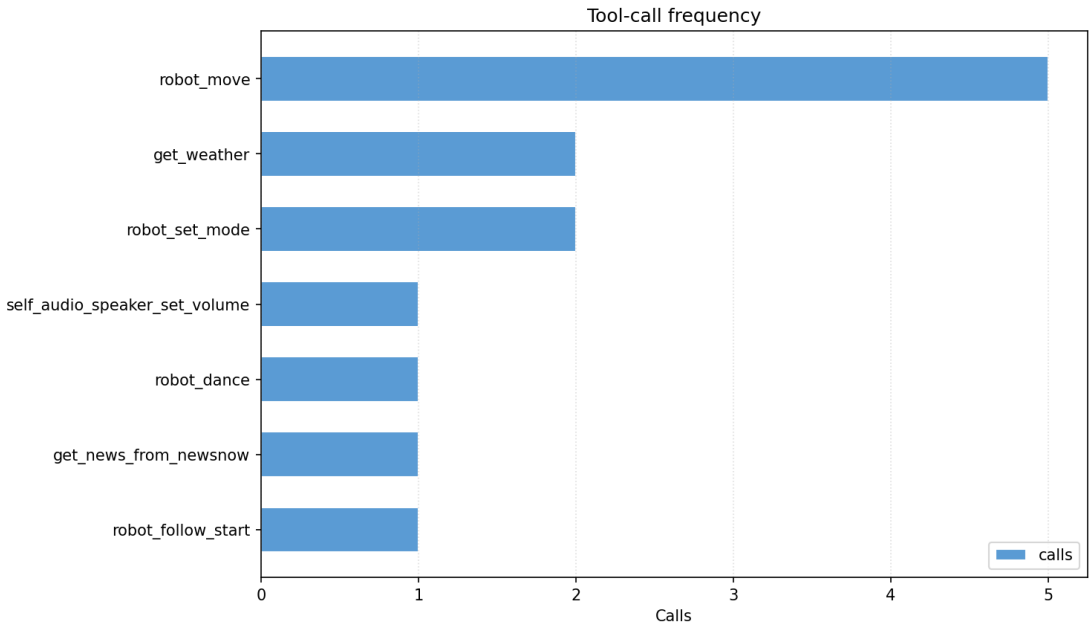

Figure 6 MCP and plugin tool call frequency during the recorded session.

Table 4 MCP Tool Results

Tool or Stage	Recorded Result	Engineering Meaning
robot_move	5 calls, average 49 ms, p50 32 ms, max 100 ms in the later metrics report.	Motion command dispatch is not a latency bottleneck.
robot_set_mode	2 calls, p50 174.5 ms.	Human-control handoff is responsive.
robot_dance	1 call, 208 ms in later metrics report; 31 ms in the 16-turn session.	Long behavior starts quickly and runs asynchronously.
get_weather	2 calls, average 2740 ms.	External APIs dominate tool latency when they are used.
MCP stage	100 percent success in the 16-turn session; zero tool timeouts in the 24-turn report.	Tool protocol was reliable during recorded testing.

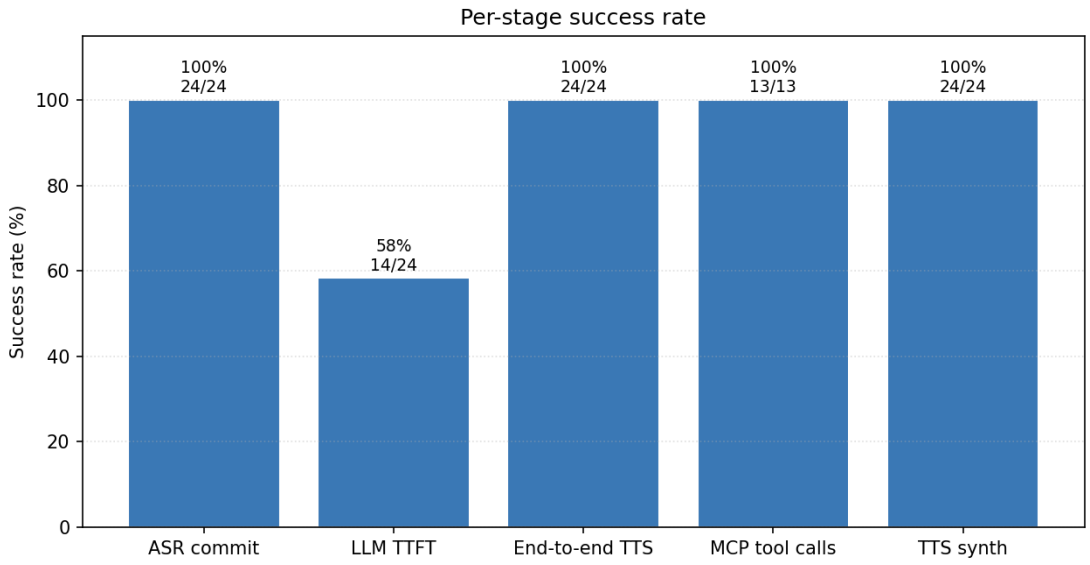

Figure 7 Per-stage success-rate chart from the metrics report.

3.3 Safety Verification

Safety was verified through firmware-level behavior testing and the breadboard motion prototype described in the project notes. Functional testing demonstrated 100% cliff detection across multiple 10-minute continuous tests. The safety architecture correctly gives hazard classification priority over normal movement. The bottom sensor is designed to fail safe: when the floor signal is lost, the robot classifies the condition as a cliff, backs off, enters an emergency stop latch, and blocks further non-stop motion until the floor condition is restored. The front sensor blocks forward-like movement when an obstacle is detected, while still allowing backward and spin commands for recovery. Human-control mode was also verified. When this mode is enabled, AI voice-motion commands are blocked so that manual Wi-Fi driving remains the active control source.

The safety tests were designed around the actual risks of a desktop robot. For cliff detection, the key result is not only that the sensor detects the desk edge, but also that motor output is overridden quickly enough to prevent higher-level commands from continuing unsafe motion. For front obstacle detection, the requirement is less severe because an obstacle in front of the robot can usually be recovered from by backing away or rotating. The final behavior reflects this difference by treating cliff detection as a latched emergency condition and front obstacles as directional motion constraints.

The human-control test also verifies command-source arbitration. In a mixed AI and manual-control robot, a browser joystick and a language model could otherwise issue conflicting commands. By making the control mode explicit and rejecting voice-driven motion during human-control mode, the system avoids a class of unsafe interactions that would not be visible during single-source testing.

Table 5 Safety Requirement Verification

Requirement	Verification Method	Status
Bottom sensor detects desk edge	Move robot or sensor over desk edge and confirm bottom_floor becomes 0 and reason becomes cliff_detected.	Yes

Cliff condition stops motion	Confirm safety manager forces stop/backoff and latches emergency stop after cliff rising edge.	Yes
Front obstacle blocks forward motion	Hold object in front and confirm forward commands are vetoed while backward and stop remain available.	Yes
Manual and AI modes do not fight	Switch to human_control and confirm voice motion is blocked while browser driving is enabled.	Yes

3.4 Known Limitation

The main remaining uncertainty is ASR word accuracy for short commands, especially during battery-powered operation. Software reliability was strong in the recorded reports, with 100% success across MCP tooling and TTS, while the LLM API path reached about 90% reliability in broader testing. However, short phrases such as “Jarvis” or “forward” were sometimes misrecognized when the robot was running on battery power. The observed drop from about 95% ASR accuracy on wall power to about 55% on battery suggests that supply-rail voltage sag may be affecting the analog front end and reducing acoustic SNR.

A second limitation is acoustic echo caused by the close physical placement of the speaker and microphone. During TTS playback, the speaker output can be picked up by the microphone and interpreted as new user speech. The current Echo Guard filter and post-TTS grace period reduce false triggers, but a dedicated acoustic echo cancellation DSP would be a stronger long-term solution for full-duplex audio and barge-in support.

These limitations are useful because they identify the next engineering bottlenecks. At the beginning of the project, latency appeared to be the most difficult part of building an AI voice robot. In the final prototype, latency was acceptable, while audio quality under battery power became the larger barrier to reliable interaction. This reflects an important embedded-system lesson: once the software architecture works, analog layout, grounding, decoupling, and power delivery can become the dominant system variables.

These limitations also shape how the results should be interpreted. A 100% ASR commit rate means the ASR service produced final text for every turn, but it does not mean every word was recognized correctly. Similarly, a 100% MCP success rate means tool calls returned valid responses, but it does not mean every user command was understood perfectly before the tool was selected.

4. Costs

The cost analysis separates materials from labor because the prototype is an academic engineering build rather than a manufactured product. Parts cost reflects the approximate component and fabrication cost needed to reproduce the prototype. Labor cost reflects the engineering time required for design, integration, debugging, firmware development, backend setup, testing, and documentation.

4.1 Parts

Table 6 Parts Costs

Part	Manufacturer / Source	Retail Cost ($)	Actual Cost ($)
ESP32-S3 module / development board	Espressif	12	12
Analog microphone / audio front end	Generic / board BOM	7	7
ST7735 LCD display	Generic SPI LCD	15	15
TCRT5000 IR sensor modules	Generic	4	4
Motor drivers	Dual TC118S / equivalent	5	5
DC gear motors	Generic GA12-N20	16	16
Li-ion battery and protection	Generic	14	14
Voltage regulation and power parts	Generic	5	5
PCB, connectors, wiring, passive components	Various	54	54
Robot chassis, wheels, and enclosure material	Various	45	45
Total		177	177

4.2 Labor

The labor estimate follows the course-style engineering cost model used in the design document. Each team member is estimated at 120 hours. Using a nominal hourly rate of $20 and the standard 2.5 multiplier gives $6,000 per team member and $18,000 total labor cost for the three-person team.

The labor number is much larger than the bill of materials because the main project risk was integration rather than component price. Many low-cost parts, such as IR sensors or motor drivers, are easy to purchase but still require careful firmware, mechanical placement, threshold tuning, and safety validation. The same pattern appears in the audio system: the microphone and amplifier are inexpensive, but achieving reliable wake-word and ASR behavior requires power integrity, acoustic testing, and backend instrumentation.

Table 7 Labor Costs

Contributor	Hours	Rate and Multiplier	Cost ($)
Jiajun Gao	120	$20/hr x 2.5	6,000
Yuchen Shih	120	$20/hr x 2.5	6,000
Zichao Wang	120	$20/hr x 2.5	6,000
Total labor	360		18,000
Total project cost		Labor + parts	18,177

5. Conclusion

5.1 Accomplishments

This project successfully integrated a dual-PCB robot architecture, embedded audio, WebSocket-based audio streaming, a local AI voice pipeline, LLM-MCP orchestration, robot motion control, safety sensing, browser-based human-control mode, and an RTOS-based LVGL face into one working desktop robot. The final prototype met the main latency goal during recorded test sessions and demonstrated that MCP tools can expose physical robot actions to both direct intent handlers and the language model without making the robot feel slow or unresponsive.

The most important accomplishment is that Jarvis functions as a complete system rather than a set of separate demonstrations. A user can wake the robot, speak to it, hear a streamed response, see the face change state, and command physical motion through the same interaction loop. The robot can also reject unsafe motion commands and explain its state through MCP responses. This level of integration makes the project closer to a real embedded AI product, where conversation, physical behavior, safety, and feedback must work together in real time.

5.2 Uncertainties

The largest remaining uncertainty is audio robustness during battery-powered operation. The software stack is responsive, but the analog microphone path is sensitive to power-rail movement caused by motor activity, Wi-Fi transmission, and speaker output. The project also showed that latency optimization depends heavily on chunked audio delivery and persistent WebSocket connections, not only on choosing a faster language model. A second hardware revision should therefore treat microphone power integrity as a primary design constraint rather than a secondary issue.

Another uncertainty is how well the IR-based sensing approach generalizes across different desk surfaces. On the tested desk surface, IR sensing is fast, inexpensive, and simple to integrate. However, on darker, matte, reflective, or strongly sunlit surfaces, the same comparator threshold may not behave consistently. For this reason, the current robot should be understood as a successful prototype for the tested environment, while a more robust product version should consider ToF sensing or sensor fusion for improved reliability.

5.3 Ethical Considerations

Jarvis uses a local wake-word stage to avoid continuous audio transmission to the backend. Audio is streamed only after the wake word is detected, and the prototype does not intentionally store personal voice histories on the robot. However, because activated speech may still be processed by cloud or third-party language services, users should be informed when the robot is active and what services are involved. The system should not be used for sensitive decision-making, medical advice, or unsupervised safety-critical behavior.

The project also raises important transparency and autonomy concerns. Because the robot uses a voice interface and expressive face, users may perceive it as more socially aware or emotionally capable than it actually is. For that reason, Jarvis should not be presented as a mental-health treatment or as a replacement for human support. Its intended role is lightweight stress relief and simple interaction, not diagnosis or therapy. The safest user experience is one in which the robot’s capabilities, limitations, and data-processing behavior are clearly communicated.

5.4 Future Work

Future work should focus first on a second PCB revision with stronger power integrity, improved microphone isolation, and more reliable command recognition during battery-powered operation. The sensing roadmap includes replacing the current IR sensors with ToF sensors for more surface-independent cliff and obstacle detection. The audio roadmap includes adding a dedicated acoustic echo cancellation DSP to support full-duplex audio and more reliable barge-in. The perception roadmap includes integrating the OV5640 camera for VLM-based contextual awareness once the core voice and safety systems are stable. The mechanical roadmap includes evaluating a tracked chassis to improve traction and movement consistency.

A stronger second revision should also include a more formal validation suite. For voice performance, this would include repeated wake-word and short-command trials at different distances, battery levels, and noise conditions. For safety performance, it would include obstacle and cliff-detection trials across multiple desk materials and lighting conditions. For interaction quality, it would include user-facing tests to determine whether the face states are understandable without additional explanation. These tests would help turn the current successful prototype into a more defensible product-style design.

References

[1] Aalund, R., and Pecht, M., "The Use of UL 1642 Impact Testing for Li-ion Pouch Cells," IEEE Access, vol. 7, pp. 176706-176711, 2019.

[2] Lai, X., Yao, J., Jin, C., et al., "A Review of Lithium-Ion Battery Failure Hazards: Test Standards, Accident Analysis, and Safety Suggestions," Batteries, vol. 8, no. 248, 2022.

[3] Lakhnati, Y., Pascher, M., and Gerken, J., "Exploring a GPT-based large language model for variable autonomy in a VR-based human-robot teaming simulation," Frontiers in Robotics and AI, vol. 11, 2024.

[4] Nakamura, K., Nakadai, K., and Okuno, H. G., "A real-time super-resolution robot audition system that improves the robustness of simultaneous speech recognition," Advanced Robotics, vol. 27, pp. 933-945, 2013.

[5] Espressif Systems, "ESP32-S3 Series Datasheet" and ESP-IDF Programming Guide.

[6] "xiaozhi-esp32," GitHub repository, https://github.com/78/xiaozhi-esp32.

[7] Google, "Gemini API documentation."

[8] Silero Team, "Silero VAD."

[9] FunAudioLLM, "SenseVoice speech recognition model."

[10] IEEE, "IEEE Code of Ethics."

[11] UL Solutions, "UL 1642: Standard for Lithium Batteries."

Appendix A Requirement and Verification Table

This appendix keeps the detailed requirement-to-verification mapping out of the main narrative while preserving the final report's audit trail.

Table 8 System Requirements and Verifications

Requirement	Verification	Verification Status
HR1: Wi-Fi/cloud connection within 5 s.	Power on robot, join configured LAN/backend, and confirm WebSocket connection is established before interaction.	Yes
HR2: Voice response latency under 6 s.	Replay metrics report and inspect ASR-final-to-first-TTS latency. Confirm average latency is 2503 ms across 24 turns.	Yes
HR3: Lift and edge detection by bottom IR.	Place robot at desk edge or lift it; confirm bottom_floor becomes 0, cliff_detected state appears, and movement is blocked.	Yes
HR4: Front obstacle detection with at least 90 percent stop success.	Place object in front of the robot during motion tests and confirm forward-like commands are vetoed before collision.	Yes
HR5: Emergency stop within 50 ms of hazard detection.	Inspect high-priority IR/safety task timing and verify hazard flags override motor PWM outputs.	Yes
HR6: LCD expressions match all operating states.	Exercise listening, thinking, speaking, moving, music, idle, and error/warning paths and observe geometric face changes.	Yes
HR7: Stable battery operation for 30 minutes without reset.	Run battery-powered session and monitor for reset while also measuring ASR degradation caused by rail sag.	Yes