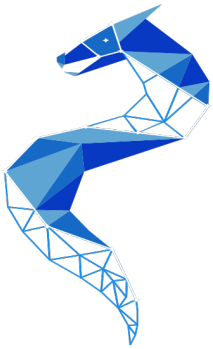


PwnBoxer

A network security device intended to provide small businesses and residential networks an automated auditing service to help rectify network vulnerabilities.

Group 22 Authors:

Franco Mezzarapa - Computer Engineering
Gugulethu Sigogo - Electrical Engineering
Abdiel Marcano - Computer Engineering
Robert Lee - Computer Engineering

Review Committee:

Dr. Mike Borowczak - ECE - Associate Professor
Dr. Nazanin Rahnavard - ECE - Professor
Dr. Justin Phelps - ECE - Adjunct Professor

Advisor:

Dr. Saleem Sahawneh - ECE - Lecturer

Contents

1 Executive Summary

The PwnBoxer system is a compact, network-connected vulnerability reporting platform designed to assist security researchers, embedded systems developers, and academic teams in monitoring the operational status of distributed hardware. The project integrates custom hardware, custom firmware, and a cloud-based backend into a unified ecosystem capable of securely transmitting device status, environmental readings, and network liveness information.

The hardware subsystem is built around a Raspberry Pi Compute Module 4 (CM4) paired with an ESP32-S3 microcontroller. The CM4 serves as the primary computational unit, running a hardened Linux environment configured for remote reporting, secure communication, and edge-level processing. The ESP32-S3 is responsible for sensor acquisition, Wi-Fi connectivity, heartbeat signaling, and supplemental data collection. Together, these components form a small yet robust platform capable of operating autonomously and communicating status updates to the cloud at regular intervals.

The cloud backend is deployed on an Azure Linux virtual machine and hosts a Node.js heartbeat API designed specifically for this project. The service exposes a protected endpoint that allows authenticated devices to send heartbeat packets. A dynamically generated status dashboard presents operational information such as the most recent heartbeat, device availability, and system health indicators. To meet the security demands of a publicly reachable endpoint, the backend incorporates several protective layers, including Fail2Ban to block brute-force SSH attempts, Apache's RequestTimeout module for Slowloris mitigation, and strict rate limiting inside the API to ensure that each device may send heartbeats only at controlled intervals. A firewall configuration using UFW restricts inbound traffic exclusively to required ports, and the system is managed with PM2 to ensure persistent uptime, monitoring, and automatic recovery.

Firmware running on the ESP32-S3 transmits heartbeat packets, environmental sensor data, and device identifiers over HTTP using a lightweight protocol designed for stability and recoverability. The CM4 runs background scripts responsible for additional reporting tasks, secure storage, and device identification. Together, these subsystems allow PwnBoxer nodes to operate as distributed monitoring agents that can report faults, outages, and network failures in real time.

The overall objective of the project is to create a small, reliable, inexpensive, and secure reporting platform that can be deployed in academic cybersecurity experiments, IoT demonstrations, and distributed systems research. The result is a fully integrated hardware–software solution capable of end-to-end communication, real-time monitoring, and fault reporting, backed by secure infrastructure and industry-standard protections. The PwnBoxer platform is modular, extensible, and designed to support future enhancements such as TLS-secured endpoints, automated alerting, and expanded device telemetry.

This report details the complete process of designing, assembling, programming, testing, and deploying the PwnBoxer system. It describes challenges such as embedded compatibility issues, network reliability constraints, secure service deployment, and hardware inte-

gration. It also documents the final implementation, performance validation, and security-hardening decisions that produced a stable and dependable system. The PwnBoxer platform ultimately fulfills its original design goals-providing a functional, secure, and scalable heartbeat reporting device suitable for real-world use cases and for future expansion by the development team.

2 Project Description

2.1 Introduction

The PwnBoxer is a hardware network auditing device intended for home and small businesses, allowing consumers to achieve affordable automated network auditing with feedback reporting for remediation of basic to intermediate network vulnerabilities. The PwnBoxer is intended to run through an automated workflow that audits the networks to which it is connected using readily available penetration testing tools to audit and generate a report. This tool also allows for technician connection into the network, making it a viable option for Internet Service Providers (ISPs) and Managed Service Providers (MSPs) to provide as a service, in which they can lend the device to do continuous penetration testing and, when needed, have a technician connect and continue the audit manually.

2.2 Motivation

With the rapid advancement of technology, including Artificial Intelligence (AI), new cyber threats and vulnerabilities are continually being discovered. This necessitates ongoing evaluations of organizational weaknesses at both the network and service levels. For small enterprises and residential users, however, conducting vulnerability assessments can be challenging due to financial and technical limitations. Traditionally, network audits require skilled professionals to identify the attack surface of the infrastructure and recommend appropriate mitigations. The proposed solution is designed not only for residential and small business users but also for ISPs and MSPs. By automating the auditing process and enabling technician intervention, ISPs can reduce the cost of conducting assessments while offering a value-added service to their customers. Furthermore, residential and small business users benefit from a low upfront cost for the device and an optional subscription for off-site processing.

2.3 Background

2.3.1 Network Security

Organizations commonly deliver various services to customers, whether deployed onsite, hosted in the cloud, or otherwise non-technological in nature. These services depend on infrastructure to maintain operation. Many enterprises rely on on-site services such as database systems and Active Directory. The software applications running on servers are referred to as services, and the overall security posture of a server is significantly influenced by the specific services running and their respective configurations.

A primary source of software vulnerabilities arises from substandard code quality, particularly when services are developed in languages that allow direct memory manipulation or when user-interface handling follows poor programming practices. Services that accept user input are susceptible to memory-based attacks such as buffer overflows without sanitazation of inputs. Despite operating systems including numerous protections [1], many remote code execution (RCE) vulnerabilities persist, especially when they can be triggered without user authentication or from user-accessible interfaces. Once an attacker compromises a vulnerable service hosted on a server, they may escalate privileges to domain-administrator levels and pivot laterally to compromise additional services within the network.

Another significant source of data breaches arises due to misconfigurations in network services, which allow adversaries to exploit insecure or unintended functionalities. Common examples include permitting unlimited authentication attempts without rate limiting, enabling brute-force or credential-stuffing attacks, and insecure file upload mechanisms that fail to validate input, potentially leading to arbitrary code execution or malware injection. Other frequent misconfigurations involve leaving default credentials enabled, exposing administrative interfaces to the public Internet, or failing to enforce proper access control policies. Misconfigurations consistently rank among the top causes of security incidents, as they create low-effort entry points for attackers compared to more complex exploitation [2]. Standardized configuration requirements, continuous monitoring, and adherence to security baselines are crucial for mitigating these risks.

Users and Employees are often easy targets for social engineering attacks due to publicly available information about them, weak authentication credentials, or habitual reuse of passwords exposed in prior breaches. Attackers exploit these weaknesses using techniques such as credential stuffing, leveraging leaked credential databases and wordlists such as rockyou.txt to automate logins into servers and services.

2.3.2 WiFi

Wireless networks, while offering flexibility and convenience, introduce a distinct set of vulnerabilities compared to wired infrastructures. A primary weakness lies in outdated or insecure encryption protocols such as WEP and WPA, which are still deployed in some environments despite being deprecated. Attackers can exploit these weak encryption schemes to intercept and decrypt network traffic using relatively inexpensive hardware and widely available software tools. Even WPA2, once considered robust, is susceptible to key reinstallation attacks (KRACK), enabling adversaries to replay packets, decrypt traffic, and potentially inject malicious data [3].

Rogue Access points and Evil Twin attacks are also of concern; adversaries can set up fake wireless networks that mimic legitimate SSIDs. Users who connect to these access points risk exposing credentials or sensitive traffic. Additionally, poor access control measures, such as guest networks without segmentation, can allow lateral movement into production networks. Preventative strategies include adopting WPA3 with modern encryption, deploying wireless intrusion detection systems, segmenting guest traffic, and enforcing strong

authentication mechanisms such as 802.1X.

2.3.3 Penetration Testing

Penetration testing is commonly employed to identify vulnerabilities within a target system by an individual, team, or external firm. However, traditional penetration tests are costly and must be repeated periodically to account for architectural changes and evolving security threats. This is addressed through Penetration Testing as a Service (PTaaS), which offers a subscription-based model that combines automated testing with human intervention. PTaaS platforms often integrate directly into development pipelines, enabling continuous vulnerability assessments and faster remediation feedback. Many services provide dashboards for vulnerability tracking and compliance reporting, making the remediation process more actionable for the customers.

2.4 Goals

The goals defined here are broad and represent the overall outlook and interpretation of our project. Basic goals constitute the fundamental ideas for what tasks PwnBoxer is intended to perform. Advanced goals are improved functionalities that we intend to make for PwnBoxer in this design, and stretch goals are functionalities for PwnBoxer which would be beneficial to the project, but are beyond the scope of our implementation.

2.4.1 Basic Goals

- Create a compact and portable system capable of auditing networks and detecting vulnerabilities through an automated pipeline.
- Generate comprehensive reports that summarize the overall security posture and identified vulnerabilities.
- Collect input from peripherals connected to the MCU and transfer the data to the compute module for processing.
- Perform network security computations on the compute module and communicate status back to the MCU.
- Provide system condition feedback through integrated LED indicators and display.

2.4.2 Advanced Goals

- Support customizable audit durations based on user requirements.
- Enable simultaneous auditing of multiple networks.
- Allow technicians and advanced users to integrate custom vulnerability tests.

2.4.3 Stretch Goals

- Implement an animated status bar to display progress during network audits.
- Integrate user reports with an online AI provider.
- Develop a web-based security dashboard that displays test results, trends, and recommendations for the end user.

2.5 Objectives

The objectives defined here are specific and represent technical and measurable deliverables that will be applied to the project. Basic objectives denote deliverables that are essential requirements for successful system operation. Advanced objectives are aspects in which the system will be significantly improved upon implementation, and stretch objectives are ideal specifications that may be implemented if resources permit.

2.5.1 Basic Objectives

- Execute industry-standard security tools such as Nmap and Nessus to audit small business or residential networks.
- Generate vulnerability assessment reports using a large language model (LLM).
- Provide a touchscreen interface for collecting network information and email addresses as user input and displaying audit progress updates.
- Utilize LEDs to convey system conditions such as network audit progress and device health.
- Transmit generated reports to the user via email, with the email address collected prior to the audit.

2.5.2 Advanced Objectives

- Enable remote technician or user support via a secure reverse shell connection to the compute module.
- Implement UART communication between the compute module and MCU to display pipeline stage information to the end user.
- Support concurrent auditing of distinct networks (e.g., Ethernet and Wi-Fi) when permitted by the user.

2.5.3 Stretch Objectives

- Add API-based integration to support custom vulnerability tests defined by the technician or end user.

- Integrate an onboard audio alert system to provide audible notifications for key operational events such as boot status, scan completion, or detected vulnerabilities.
- Host a secure, authenticated web dashboard using Apache to present historical performance metrics and comparative results.

2.6 Existing Products

2.6.1 Bonelli Systems Automated Penetration Testing

Bonelli Systems offers an automated penetration testing platform that provides real-time network vulnerability assessments and actionable reporting [4]. It is a cloud-based service suitable for small to medium-sized businesses that require continuous security testing. Like PwnBoxer, Bonelli Systems provides automated vulnerability assessments and delivers reports to help users remediate security issues. The differences lie in the deployment and delivery model: Bonelli Systems operates entirely as a cloud service requiring ongoing subscriptions, whereas PwnBoxer is a portable hardware device with optional on-site technician support. Additionally, PwnBoxer is designed for periodic assessments and usability by non-expert users, whereas Bonelli focuses on continuous automated testing.

2.6.2 Kali Linux on Raspberry Pi

Kali Linux is a widely adopted penetration testing distribution that can be deployed on lightweight single-board computers, such as the Raspberry Pi [5]. This configuration transforms an inexpensive device into a portable penetration testing platform with hundreds of pre-installed tools covering web, network, wireless, and cryptographic security assessments. Operation is highly flexible but requires manual setup of the Raspberry Pi, installation of Kali Linux, and ongoing system administration by the user. User interaction is almost entirely command-line and GUI-driven, depending on the selected tools, making it best suited for technically proficient security professionals, researchers, and enthusiasts. While this approach shares PwnBoxer's small hardware form factor and portability, it lacks PwnBoxer's built-in automation, integrated reporting, and technician support, placing the burden of customization and tool management on the user.

2.6.3 LAN Turtle by Hak5

The Hak5 LAN Turtle is a covert USB Ethernet adapter designed for stealthy penetration testing and remote access [6]. Once connected to a target network via USB, it operates by establishing a hidden communication channel that allows penetration testers or red teamers to remotely access internal systems. Its capabilities include network sniffing, man-in-the-middle attacks, credential harvesting, and persistent remote shells, making it a versatile and powerful tool for covert operations. Interaction with the LAN Turtle requires command-line configuration and scripting, which assumes a high level of technical expertise from the user. The intended audience primarily consists of professional penetration testers and red team operators who value stealth and persistent access. Like PwnBoxer, it is hardware-based and portable; however, its design philosophy emphasizes covert deployment and

deep customization rather than user-friendly automation and reporting.

2.6.4 MiniPwner by Hacker Warehouse

MiniPwner is a compact penetration testing device built on OpenWrt, typically utilizing a TP-Link MR3040 router and configured to run tools such as Metasploit and Aircrack-ng. It is battery-powered, enabling portable network assessments and providing flexibility for penetration testers to deploy in a variety of environments [7]. Similar to the PwnBoxer, MiniPwner is hardware-based and allows for automated vulnerability testing, including assessments of wireless networks. However, MiniPwner is primarily a DIY device that requires technical expertise to configure and lacks integrated reporting or technician support features. Its focus is more on stealth and portability rather than on providing a fully automated workflow for users with limited technical knowledge.

2.6.5 Penetrator by SecPoint

SecPoint's Penetrator is a comprehensive vulnerability scanning and penetration testing appliance available as a physical device or virtual machine. It performs automated scanning across internal and external networks, identifying vulnerabilities such as SQL injection, cross-site scripting, and misconfigurations. It also generates detailed reports with remediation suggestions [8]. Like PwnBoxer, Penetrator automates network assessments and produces actionable reports for remediation. However, Penetrator is primarily designed for enterprise environments, is less portable than PwnBoxer, and offers a wider range of scanning profiles that may require technical expertise to operate. PwnBoxer, by contrast, balances automation with ease of use for small businesses and residential users and integrates optional technician intervention and subscription-based off-site processing.

2.6.6 Pwn Plug by Pwnie Express

The Pwn Plug, developed by Pwnie Express, is a penetration testing device disguised as a power adapter that plugs into an Ethernet port to provide remote access to networks. It is widely used by professional penetration testers for enterprise-level assessments and allows the execution of various penetration testing tools [9]. Similar to the PwnBoxer, it provides hardware-based access for network vulnerability assessments and supports a range of penetration testing tools. The key differences are that the Pwn Plug emphasizes stealth and covert deployment, is designed for professional security experts, and does not provide user-friendly reporting or optional technician intervention like PwnBoxer.

2.6.7 WiFi Pineapple by Hak5

The WiFi Pineapple is a specialized wireless auditing platform designed for penetration testing of WiFi networks [10]. It streamlines and automates many common wireless attacks, including rogue access point creation, credential harvesting, client deauthentication, and traffic interception. The device provides a web-based interface for configuration, making it more user-friendly than command-line-only tools while still allowing advanced customization through downloadable modules. Its primary audience is penetration testers, red

teamers, and IT security professionals focused on wireless security assessments. Compared to PwnBoxer, the WiFi Pineapple automates significant parts of wireless exploitation but is limited in scope to WiFi networks. PwnBoxer, by contrast, supports broader wired and wireless network auditing with integrated reporting and optional technician assistance, making it more accessible to non-expert users while serving a wider range of use cases.

2.7 Required Specifications

2.7.1 Overall System Specifications

Report Accuracy	70%
Maximum Power Consumption	15W
Onboard Data Storage	32GB + 4MB
Onboard Memory Size	8GB + 512KB
Weight	2 lbs
Number of Concurrent Programs	2
Display	3.5” TFT with Touch
AI Model	Gemma3n

2.7.2 Component Specifications

2.7.3 Raspberry Pi Compute Module 4 Specifications

GPIO Pins	28 Pins
Standard Clock Frequency	1.5 GHz
Supply Voltage	5V
Onboard Memory	8GB
Onboard Storage	32GB
Communication Standards	UART, I2C, SPI, Ethernet, PCIe, USB 2.0, HDMI, Wi-Fi

2.7.4 ESP32 Specifications

GPIO Pins	45 Pins
Maximum Clock Frequency	240MHz
Supply Voltage	3.3V
Onboard Memory	512KB SRAM
Communication Standards	UART, I2C, SPI, Wi-Fi, Bluetooth

2.8 Hardware Block Diagram

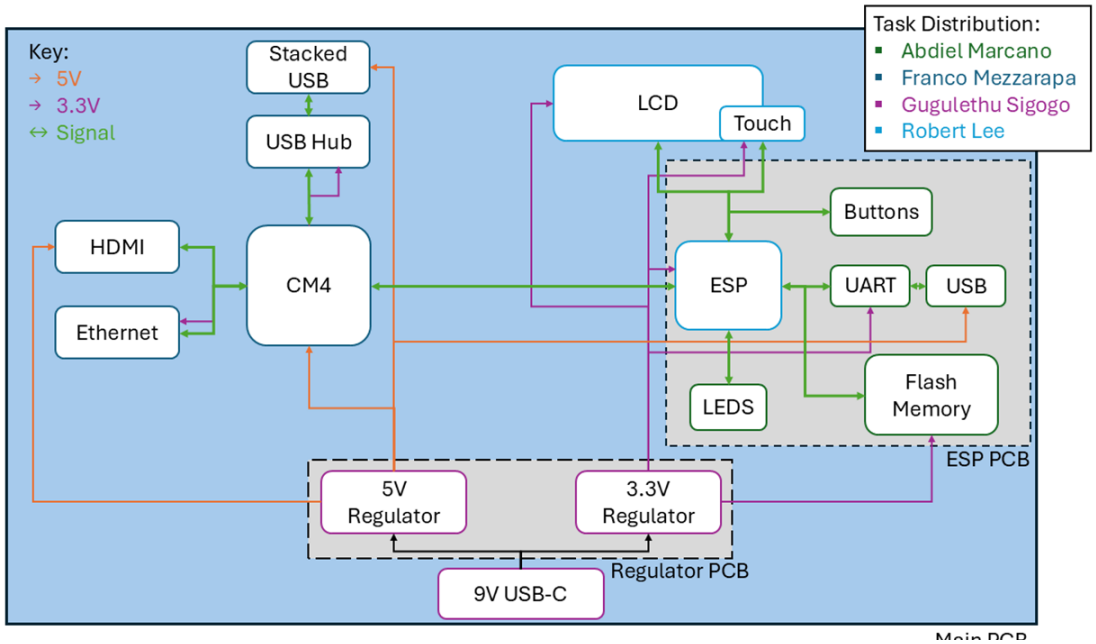


Figure 1: Hardware Block Diagram by Authors

Hardware block diagram of the PwnBoxer hardware network auditing system, illustrating component interconnections, power regulation, and data flow, including the CM4 module, ESP controller, regulators, peripherals, and communication interfaces.

2.9 Software Flowchart

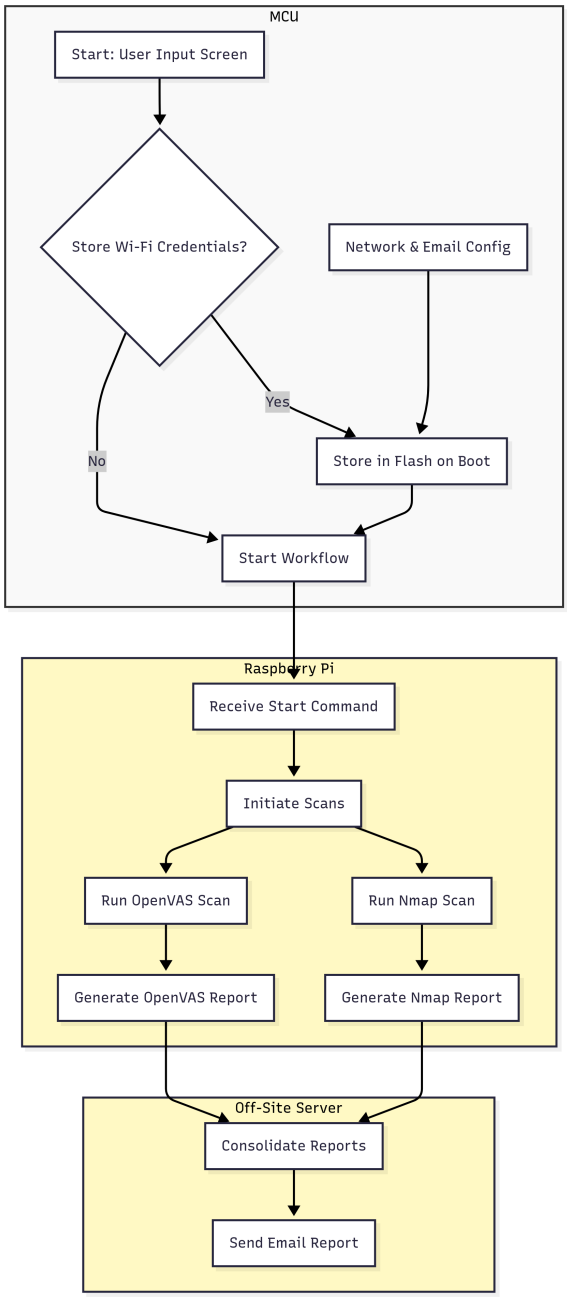
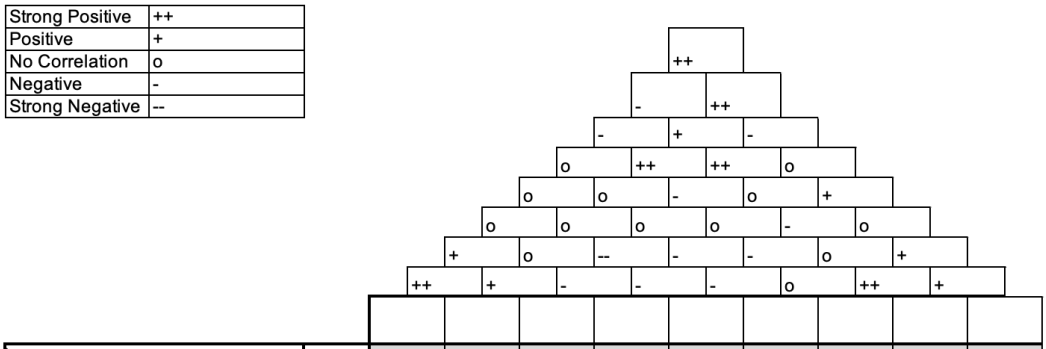


Figure 2: Software Flowchart

The Software Flowchart outlines the process the system follows for automated vulnerability assessment, including user configuration via the MCU, parallel scan execution using Nmap and Nessus on the Raspberry Pi, and a final reporting mechanism handled by an off-site server.

2.10 House of Quality

Strong Positive	++
Positive	+
No Correlation	o
Negative	-
Strong Negative	--



Design Requirements / Customer Requirements		Size	Weight	Battery Life	Task Completion Time	Power Consumption	Report Complexity	Manufacturability	Maintainability	Cost
		-	-	+	-	-	-	+	+	-
Ease of Use	+	++	++	o	++	o	o	o	o	o
Portability	+	++	++	++	o	++	o	o	+	+
Audit Accuracy	+	o	o	o	-	o	--	o	o	o
Customizability	+	-	o	o	o	o	-	-	-	-
Longevity	+	o	o	+	o	++	o	+	++	o
Retail Cost	-	+	+	-	o	-	o	++	++	+
Ease of Report Dissemination	+	o	o	o	+	o	++	o	o	o

Figure 3: House of Quality by Authors

The House of Quality provides a visual aid representing engineering tradeoffs that may be made during design and production. The column on the top represents the engineering specifications that can be altered while under consideration. The column to the left represents the specifications that are most important to the customer. The matrices above and between the columns signify the correlations among each of the items individually.

3 Research and Investigation

Unless otherwise stated, all software selections are constrained by:

- **On-device RAM = 8 GB**
- **On-device storage = 32 GB**
- **Maximum audit completion time = 4 hours**

These constraints drive the containerization strategy, scan profiles, and report pipeline described below.

3.1 Technologies

This section presents the hardware portion of our research and investigation. For each component, we follow the required structure: (i) *technology comparison and selection* (table with $\geq$3 options), followed immediately by (ii) *part/product comparison and selection* (table with $\geq$3 candidates). The hardware discussion is divided into four main sections: the microcontroller (MCU) subsystem and its supporting hardware, the single-board computer (SBC) subsystem and its associated components, the display subsystem, and the voltage regulation subsystem.

3.1.1 MCU vs. FPGA vs. SBC

Due to the networking and embedded systems requirements for our project, there will be a choice between using a microcontroller, field-programmable gate array, single-board computer, or some combination of the three. The differences in these technologies create the necessity for a thorough comparison, to ensure the correct choice is made for the implementation of our project. For example, FPGAs are incredibly powerful and flexible in comparison to an MCU but typically come at a much higher cost per unit and consume much more power. To select the correct technology to implement PwnBoxer, it is essential to determine which platform best meets the requirements of the system. PwnBoxer is a networking device, which means it must have a means of connecting to a network through Ethernet or Wi-Fi. Additionally, the system must have a display and set of peripherals for a developer to control the system from, which make use of communication protocols like SPI, I2C, and UART.

MCUs have many features directly built in that fulfil the peripheral and communication needs for PwnBoxer. Microcontroller development boards often have immense amounts of documentation, either officially from the companies who developed them or from users who have built projects using them, which cover every feature that the MCU could be used for. The design of MCUs include the processor, memory, and peripheral capability within a single chip that is very energy efficient, and very cost effective when compared to FPGAs and SBCs. Using common programming languages like C, C++, and Python, MCUs are easy to program. Code libraries created by the developing companies and users of the boards allow for the integration of many different input and output peripherals with simple programming instructions. Additionally, being able to program the MCU in a common language allows the use of debugging tools that programmers are familiar with, which improves ease of use.

FPGAs are highly customizable and powerful, allowing the user to design and replicate any design down to the gate-level. Their uses in parallel processing applications cannot be matched using an MCU or most SBCs. With this level of processing power brings increased power consumption and higher cost. Additionally, FPGAs are programmed using hardware description languages (HDLs) rather than more common desktop programming languages, apart from the use of high-level synthesis (HLS) to convert a language like C into synthesizable HDL code. The ease of use in comparison to MCUs is significantly lower, as the learning curve for creating and implementing designs that contain peripherals on FPGAs

requires a much greater level of sophistication. Similarly, where MCUs have components like analog-to-digital converters (ADCs) and communication protocols integrated already, external components need to be added into the design when implementing these things on an FPGA. Documentation for FPGAs when compared to that of MCUs is smaller and more difficult to understand. There is far less user-created documentation for FPGAs, and the documentation that exists can be difficult to understand as it is mostly aimed at skilled engineers who are familiar with complex topics in the use of FPGAs.

SBCs provide a solid middle ground between MCUs and FPGAs based on cost and ease of use. Included with these development boards are a full operating system environment in a compact form-factor, with input and output peripheral pins capable of handling communication protocols like SPI, I2C and UART. For the networking-related tasks that PwnBoxer must complete, SBCs offer features that allow for easy implementation. Because they run Linux-based operating systems, SBCs support standard networking tools, scripting languages, and server software, enabling users to configure routers, firewalls, network monitors, or IoT gateways with relative ease. Additionally, SBCs consume little power and can operate continuously, which is valuable for long-term testing or deployment. Their strong community support and extensive online documentation also make troubleshooting and project scaling much more accessible.

Why both MCUs and SBCs?
For the implementation of PwnBoxer, a combination of an MCU and an SBC will provide the best platform to carry out the necessary tasks outlined for this project. Including both an MCU and an SBC allows the system to take advantage of the strengths of each device, creating a more powerful, flexible and efficient design. The MCU excels at real-time control, handling tasks that require precise timing and direct interaction with peripherals. It can process signals quickly, consume very little power, and operate reliably even in resource-constrained environments. Meanwhile, the SBC provides the high-level processing power, memory, and operating system support needed for networking, data management, and communication protocols. By having the MCU handle hardware-level operations and the SBC manage complex computations, data storage, and network communication, the project can achieve a clear division of labor that optimizes both performance and efficiency. Overall, combining an MCU and an SBC enables the project to bridge the gap between embedded control and high-level networking, resulting in a smarter, more responsive, and interconnected system suitable for Internet of Things (IoT) or distributed automation applications.

3.1.2 MCU Subsystem

Technology comparison and selection
The microcontroller selected for our system is a variant from the Espressif Systems ESP32 family, the ESP32-S3. To reach this selection, we compared several microcontroller technologies to determine which offered the processing power and set of features that would meet the requirements of the PwnBoxer system.

ESP32-WROOM-32

The ESP32-WROOM-32 is the most widely adopted variant of the ESP32 family of microcontrollers among embedded developers and students. It is built around a dual-core Xtensa LX6 processor running at up to 260MHz, and includes support for 2.4GHz Wi-Fi and Bluetooth 4.2 and Bluetooth Low Energy (BLE). With a cost of $10 USD for the full development board, developers can use this board to create a plethora of IoT devices and embedded systems applications. The module integrates 520 KB of SRAM, 448 KB ROM, and typically 4 MB of SPI flash, though configurations with up to 16 MB are available. Multiple sleep modes for low-power operation are included, with current draw dropping to the microamp range in deep sleep.

The WROOM-32 variant provides a rich set of communication protocols and peripheral standards including, SPI I^2C, UART, PWM, and ADC/DAC. 26 GPIO pins give the module the capability to support peripherals like sensors, motors, and displays.

From a security standpoint, the ESP32-WROOM-32 includes hardware encryption standards like Advanced Encryption Standard (AES), Secure Hash Algorithms (SHA), the Rivest Shamir Adleman (RSA) cryptosystem, and Elliptic Curve Cryptography (ECC). Additionally, secure boot and flash encryption ensures data integrity and secure communication. It is fully supported by Espressif's ESP-IDF SDK, Arduino IDE, and popular frameworks like MicroPython and PlatformIO, offering developers a broad ecosystem of tools. The WROOM-32's dual-core architecture allows for heavy computation to be split into tasks among the two cores, improving efficiency and computation time.

With the ESP32-WROOM-32 being designed as a general-purpose embedded development powerhouse, it can be used in many applications, including the PwnBoxer system. However, The number of GPIO pins and processing power are improved upon significantly in other variations of the ESP32 family, which are described below [11].

ESP32-S2

The ESP32-S2 is built around a single-core 240 MHz Xtensa LX7 processor, which offers a strong balance of performance and power efficiency. Within the module includes 320 KB or SRAM and up to 128 KB of ROM, but no internal SPI flash memory. However, support for external SPI flash and PSRAM is available for more demanding applications. A feature that the S2 holds over the WROOM-32 variant is support for USB On-the-Go (OTG), which allows the S2 to act as a USB host, enabling it to connect to and control USB devices like a keyboard, mouse, or external storage device without needing a separate computer. The S2 also integrates capacitive touch sensing across up to 43 GPIO pins, temperature sensors and advanced ADCs for precision analog input.

Similarly to the WROOM-32 variant, the S2 includes a list of security features like secure boot, flash encryption, and hardware encryption. The Wi-Fi subsystem supports 2.4GHz IEEE 802.11 b/g/n, with Wi-Fi Protected Access 3 (WPA3) encryption. To simplify the RF design and save power consumption, the S2 does not include Bluetooth connectivity. This microcontroller is a strong contender for use in the PwnBoxer system, but is improved upon significantly in the S3 variant of the ESP32 family [12].

ESP32-S3
The ESP32-S3 takes the foundation of the S2 variant and improves significantly upon it. It features a dual-core Xtensa LX7 processor running at up to 240MHz to improve processing speed and parallelization of tasks. Also, both Wi-Fi 2.4 GHz and Bluetooth 5.0 with BLE are included inside the module, with long-range and mesh capabilities. The S3 introduces vector instructions for AI acceleration, significantly improving efficiency in neural network inference, signal processing, and other DSP tasks. Included is 512 KB of SRAM, and has external support for up to 16 MB of PSRAM. With additional USB OTG, LCD interfaces, SPI, I^2C, UART, PWM and ADC controllers makes it versatile for the tasks it will be given within the PwnBoxer system.

The S3 has two additional GPIO pins over the S2 variant, with a total of 45 GPIO pins available. This is enough to support peripherals like LCD displays with capacitive touch, which are discussed in a later section. The development board for the ESP32-S3 is an additional $5 over the ESP32-WROOM-32, for a total of $15. This is well within the budget for the project, and its increased cost does not outweigh the benefit of the improved performance. The increase in processing power combined with the increased number of GPIO pins make the ESP32-S3 stand out as the choice for use in PwnBoxer [13].

ESP32-C3
The ESP32-C3 aims to be a cost-effective player in the open-source MCU space, with a single core 32-bit RISC-V processor operating at 160 MHz. Paired with the RISC-V core are the connectivity capabilities of 2.4 GHz Wi-Fi and Bluetooth 5 with low-energy and long-range functionality. The unit includes 400 KB SRAM 384 KB ROM and external SPI flash support. While Wi-Fi and Bluetooth are included in the package for the C3, support for IEEE 802.15.4 connectivity with Thread and Zigbee are not included [14].

ESP32-C6
Building upon the ESP32-C3, the C6 variant retains the same single core RISC-V processor running at 160 MHz, but includes greater connectivity features which were missing in the C3. Most notably, IEEE 802.15.4 connectivity with support for Thread and Zigbee protocols are supported on the C6. Additionally, the ESP32-C6 is the first MCU from Espressif Systems to support Wi-Fi 6, increasing throughput and power efficiency while creating a better network coexistence. The C6 can be configured to have up to 31 GPIO pins, with support for peripherals and communication protocols like SPI, I^2C, UART, ADC and PWM [15].

ESP32-H2
The ESP32-H2 is a significant departure from the other variants, as it is based around a RISC-V processor and is designed specifically for energy efficient wireless communication. The module omits Wi-Fi capability to focus solely on Bluetooth 5 LE and the IEEE 802.15.4 standard for low-power, short-range, wireless networks. The H2 variant puts the Zigbee and Thread communication protocols at the forefront of the design, which run on top of 802.15.4. The RISC-V processor inside runs at up to 96 MHz with 320 KB of SRAM

and has support for external flash memory. The maximum frequency of this RISC-V processor is significantly less than the other members of the ESP32 family, but it is enough power to handle mesh networking and sensor data aggregation.

Since wireless communication is the main purpose for the design of the H2, security is also a priority for its architecture. Secure boot, flash encryption, digital signatures, and hardware cryptographic acceleration for AES, SHA-2, RSA, and ECC algorithms are all included in the packaging of the module. The chip's low active and sleep currents enable multi-year operation on coin-cell batteries. Peripherals include SPI, I^2C, UART, ADC, and GPIO interfaces, enabling flexible integration with sensors and actuators. While having a RISC-V based MCU would make PwnBoxer somewhat unique, the processing speeds this variant provides would not be tolerated for the requirements of the system [16].

Table 3.1: Comparison of ESP32-WROOM-32, ESP32-S2, and ESP32-S3 by Authors.

Feature	**ESP32-WROOM-32**	**ESP32-S2**	**ESP32-S3**
GPIO	26 Pins	43 Pins	45 Pins
UART	Yes	Yes	Yes
I^2C	Yes	Yes	Yes
SPI	Yes	Yes	Yes
PWM	Yes	Yes	Yes
ADC	Yes	Yes (12-bit)	Yes (12-bit)
DAC	Yes (2-ch 8-bit)	No	No
Wi-Fi	Yes 2.4 GHz	Yes 2.4 GHz	Yes 2.4 GHz
Bluetooth	Yes v4.2	No	Yes v5.0 LE
Zigbee / Thread	No	No	No
6LoWPAN	No	No	No
Flash Memory	4 MB (ext.)	Up to 4 MB (ext.)	Up to 16 MB (ext.)
RAM	520 KB SRAM	320 KB SRAM	512 KB SRAM
Operating Voltage	3.0–3.6 V	3.0–3.6 V	3.0–3.6 V
Typical Current Draw	160 mA	80 mA	80–100 mA
Power Consumption	0.53 W	0.26 W	0.33 W
Operating Temp.	40 °C to +125 °C	40 °C to +105 °C	40 °C to +85 °C
CPU Speed	240 MHz	240 MHz	240 MHz
CPU Cores	2 (Xtensa LX6)	1 (Xtensa LX7)	2 (Xtensa LX7)
WPA3 Encryption	Yes	Yes	Yes

Table 3.2: Comparison of ESP32-C3, ESP32-C6, and ESP32-H2 by Authors.

Feature	ESP32-C3	ESP32-C6	ESP32-H2
GPIO	22 Pins	31 Pins	28 Pins
UART	Yes	Yes	Yes
I^2C	Yes	Yes	Yes
SPI	Yes	Yes	Yes
PWM	Yes	Yes	Yes
ADC	Yes (12-bit)	Yes (12-bit)	Yes
DAC	No	No	No
Wi-Fi	Yes 2.4 GHz	Yes Wi-Fi 6 (2.4 GHz)	No
Bluetooth	Yes v5.0 LE	Yes v5.0 LE	Yes v5.0 LE
Zigbee / Thread	No	Yes (802.15.4)	Yes (802.15.4)
6LoWPAN	No	Yes	Yes
Flash Memory	Up to 4 MB (ext.)	Up to 4 MB (ext.)	Up to 4 MB (ext.)
RAM	400 KB SRAM	512 KB SRAM	320 KB SRAM
Operating Voltage	3.0–3.6 V	3.0–3.6 V	3.0–3.6 V
Typical Current Draw	75 mA	90 mA	70 mA
Power Consumption	0.25 W	0.30 W	0.23 W
Operating Temp.	40 °C to +105 °C	40 °C to +85 °C	40 °C to +85 °C
CPU Speed	160 MHz	160 MHz	96 MHz
CPU Cores	1 (RISC-V)	1 (RISC-V)	1 (RISC-V)
WPA3 Encryption	Yes	Yes	Yes

MSP430FR6989

The MSP430FR6989 is a microcontroller built by Texas Instruments which is designed to be used for very low power embedded applications. It is centered around a single 16-bit RISC core running at up to 16 MHz, and can operate at voltages as low as 1.8 V. A major difference when compared to other microcontrollers is the presence of ferroelectric RAM (FRAM) rather than SRAM. FRAM technology allows for ultra low-power writes that have a similar write speed to SRAM, and similar endurance metrics to flash memory.

For peripherals, the MSP430FR6989 offers a rich set of analog and digital features: a 12-bit ADC with up to 16 external input channels, 16-channel analog comparator, Extended Scan Interface (ESI) suited for measurement tasks (water, heat, gas volume), five 16-bit timers (each up to 7 capture/compare registers), a 32-bit hardware multiplier, three-channel Direct Memory Access (DMA), touch-capable I/O on ports P1-P10 and PJ, as well as AES-128/256 encryption and a true random number generator. Depending on the packaging for the MCU, the MSP430FR6989 can be had with up to 83 GPIO pins.

The MSP430 family of microcontrollers is programmed using the C programming language

with libraries created by Texas Instruments and third-party developers. Texas Instruments created an integrated development environment for their microcontrollers that allows for easy programming and design implementation called Code Composer Studio. As embedded programming on the MSP430FR6989 using Code Composer Studio is often taught in the curriculum of related engineering degrees, the ease of use for implementation is significantly higher than other microcontroller families [17].

STM32F103C8

The STM32F103C8 is a microcontroller built by STMicroelectronics that features a 32-bit Arm Cortex-M3 core running at up to 72 MHz. It is equipped with up to 128 KB of flash memroy, 20 KB SRAM and supports a wide operating voltage form 2.0 V to 3.6 V. Looking at peripherals, the STM32 supports SPI, I^2C, UART, 2 12-bit ADCs, a 7-channel DMA controller, and 37 GPIO pins.

The STM32 family of microcontrollers has a steep learning curve for beginners, as the programming process is geared toward professionals in the embedded systems space. STMicroelectronics provides an integrated development environment called STM32CubeIDE, where the microcontrollers can be programmed using C or C++. When compared to an environment like Arduino IDE, STM32CubeIDE requires a more explicit, low-level understanding of the microcontroller's inner workings. This reduces the ease of use during software development significantly, which makes the STM32F103C8 a difficult choice for implementation [18].

Arduino UNO

The Arduino UNO is a development board from Arduino built around the ATmega328P 8-bit AVR microcontroller, running at up to 16 MHz. It provides 32 kB of Flash memory for code, 2 kB of SRAM, and 1 kB of Electrically Erasable Programmable Read-Only Memory (EEPROM). The board exposes 14 digital IO pins (6 of which support PWM), 6 analog input pins, USB connectivity, a power jack, ICSP header, and a reset button. The board supports power supply via USB or external 7–12 V and logic levels at 5 V.

The UNO is designed as Arduino's beginner friendly and widely adopted platform for prototyping and education, which brings with it a plethora of documentation from both Arduino and independent developers. Using Arduino's own development environment, Arduino IDE, the boards can be easily programmed in what are called sketches. Sketches are a variant of C++ with Arduino-specific functions and libraries. The use of sketches makes programming the UNO simple in comparison to using something like the STM32 [19].

Arduino Nano

The Arduino Nano is a compact, breadboard-friendly development board also built on an ATmega328P 8-bit AVR microcontroller running at 16 MHz. It offers 32 kB Flash, 2 kB SRAM, 1 kB EEPROM, and supports up to 22 digital IO pins, 8 analog input pins, and 6 PWM outputs. The board can be powered via mini-USB or via VIN (7-15 V) or regulated 5 V supply, making it suitable for embedded breadboard prototyping. The main difference

the Nano has over the UNO is the form factor, with the Nano being significantly smaller and having a reduced GPIO set. The compatibility with the Arduino IDE and set of libraries makes the Nano a strong choice from an ease of use perspective, but it lacks sorely in processing power, operating voltage requirements, and GPIO availability [20].

Table 3.3: Comparison of MSP430FR6989, STM32F103C8, Arduino UNO, and Arduino Nano by Authors.

Feature	MSP430FR6989	STM32F103C8	Arduino UNO	Arduino Nano
GPIO Pins	83	37	20	30
UART	Yes	Yes	Yes	Yes
I²C	Yes	Yes	Yes	Yes
SPI	Yes	Yes	Yes	Yes
PWM Channels	14	12	6	6
ADC Channels	Yes (12-bit)	Yes (12-bit)	Yes (10-bit)	Yes (10-bit)
DAC	No	No	No	No
Wi-Fi	No	No	No	No
Bluetooth	No	No	No	No
Flash Memory	128 KB FRAM	64–128 KB Flash	32 KB Flash	32 KB Flash
RAM	2 KB	20 KB	2 KB	2 KB
Operating Voltage	1.8–3.6 V	2.0–3.6 V	5 V (7–12 V input)	5 V (7–15 V input)
Typical Current Draw	100 μA active	36 mA active	15–20 mA active	15–20 mA active
Power Consumption	$\sim$0.4 μW standby	$\sim$120 mW active	$\sim$100 mW typical	$\sim$100 mW typical
Operating Temperature	–40°C to +85°C	–40°C to +85°C	0°C to +70°C	0°C to +70°C
CPU Speed	16 MHz	72 MHz	16 MHz	16 MHz
CPU Cores	1 (16-bit RISC)	1 (32-bit ARM Cortex-M3)	1 (8-bit AVR)	1 (8-bit AVR)

Selection

For the implementation of PwnBoxer, the ESP32-S3 will be used as it presents the best balance of features, available GPIO pins, peripheral support, power consumption, cost, and processing speed. Having 45 available GPIO pins allows the system to include features like a display in either 8-bit or 16-bit parallel with pins left over to communicate with the other subsystems. The operating voltage is a low 3.3 V, which is easily met by the use of our intended voltage regulation subsystem, and contributes to a very low power consumption metric of only 0.33 W. These specifications combined with the increased processing power of the two Xtensa LX7 cores make the ESP32-S3 the strongest choice for the PwnBoxer

system.

3.1.3 SBC Subsystem

Technology comparison and selection.
The main processing unit of our system is based on the Raspberry Pi Compute Module family. Before finalizing our selection, we compared several compute module technologies to determine which offered the best balance between processing power, expandability, and integration capability. The comparison focused on factors such as CPU performance, memory capacity, available I/O interfaces, power consumption, and cost.

BeagleBone AI-64
The BeagleBone AI-64 integrates the Texas Instruments AM5729 processor, featuring dual-core ARM Cortex-A15 CPUs alongside embedded C66x DSP cores and dual embedded vision engines. This heterogeneous architecture allows the board to handle both high-level application tasks and deterministic real-time processing with exceptional efficiency. Its Programmable Real-Time Unit (PRU) subsystem further enhances low-latency control, making it well-suited for robotics, motor control, and industrial automation. The inclusion of onboard Wi-Fi, Bluetooth, Gigabit Ethernet, and USB-C connectivity provides broad interface flexibility for complex embedded networks. Supported by an open-source software ecosystem and a robust Linux environment, the BeagleBone AI-64 appeals to developers requiring fine-grained hardware control and high computational performance. However, its steep learning curve and configuration complexity often position it as a platform for advanced developers and research-oriented applications rather than entry-level prototyping. [21].

Raspberry Pi Compute Module 4 (CM4)
The Raspberry Pi Compute Module 4 (CM4) is based on the Broadcom BCM2711 system-on-chip (SoC), which incorporates a quad-core ARM Cortex-A72 processor clocked at 1.5 GHz. It supports up to 8 GB of LPDDR4 memory and optional eMMC storage capacities of up to 32 GB, providing a balance between computational performance and energy efficiency. Designed explicitly for embedded and industrial environments, the CM4 utilizes dual 100-pin high-density connectors that expose a wide range of interfaces, including PCIe, USB 2.0 and 3.0, HDMI, and multiple GPIO, I^2C, and SPI buses. This modular approach allows integrators to tailor carrier boards to their specific application requirements. The small form factor, mature Linux support, and extensive open-source community make the CM4 an attractive platform for Internet of Things devices, automation systems, and network-oriented embedded applications. Its combination of affordability, flexibility, and software ecosystem continuity further reinforces its role as one of the most accessible and reliable SBC solutions for modern embedded design [22].

Raspberry Pi Compute Module 5 (CM5)
The Raspberry Pi Compute Module 5 (CM5) represents a significant architectural advancement over the CM4, leveraging the Broadcom BCM2712 SoC with a quad-core ARM

Cortex-A76 processor and an upgraded VideoCore VII GPU. These enhancements deliver substantially improved processing throughput, higher memory bandwidth, and advanced multimedia capabilities suitable for AI-driven and high-performance embedded systems. The CM5 adds native support for PCIe Gen 3, enabling integration with high-speed peripherals such as NVMe storage or AI accelerators. While the module introduces a new form factor and connector interface, increasing design flexibility, it also limits backward compatibility with previous carrier boards. Its improved energy efficiency per watt makes it competitive in performance-sensitive applications, though its higher power requirements and limited early availability may constrain adoption in cost-sensitive or low-power designs. Overall, the CM5 expands the Raspberry Pi ecosystem into more demanding industrial, vision, and edge computing workloads while maintaining the ecosystem's hallmark accessibility and open development environment [23].

NVIDIA Jetson Nano

The NVIDIA Jetson Nano is a compact yet powerful single-board computer built around a quad-core ARM Cortex-A57 CPU paired with a 128-core Maxwell GPU. This architecture is specifically optimized for parallel computation, enabling efficient execution of AI inference, deep learning, and computer vision workloads at the network edge. With 4 GB of LPDDR4 RAM and full integration with NVIDIA's JetPack SDK, developers gain access to CUDA, cuDNN, and TensorRT libraries, simplifying deployment of neural networks and other GPU-accelerated tasks. The Nano features extensive I/O capabilities, including multiple CSI camera interfaces, USB 3.0, Gigabit Ethernet, and GPIO expansion, making it ideal for real-time image processing and autonomous robotics applications. Despite its strong performance profile, the Jetson Nano draws more power and incurs higher costs than platforms like the Raspberry Pi CM4, which may reduce its suitability for constrained, battery-powered, or cost-sensitive embedded systems. Nevertheless, it remains an outstanding option for developers seeking affordable access to edge AI computing and GPU-accelerated embedded development. [24].

Table 3.4 SBC Comparison

Feature	*AI-64*	*CM4*	*CM5*	*Jetson Nano*
UART	Up to 12	Up to 5	Up to 5	Up to 3
I^2C	Up to 10	Up to 5	Up to 5	Up to 4
SPI	Up to 11	Up to 5	Up to 5	Up to 2
PWM	1 Fan	Up to 2	Up to 4	1 Fan
ADC	Yes (2x 12-bit)	No	No	No
DAC	No	No	No	No
Wi-Fi (Build-In)	No	Some Configurations (2.4 GHz, 5.0 GHz)	Some Configurations (2.4 GHz, 5.0 GHz)	No
Bluetooth (Build-In)	No	Some Configurations	Some Configurations	No
RAM	4 GB	1 - 8 GB	2 - 16 GB	4 GB
eMMC	16 GB	0 - 32 GB	0 - 64 GB	16 GB
eMMC Speed	400 Mbps	100 Mbps	400 Mbps	200 MHz
Operating Voltage	5 V	5 V	5 V	5 V
Typical Current Draw	$\geq$3 A (Active)	1.400 A (Active)	.900 A (Active)	2 A (Active)
Power Consumption	15 W Typical	2 W Idle / 7 W Typical	4.5 W Typical	5 W Idle / 10 W Typical
Operating Temp.	Not Specified	–20°C to +85°C	–20°C to +85°C	–25 to 97°C
CPU Speed	2 GHz	1.5 GHz	2.4 GHz	1.43 GHz
Ethernet Speed	10 - 1,000 Mbps	1,000 Mbps	1,000 Mbps	10 - 1,000 Mbps
GPIO	Up to 10	Up to 28	Up to 30	Up to 15
Cost		30 - 95 USD	45 - 135 USD	129 USD

Selection

The Raspberry Pi CM4 was selected for its balance of processing power, community support, and integration flexibility. It offers sufficient computational resources for automated network scanning and analysis, with native Ethernet and USB support ideal for our device.

Following the selection of the Compute Module 4 as our core platform, we compared the available CM4 configurations to determine the most appropriate model for our system requirements. These configurations differ in RAM size, eMMC storage capacity, and wireless connectivity options.

Part Selection

CM4001000
The CM4001000 represents the most minimal configuration within the Compute Module 4 family, featuring 1 GB of LPDDR4 RAM and no onboard eMMC storage. This variant is well-suited for highly constrained embedded systems that depend on external storage media, such as SD cards, USB devices, or network-based boot mechanisms. Its reduced memory capacity limits its suitability for applications involving parallel computation, large data structures, or advanced graphical workloads. However, the absence of eMMC and the reduced RAM footprint result in the lowest cost, smallest thermal budget, and simplest integration requirements across the CM4 lineup. As a result, the CM4001000 is often selected for deterministic single-purpose devices where cost efficiency, minimal power draw, and ease of deployment outweigh the need for extensive local processing resources.

CM4004008
The CM4004008 upgrades the baseline CM4 capabilities by incorporating 4 GB of LPDDR4 RAM and 8 GB of onboard eMMC storage, offering a balanced configuration for mid-range embedded workloads. The increased memory capacity makes this variant significantly more capable for concurrent tasks, buffered I/O operations, and applications requiring moderate data handling. The inclusion of eMMC enables faster and more reliable storage access compared to SD-based solutions, improving boot times and system responsiveness. These characteristics make the CM4004008 an attractive choice for embedded control systems, lightweight data acquisition platforms, and edge devices that benefit from improved local storage without incurring the cost of high-capacity eMMC. Its combination of affordability and enhanced performance makes it a common selection for general-purpose deployments.

CM4008032
The CM4008032 sits at the upper end of the non-wireless CM4 variants, providing 8 GB of LPDDR4 RAM and 32 GB of eMMC storage for demanding compute-oriented applications. The expanded memory supports complex real-time analytics, multi-threaded processes, and workloads that involve caching or large data buffers. With a sizable eMMC module, this variant accommodates applications requiring substantial local datasets, system logs, or containerized runtime environments. Importantly, the absence of wireless radios (Wi-Fi and Bluetooth) makes the CM4008032 particularly suitable for security-sensitive, EMI-constrained, or compliance-driven environments where wireless interfaces are either undesirable or explicitly prohibited. Typical use cases include industrial automation controllers, forensic network appliances, and instrumentation systems requiring high reliability and controlled communication pathways.

CM4108032
The CM4108032 is one of the most fully featured offerings in the Compute Module 4 series, combining 8 GB of LPDDR4 RAM, 32 GB of eMMC storage, and integrated dual-band Wi-Fi and Bluetooth connectivity. This variant delivers the highest level of system integration, enabling both high-performance local computation and versatile communication options within a compact embedded platform. Its wireless subsystems allow seamless deployment in mobile, distributed, or IoT-centric environments without requiring additional

communication modules. As such, the CM4108032 is well-suited for advanced edge-computing devices, portable instrumentation, fleet-connected systems, and gateways that aggregate or preprocess data before upstream transmission. The combination of storage, memory, and wireless hardware makes this configuration ideal for applications prioritizing flexibility, high throughput, and minimal external component requirements.

Table 3.5 Compute Module Configuration Comparison

Option	*Wireless*	*RAM*	*eMMC*	*Cost*	*Summary*
CM4001000	-	1GB	Lite	$30.00	Insufficient for parallel scanning or running multiple network tools; external storage requirement adds complexity.
CM4004008	-	4GB	8GB	$55.00	Can support basic scanning functions, but limited RAM restricts simultaneous tool execution and local report generation.
CM4008032	-	8GB	32GB	$90.00	Strong balance of performance and storage for network auditing, ideal for handling multiple processes and maintaining logs without lag.
CM4108032	Yes	8GB	32GB	$95.00	Adds wireless connectivity, improving remote control capability, but not essential since the PwnBoxer includes a separate ESP32 module for wireless communication.

Selection

The Raspberry Pi Compute Module 4 CM4008032 was selected for its optimal combination of high memory capacity, ample onboard storage, and cost efficiency. With 8 GB of RAM

and 32 GB of eMMC storage, it provides sufficient resources for simultaneous network scanning, data logging, and report generation without external storage dependencies. The absence of wireless connectivity ensures reduced interference and enhanced security, aligning with the project's focus on wired network auditing. Compared to other CM4 variants, the CM4008032 offers the best balance between performance, storage, and price, making it the most suitable configuration for the PwnBoxer system's operational and budgetary requirements.

3.1.4 Display Subsystem

Technology Comparison and Selection
The display technology chosen for our system is a capacitive touch-enabled thin-film-transistor liquid-crystal display (TFT LCD). To make this selection, several display technologies were compared based on responsiveness, interactivity, cost, legibility, and ease of use during development.

Thin-Film-Transistor Liquid Crystal (TFT LCD)
Thin-Film-Transistor Liquid Crystal Displays are among the most widely adopted technologies for embedded graphical interfaces, owing to their technological maturity, broad availability, and cost-effectiveness. The active-matrix structure allows precise pixel control through individual transistors, supporting higher refresh rates and improved image uniformity compared to passive alternatives. TFT panels typically achieve brightness levels between 400 and 1000 nits, ensuring adequate legibility in a range of lighting environments, including well-lit indoor settings. Their accurate color reproduction and low latency render them suitable for interactive kiosk and instrumentation applications that demand immediate user feedback.

From an integration standpoint, capacitive touch overlays interface conveniently through USB or I^2C communication channels, and the widespread availability of open-source libraries such as LVGL, Adafruit_GFX, and LittlevGL facilitates rapid GUI development. For faster refresh rates, 8-bit and 16-bit parallel communication between the screen and microcontroller are available also. Although TFT displays exhibit lower contrast ratios than emissive technologies such as OLED and require continuous backlighting, these disadvantages are offset by their durability, predictable supply chain, and extended service life (often exceeding 50,000 hours). Consequently, TFT LCDs offer a well-balanced compromise between performance, power efficiency, and long-term reliability for embedded system design.

Organic Light Emitting Diode (OLED)
Organic Light Emitting Diode (OLED) displays employ self-emissive pixels that produce light directly, thereby eliminating the need for a backlight. This characteristic enables exceptional contrast ratios, vibrant color reproduction, and wide viewing angles, enhancing the perceived image quality for end users. The absence of a backlight also provides significant power savings in scenarios where darker imagery dominates, making OLED displays

appealing for energy-constrained or battery-operated embedded devices. Their slim mechanical profile further supports compact and aesthetically refined product designs.

Despite these benefits, several limitations restrict OLED suitability for larger embedded applications such as kiosks or control panels. Manufacturing costs for panels exceeding 5–7 inches remain high, which can substantially elevate the overall system cost. Additionally, OLED materials are susceptible to differential pixel aging and image retention under prolonged static display conditions, which is a critical drawback for installations that present fixed interface elements. Software driver support on embedded platforms such as the Raspberry Pi or ESP32 is comparatively limited, often requiring device-specific customization. When combined with capacitive touch overlays, the integration process may introduce electromagnetic interference and calibration challenges. Considering these characteristics, OLED technology is most appropriate for premium, low-power products rather than a networking tool which will be continuously powered.

Memory-In-Pixel (MIP)

Memory-In-Pixel (MIP) technology incorporates storage elements within each pixel cell, enabling individual pixels to retain their state without continuous refresh. This architecture results in significantly reduced power consumption and is particularly effective for low-update-rate applications such as wearable or Internet-of-Things (IoT) devices. MIP displays maintain legibility under bright ambient conditions due to their reflective or transflective optical designs, often outperforming conventional backlit displays outdoors. The capacity for partial refresh further enhances efficiency by limiting pixel updates to regions where content has changed.

These advantages come with trade-offs in performance and display capability. MIP panels generally exhibit slower refresh rates and reduced color depth relative to TFT and OLED technologies, limiting their suitability for real-time, interactive, or multimedia-rich user interfaces. Additionally, the commercial availability of large-format MIP displays with integrated capacitive touch functionality is limited, constraining their application in high-interactivity contexts. Consequently, while MIP displays excel in ultra-low-power, static-display environments, they are not a practical choice for dynamic, touch-driven kiosk or control-panel systems where responsiveness and visual fluidity are essential.

Table 3.6 Display Technology Comparison

Display Type	*Advantages*	*Drawbacks*	*Typical Use Case*
TFT LCD (Capacitive)	Low cost; good brightness; wide driver support; proven CM4 compatibility	Higher power draw; lower contrast	Kiosk, industrial UI
OLED	High contrast; wide viewing angle	Expensive; burn-in risk; limited touch options	Wearables, premium devices
MIP	Extremely low power; retains static images	Low refresh; limited color; rare in large sizes	E-paper, IoT displays

Selection

The final selection for the display subsystem is a **TFT LCD** with a capacitive multi-touch overlay, driven directly by the **ESP32** over **SPI** using the **eTFT_spi** driver and rendered with **LVGL**. This updated configuration offloads all user-interface rendering from the Compute Module 4 (CM4) to the microcontroller, reducing system RAM usage and improving deterministic frame timing. The SPI bus operates at 8–16 MHz with DMA-backed transfers, providing smooth, flicker-free screen updates for status dashboards, progress indicators, and setup menus.

Touch input is handled via an I^2C controller attached to the ESP32, which maps coordinates directly into LVGL's input subsystem for seamless GUI interaction. This reduces wiring complexity and allows the CM4 to focus on network-scanning and orchestration tasks rather than UI rendering. From a power standpoint, the display's LED backlight remains PWM-controlled through the ESP32's GPIO, allowing dynamic brightness scaling during low-activity periods.

The chosen architecture simplifies integration, as the ESP32 handles both control and display functions under a single firmware stack, while the CM4 communicates over SPI for updates and event reporting. Compared to the previous HDMI-driven approach, this design minimizes latency, improves reliability, and better aligns with the embedded focus of the PwnBoxer system.

Part Selection and Comparison

In the following section, three distinct TFT LCD parts will be considered for use in PwnBoxer. Specifications that will be under review for each part are as follows: size, luminance, contrast ratio, interface type, power supply requirements, resolution, touchscreen technology type and cost. The PwnBoxer system is a somewhat compact device that will operate primarily indoors, and aims to be as efficient as possible while providing the user with a clean and readable display experience. The device will operate in near-ideal conditions most of the time, likely not requiring the use of gloves or a stylus.

Crystalfontz CFAF320480C7-035TC

The Crystalfontz CFAF320480C7-035TC is a 3.5-inch full-color TFT LCD module with a 320×480 pixel resolution and capacitive touch support, integrating the Ilitek ILI9488 display controller and FT5436 touch controller. Designed for embedded applications, it operates from a single 3.3 V power source for both logic and analog domains, simplifying power management. The module supports multiple interface configurations, including 8-, 9-, 16-, and 18-bit parallel, as well as 3- or 4-wire SPI, making it highly adaptable for systems with varying bandwidth and pin availability. The I^2C-based touch interface operates at address 0x38 and supports up to five simultaneous touch points, enabling responsive and modern multi-touch interaction. Its optical performance includes a brightness of approximately 300 cd/m^2, a contrast ratio around 1000:1, and a typical response time of 20–40 ms, making it well-suited for indoor embedded systems like PwnBoxer.

Mechanically, the module is compact (55.5 mm × 84.96 mm × 4.25 mm) and lightweight (42 g), with a 6 o'clock viewing orientation. The LED backlight operates at 3.2 V and

typically draws 120 mA, with an estimated lifetime of 50,000 hours. The display's wide operating temperature range (-20 to 70 degrees Celsius) ensures robustness for industrial and field environments. Its electrical characteristics, particularly the low 8-16 mA logic current consumption, make it efficient for microcontroller-based systems. Additionally, the 3.5" diagonal measurement of the display provides a good balance between readability and power consumption [25].

Newhaven Display NHD-1.8-128160EF-SSXN-FT

The Newhaven Display NHD-1.8-128160EF-SSXN-FT is a 1.8-inch full-color TFT LCD display with 128 x 160 pixel resolution, optimized for low-power systems and handheld interfaces. For touch capability, the NHD module offers 4-wire resistive touch rather than a capacitive touch module. Resistive touch screens detect input through physical pressure between conductive layers, allowing use with any object or gloves, while capacitive touch screens sense changes in an electrostatic field from a finger or stylus. This means that capacitive touch typically offers higher precision, multi-touch capability, and improved optical clarity, while resistive touch provides greater usability and reliability when used with gloves or in harsher environments. The NHD operates from a single 3.0–3.3 V supply, making it electrically compatible with the ESP32's 3.3 V output pin. The display integrates an ST7735S controller, which supports both 3-wire and 4-wire SPI interfaces, offering a solution for reduced pin usage compared to an 8-bit parallel interface.

Typical luminance for the Newhaven Display is 850 cd/m^2, with a contrast ratio of 500:1, which is sufficient for clear visibility in indoor and moderately bright conditions. The backlight is composed of white LEDs driven by a constant-current source, typically consuming 100 mA at full brightness, and can be dimmed via PWM control if power conservation is required. The NHD-1.8-128160EF-SSXN-FT's small footprint and low profile make it suitable for space-constrained designs, but presents a significant lack in readability for the user when compared to larger display options [26].

Adafruit Color TFT Display CH700WV03A-T

The Adafruit 7-inch TFT display module is a large-format, full-color display designed for embedded systems that require a vivid, interactive interface. Featuring a native resolution of 800×480 pixels, it provides crisp, detailed graphics suitable for control panels, dashboards, and user terminals. The display operates from a single 3.3 V power supply with a typical current draw of 140 mA, making it compatible with a wide range of microcontroller and single-board computer platforms. Its 40-pin interface supports a 24-bit parallel RGB data bus, enabling high-speed image updates and smooth color rendering without the need for complex serial communication protocols. The integrated resistive touch panel offers reliable, single-point input through pressure detection, allowing operation with gloves, styluses, or any object. This makes it particularly well-suited to industrial embedded applications or systems in harsher environments.

The module achieves a contrast ratio of approximately 500:1 and a typical luminous intensity of 280 cd/m^2, ensuring strong color performance and readability under standard indoor lighting. The LED backlight delivers uniform illumination across the wide 7-inch surface

while maintaining efficient power consumption. The resistive touch layer is both durable and cost-effective, providing tactile responsiveness with a simple analog interface that can be easily read by most microcontrollers through ADC channels. Disadvantages for this part pertain to the size, as a 7-inch display is likely to require much more power than something smaller, and the resistive touch capability when compared to capacitive touch technology for this use case [27].

Table 3.- Display Part Comparison

Specification	**CFAF320480C7-035TC**	**NHD-1.8-128160EF-SSXN-FT**	**CH700WV03A-T**
Size (in)	3.5	1.8	7.0
Cost (USD)	$\sim$ \$35	$\sim$ \$20	$\sim$ \$60
Contrast Ratio	1000:1	500:1	500:1
Luminance (cd/m^2)	300	300	280
Resolution (pixels)	320 × 480	128 × 160	800 × 480
Interface	8/9/16/18-bit Parallel, I^{2}C (Touch)	3/4-wire SPI	40-pin 24-bit Parallel
Supply Voltage (V)	3.3	3.0–3.3	3.3
Touch Type	Capacitive	Resistive	Resistive

Selection

The Crystalfontz CFAF320480C7-035TC is the optimal choice for the system due to its balanced combination of performance, flexibility, and ease of integration with microcontroller-based embedded platforms like PwnBoxer. Its 3.5-inch TFT LCD provides a high-resolution 320×480 pixel display with vivid color reproduction and a 1000:1 contrast ratio, ensuring clear, legible visuals for dynamic graphical user interfaces. The integrated capacitive touch layer offers responsive multi-touch capability, enabling intuitive and modern interaction while maintaining a compact form factor suitable for embedded systems. Electrically, the display supports 8-bit and 16-bit parallel interfaces for fast display refresh rates, and I^{2}C for the capacitive touch interface, which allows for robust communication between the display and microcontroller without consuming excessive GPIO resources. Additionally, the module operates efficiently on a single 3.3 V supply with low current draw, making it compatible with low-power designs, and its long LED backlight lifetime ensures durability and reliability for continuous operation. Combined with mature driver support, extensive software libraries, and predictable availability, the CFAF320480C7-035TC delivers a compelling balance of responsiveness, visual quality, and embedded-system compatibility, making it the most suitable display for PwnBoxer.

3.1.5 Voltage Regulation Subsystem

Technology Comparison and Selection

Voltage regulation is required in this system due to the varied voltage requirements of the components of the other involved subsystems. The voltage regulation subsystem would be placed in between the load and the subsystem that requires the voltage. There are two primary types of voltage regulators, linear and switching regulators. Linear regulators provide a simple, low-noise way to step down voltage but waste energy as heat, making them inefficient for large voltage differences. Switching regulators, on the other hand, use transistors

and inductors to efficiently step up or down voltage with higher complexity and more electrical noise. Therefore we have chosen to use a switching regulator from Texas Instruments to regulate the voltage between each of our subsystems.

Linear Regulators

Linear voltage regulators operate by continuously adjusting a pass transistor to maintain a constant output voltage. They work much like variable resistors, dissipating excess voltage as heat to drop the input voltage to the desired output. Because they regulate by linear conduction, their operation is simple and produces minimal electrical noise, making them ideal for powering sensitive analog or RF circuits. They also require very few external components, often just a pair of capacitors for input and output filtering, which makes circuit design straightforward and compact.

However, linear regulators are inefficient when the difference between input and output voltage is large or when supplying high currents. The efficiency is roughly equal to the ratio of output voltage to input voltage, meaning significant energy can be wasted as heat. This not only requires larger heat sinks but can also limit their use in battery-powered or thermally constrained systems. Additionally, linear regulators are only capable of performing step-down voltage regulation. Despite these drawbacks, linear regulators remain popular in low-power applications or as post-regulators for switching supplies where clean, low-noise voltage is essential.

Switching Regulators

Switching voltage regulators use high-speed switching elements and energy storage components such as inductors and capacitors to convert electrical energy efficiently. Instead of dissipating the excess energy as heat, they store and transfer it in discrete packets using pulse-width modulation or similar techniques. This allows switching regulators to achieve higher efficiencies even when stepping down from high input voltages or delivering large output currents. They can also perform step-up, step-down, or even inverting voltage conversions, giving them flexibility that linear regulators cannot match.

The main disadvantages of switching regulators are complexity, cost, and noise. Their circuits require more external components like inductors, diodes, and high-frequency capacitors, along with careful PCB layout to minimize electromagnetic interference. The fast switching can introduce voltage ripple and high-frequency noise, which may interfere with sensitive analog circuits unless filtered properly. Despite this, switching regulators dominate modern power supply design for high-efficiency, portable, or high-current systems such as laptops, embedded devices, and renewable energy converters.

Table 3.7 Linear and Switching Voltage Regulator Compatison

Specifications	Linear Regulator	Switching Regulator
Cost	Low cost; inexpensive components and simple design	Higher cost; requires more components (inductor, diode, capacitor, switch)
Efficiency	Low (typically 30–60%); energy lost as heat	High (typically 80–95%); minimal energy wasted
Heat	Generates significant heat due to voltage drop across pass transistor	Much cooler operation due to efficient power conversion
Voltage Output (relative to input)	Can only step down	Can step up, step down, or invert voltage
Complexity	Simple design and easy to implement	Complex design; requires careful layout and EMI considerations

3.1.6 Power Supply Subsystem

Technology Comparison and Selection

Upon calculating the estimated power usages for each major subsystem, the power output of the PwnBoxer device will be around 7.7557 W (power requirements and usage calculations shown in Table 3.1.6.1). Given this power requirement, it makes sense to consider several connector options to balance flexibility, cost, and deployment needs. Additionally, these technologies will be compared on their simplicity of implementation, standardization, and safety.

Table 3.8 Power Usage Breakdown

Device	**Power Requirement**	**Power Usage**
ESP32-S3	3.3 V, 90 mA	297 mW
Raspberry Pi CM4	5 V, 1.4 A	7 W
TFT LCD Display	3.3 V, 139 mA	458.7 mW

Technologies to be considered in this section are the 12 V DC barrel jack, USB Type-A, Micro-USB Type-B, USB Type-C, and Power over Ethernet (PoE). A 12 V DC barrel jack provides a straightforward and reliable input for wall adapters or bench supplies, making it ideal for development environments or fixed installations where power is stable. Similarly, USB Type-A and Micro-USB Type-B ports offer compatibility with the vast ecosystem of 5 V USB power sources and cables. USB Type-C can deliver anywhere from 5 V up to 20 V through USB Power Delivery negotiation, enabling higher efficiency by stepping down from a higher bus voltage and reducing cable losses. PoE allows both data and power over a single Ethernet cable, which is especially valuable for embedded systems in networked or remote installations.

12 V DC Barrel Jack Connector

A 12 V barrel jack offers a simple, robust, and widely compatible solution for powering the PwnBoxer system. Its main advantage over the other connector types is the ease of integration, as it does not contain any circuitry for communication. This allows for a the integration of a simple method to directly supply 12 V to the switching regulators, where both of them will only be required to step-down the voltage to 5 V and 3.3 V for the Raspberry Pi CM4 and ESP32-S3, respectively. The connector is inexpensive, easy to source, and supports a wide range of wall adapters and bench supplies, making it convenient for both prototyping and production. The 12 V DC barrel jack can also safely handle the estimated current draw of about 0.65 A for the system with ample buffer, particularly when using a 2.1 or 2.5 mm connector rated for 2 A or higher. Additionally, 12 V DC barrel jack connectors are mechanically very easy to mount to a printed circuit board, and are very suitable for a system in which the cable would remain connected most of the time, as PwnBoxer is intended to be.

Looking at the limitations of the 12 V DC barrel jack connector, the most glaring is the lack of standardization. There are many connectors of varying inner and outer diameters and polarity conventions, increasing the risk of incorrect connections or damage if the wrong adapter is used. The lack of communication features brings with it a lack of protection features, meaning reverse polarity, overvoltage, or short-circuit protection must be implemented on the board. For these reasons, while a 12 V barrel jack is excellent for simplicity and reliability, it should be complemented by protection circuitry and potentially other power input options for flexibility.

USB Type-A and Micro-USB Type-B

USB Type-A and MicroUSB Type-B connectors provide excellent compatibility and accessibility advantages for the PwnBoxer system, as both are widely used and supported across countless consumer and development devices. This makes them convenient for powering or interfacing with standard 5 V USB power sources. Both are compact, inexpensive, and easy to integrate into a PCB layout, which simplifies both prototyping and production. For a system requiring around 7.76 W, these connectors can supply sufficient power when paired with a quality 5 V, 2 A adapter. Additionally, their ubiquity makes them ideal for testing, firmware updates, and powering the system through readily available cables, reducing the need for specialized hardware during development or field service. This will improve the life cycle of the PwnBoxer system, as the power supply system can be updated as needed.

However, these connectors also have notable limitations for a power-hungry or performance-focused design. Micro-USB Type-B and USB Type-A connectors are mechanically less robust and prone to wear or damage after repeated insertion cycles, which can reduce long-term reliability in frequently handled devices. Both connectors are also limited to 5 V operation and lack the power negotiation and higher voltage capabilities offered by USB Type-C or PoE, which can lead to higher current draw and greater resistive losses over cables. In addition to the robustness of the MicroUSB Type-B cable, the asymmetrical nature of the connector's plug orientation can be inconvenient to the end user. While USB Type-A connectors can be used to offer power and data to devices, they are typically designed for

use with hosts. This means careful consideration will be required when configuring the circuitry to avoid any incorrect connections. Although these connectors are cost-effective and easy to source, they are best suited for low-to-moderate power designs or for auxiliary power and debugging interfaces rather than as the primary power input in modern embedded systems.

USB Type-C

USB Type-C provides a modern and versatile power interface for the PwnBoxer system. Unlike USB Type-A or Micro-USB Type-B, Type-C supports flexible voltage and current delivery through the USB Power Delivery (PD) specification. While standard USB-C without PD negotiation supplies 5 V, implementing PD negotiation allows the system to request higher voltages such as 9 V, which is advantageous for regulator efficiency and system stability. The total power requirement of the PwnBoxer system is approximately 7.7557 W, and although 5 V at 3 A (15 W) would meet the power budget, stepping up to 9 V via PD negotiation improves overall power conversion efficiency. Supplying 9 V to a high-efficiency buck regulator reduces input current compared to 5 V operation, minimizing resistive losses, voltage drop across traces, and thermal stress on connectors and PCB copper. This result may improved thermal performance and more reliable voltage regulation under dynamic load conditions, particularly during peak processing activity from the CM4.

Implementing USB-C with Power Delivery negotiation introduces additional circuit complexity compared to legacy connectors. The design must properly manage the Configuration Channel (CC) pins to negotiate voltage profiles with compliant power sources. This typically requires either a dedicated PD controller IC or a microcontroller-based PD negotiation interface.

Additionally, protection circuitry must be included to ensure that only the negotiated voltage is delivered to the downstream regulator stage. Over-voltage protection, current limiting, and transient suppression components are required to guard against improperly configured or non-compliant power adapters. USB-C connectors are also slightly more expensive and less sturdy than traditional barrel jacks and require careful PCB layout due to their higher pin count. However, these tradeoffs are justified by the electrical and mechanical advantages provided by controlled PD voltage negotiation.

Power over Ethernet (PoE)

Power over Ethernet provides a unique and compelling solution for powering the PwnBoxer system, especially since it already relies on a network connection. By delivering both data and power through a single Ethernet cable, PoE simplifies installation, reduces wiring complexity, and minimizes the number of external power adapters or connectors required. PoE also supplies sufficient power for this use case, as it supports standardized power delivery (up to 15.4 W for PoE, 25.5 W for PoE+, and 60–90 W for newer standards), which easily exceeds the 7.7557 W usage requirement for PwnBoxer.

However, integrating PoE introduces additional design and cost considerations. Implementing it requires a PoE PD (Powered Device) controller and an isolated DC-DC converter to safely step down the 48 V line voltage to your system's 12 V or 5 V rails. This adds

circuit complexity and board space compared to simpler connectors like a barrel jack or USB-C. Ethernet transformers and isolation requirements can also increase BOM cost and design effort. Moreover, PoE infrastructure depends on compatible switches or injectors, which can add expense and limit portability if power must come from a traditional outlet. While PoE is incredibly advantageous for many networking devices, it is likely overkill for our system, and adds too much complexity to the design to outweigh its benefits.

Table 3.9.1 Comparison of Power Input Options

Specification	**USB Type-C**	**USB Type-A**	**Micro-USB Type-B**
Voltage Input	5/9/12 V	5 V	5 V
Max Current	3 A (at 9 V)	2.5 A	2 A
Max Power	27 W	12.5 W	10 W
Plug Orientation	Reversible	Non-reversible	Non-reversible
Mechanical Durability	Low	High	Low
Complexity of Implementation	High	Low	Low

Table 3.9.2 Comparison of Power Input Options

Specification	**12 V Barrel Jack**	**PoE**
Voltage Input	12 V	44 V
Max Current	2 A	0.35 A
Max Power	24 W	15.4 W
Plug Orientation	Non-reversible	Non-Reversible
Mechanical Durability	High	High
Complexity of Implementation	Low	High

Selection

USB Type-C with Power Delivery negotiation was selected as the primary power interface for the PwnBoxer system. By negotiating a 9 V input, the system achieves improved regulator efficiency, reduced input current, and enhanced thermal performance compared to 5 V-only operation. Although PD negotiation increases circuit complexity and slightly raises component cost, it provides better electrical headroom for downstream voltage regulation and improves overall system reliability. The widespread adoption of USB-C ensures compatibility with modern chargers and power supplies, while the reversible connector enhances usability and durability. This approach balances electrical performance, modern compatibility, and long-term scalability, making USB Type-C with PD negotiation the most suitable power solution for the PwnBoxer design.

3.2 Communication Protocols

3.2.1 Communication between CM4 and ESP32-S3

Technology Comparison and Selection

The PwnBoxer architecture requires two distinct classes of interconnects: (i) a **simplistic,**

low-latency, short-reach link between the Compute Module 4 (CM4) and the ESP32 microcontroller for status, control, and bulk telemetry, and (ii) **high-speed, robust** control links for optional sensors and peripherals. We evaluated **SPI**, **UART**, and **I^2C** against the following criteria: wiring complexity, software overhead, throughput, and robustness. The CM4 and ESP32 both operate at 3.3 V logic, allowing direct coupling without level shifting for all three buses.

SPI

SPI is the best choice when high throughput and low latency are priorities between the Raspberry Pi CM4 and ESP32-S3. Unlike UART and I^2C, SPI is fully synchronous and uses a shared clock line, which allows both devices to operate at much higher speeds, often in the tens of megahertz range. The protocol supports full-duplex data transfer, and large buffers can be transferred quickly with very little overhead. Both the Pi and ESP32-S3 support SPI hardware acceleration, meaning you can push significant amounts of data with predictable timing and minimal CPU load.

The trade-off is that SPI requires more wires and slightly more setup. You need at least four lines: MOSI, MISO, SCLK, and CS, and you must explicitly configure one device as the master and the other as the slave, which requires additional configuration compared to UART's simple point-to-point design. That said, once SPI is initialized, it is highly reliable and scales well to demanding applications such as streaming sensor data, transferring buffers, or synchronizing tasks between the two systems. If raw speed matters more than wiring simplicity, SPI outperforms UART and I^2C by a wide margin. However, since the priority for selection over a communication protocol between the Raspberry Pi CM4 and the ESP32-S3 is simplicity, SPI will not be the greatest choice.

UART

UART is generally the easiest and most lightweight communication protocol to use between a Raspberry Pi CM4 and an ESP32-S3. It requires only two data lines, TX and RX, plus a shared ground, and both devices have dedicated UART hardware with incredibly simple and mature software libraries. Because UART is asynchronous, you don't need to worry about clock synchronization, making setup extremely simple. Configuration usually comes down to choosing a baud rate, parity, and stop bits, which both Linux (on the Pi) and ESP-IDF (on the ESP32-S3) support natively. For many applications, especially text-based or command-based communication, UART is the most straightforward option with the least wiring and coding complexity.

The main limitation of UART is speed and reliability at higher data rates. Typical UART communication is stable up to around 1–2 Mbps, but pushing beyond that can introduce noise sensitivity, especially over longer wires. UART is also strictly point-to-point, so it cannot scale to multiple devices on the same bus, and it lacks the deterministic timing of synchronous protocols like SPI. For applications that require large amounts of data, rapid sensor updates, or low-latency streaming, UART may not offer enough throughput. However, for implementation in PwnBoxer, the amount of data that is required to be sent fits within the bounds of UART.

I²C

I²C uses only two shared lines, SDA and SCL, which makes wiring extremely simple, and both the Raspberry Pi CM4 and ESP32-S3 support it in hardware. It is designed for communication with multiple peripheral devices, and its addressing system allows many nodes to operate on the same bus. If you were connecting sensors, displays, or low-bandwidth peripherals to either board, I²C would be the most convenient option. The software support on both platforms is also mature, making it easy to get basic communication working with minimal code.

However, I²C is not ideal when establishing a fast, robust link strictly between two microcontrollers. Standard mode top-speeds, 100 kHz to 400 kHz, are quite slow, and even the fast modes, 1 MHz to 3.4 MHz, fall short of SPI's performance. The protocol also adds overhead such as acknowledgments, start/stop conditions, and pull-up resistor balancing, which further reduces effective throughput. I²C is more prone to noise, and long wires can destabilize the bus. For a simple and fast processor-to-processor link, I²C's ease of wiring does not compensate for its speed and reliability limitations.

Table 3.10 Protocol Technology Comparison

Protocol	*Nominal Rate*	*Duplex*	*Wires (min.)*	*Topology*	*SW Stack*	*Typical Use*
SPI	1–20 MHz (project 8–16 MHz)	Full	4 (SCLK, MOSI, MISO, CS)	Point-to-point, multi-CS	Medium	CM4 ↔ ESP32 high-rate control
UART	9.6 kbps–3 Mbps (proj. 115200)	Half	2 (TX, RX) + GND	Point-to-point	Low	Console, debug, recovery shell
I²C	100 k/400 k/1 MHz	Half	2 (SCL, SDA) + pull-ups	Multi-drop bus	Medium	Low-speed sensors, GPIO expanders

Environmental, EMI, and Recoverability Considerations

UART offers generally good environmental and EMI resilience for short, internal board-to-board links like the one used in PwnBoxer, but there are important considerations to ensure reliable operation. Because UART is an asynchronous protocol without a shared clock, it can be more sensitive to noise if long traces or cables are used; however, within a compact enclosure and with short PCB routing, EMI impact is minimal. Proper grounding and avoiding parallel runs near high-current switching components further protect signal integrity. From a recoverability standpoint, UART is forgiving, if data corruption occurs due to interference, only the affected bytes are lost, and higher-level software can easily detect malformed packets and request retransmission. Additionally, UART's stateless nature allows communication to restart cleanly after resets or brownouts on either processor, making it a robust option for environments where occasional interference or unexpected resets may occur.

Table 3.11 Electrical Characteristics and Integration

Protocol	*Voltage/Levels*	*Termination*	*ESD/EMI Notes*	*CM4/ESP32 Integration*
SPI	3.3 V CMOS	Optional series 22–47 Ω	Short, shielded ribbon; ground adjacent to SCLK	Native controllers on both; DMA-friendly on CM4
UART	3.3 V CMOS (TTL)	None (short runs)	Optional TVS at connector; avoid RS-232 levels	Native; CM4 mini-UART or PL011; ESP32 UARTx
I^2C	3.3 V open-drain	2.2–4.7 kΩ pull-ups	Keep stubs short; consider bus buffers if long	Native; shared bus for optional sensors

Selection

We select UART as the *primary* CM4 ↔ ESP32 transport. UART is the most practical choice for communication within the PwnBoxer device because it offers the ideal balance of simplicity, reliability, and sufficient performance for the system's needs. With only two data lines required and no need for a shared clock, UART minimizes wiring complexity and reduces the chances of setup errors during development and assembly. Both the Raspberry Pi CM4 and ESP32-S3 provide robust hardware UART support, enabling a stable, low-latency link that is easy to configure in software. While UART does not reach the maximum speeds of SPI, its available baud rates are more than adequate for exchanging status updates, control messages, and moderate data payloads between the processors. This makes UART a clean and efficient choice that supports PwnBoxer's emphasis on straightforward integration without sacrificing dependable communication.

Table 3.12 CM4 ↔ ESP32 UART Pin Map

Signal	*CM4 Pin*	*ESP32 Pin*	*Notes*
UART_TX	GPIO14 (UART5_TX)	GPIO RX Pin (e.g., GPIO 44)	CM4 TX → ESP32 RX
UART_RX	GPIO15 (UART5_RX)	GPIO TX Pin (e.g., GPIO 43)	CM4 RX ← ESP32 TX
GND	Any GND	Any GND	Required common reference
3V3 (Optional)	3V3 Power Rail	3V3 Input	Only used if powering ESP32 from CM4; not required for UART

3.2.2 Communication Between ESP32-S3 and TFT LCD

Part Comparison and Selection

In designing the communication between the ESP32-S3 and the Crystalfontz CFAF320480C7-035TC TFT display module, there are three options listed in the documentation that we may

choose from. 4-wire SPI, DOTCLK RGB with SPI initialization, and 8/9/16/19-bit parallel. For capacative touch capability, the only available option is I^2C communication separate from the display wiring. In choosing the display communication protocol, the primary considerations are frames per second, simplicity, and robustness. In the next sections, the three communication protocols will be compared, and one will be selected for use in the PwnBoxer system.

4-Wire SPI

The 4-wire SPI interface offers the simplest physical and logical connection among the available communication modes, and uses only MOSI, SCL, CS, and D/C. This low pin count reduces routing complexity, cuts down on EMI exposure, and makes integration straightforward in compact embedded systems. Most microcontrollers provide dedicated SPI hardware with flexible clocking, which allows for fast initial bring-up and easy debugging. SPI also preserves valuable GPIO resources, which is often important when pairing devices like the Raspberry Pi CM4 and the ESP32-S3.

Its simplicity, however, comes at the cost of bandwidth. Because the ILI9488 sends pixel data serially, the achievable throughput is limited even at high SPI clock speeds, and this restricts full-screen updates and smooth animations. Although SPI is not ideal for video or high-frequency UI refreshes, it is extremely robust. It tolerates longer traces, modest PCB constraints, and moderate electrical noise. For applications that rely mainly on static images or occasional screen changes, SPI is an efficient and reliable choice.

8/9/16/18-Bit Parallel

Parallel interfaces such as 8, 9, 16, or 18-bit provide dramatically higher data throughput than SPI because they move many bits at once. The wider data bus allows faster pixel writing, making partial refreshes and animation significantly more responsive. Applications that require more rapid updates, but do not need full streaming video, often land in this middle ground. As a result, parallel communication forms a strong balance between reasonable complexity and noticeable performance improvements.

The increase in bandwidth does require more engineering care. A parallel interface consumes far more GPIO pins, and the related signals must be routed carefully to maintain proper timing. This also raises susceptibility to EMI and timing violations, which means that layout discipline and signal integrity become more important. When the host processor has enough pins available and the designer is willing to manage additional routing effort, parallel delivers excellent performance while remaining simpler to implement than a full DOTCLK RGB interface.

DOTCLK RGB with SPI Initialization

DOTCLK RGB offers the highest performance of the available communication modes because it streams pixel data continuously using a pixel clock and synchronized horizontal and vertical timing signals. With a dedicated data bus and real-time refresh capability, it enables fluid motion, smooth transitions, and full-frame updates at video-rate speeds. Systems designed for rich visual output, including UI animations or video playback, benefit

significantly from this mode.

This capability does come with substantial complexity. DOTCLK requires a large number of parallel signals, along with strict timing constraints and clean PCB routing to maintain signal integrity. EMI sensitivity increases due to the constant high-frequency switching of the pixel clock and data lines. Initialization still occurs over SPI, which simplifies configuration but adds an extra step to the startup sequence. Once implemented correctly, DOTCLK RGB operates reliably and provides unmatched display responsiveness, making it the preferred option when maximum visual quality and speed are required.

Table 3.13 Comparison of Display Communication Protocols

Protocol	Wiring Complexity	Speed	Pins Used
4-Wire SPI	Simple	Slow–Medium	4–5
8/9/16/18-bit Parallel (DBI Type-B)	Medium–Difficult	Medium–Fast	10–30
DOTCLK RGB + SPI Init	Difficult	Fast	20–30
I^2C (Touch Only)	Simple	Slow	2–3

Selection

For the PwnBoxer system, 4-wire SPI is the preferred communication method because it offers the simplest and most reliable wiring configuration while still meeting the performance demands of the system. With only a few required connections, the interface minimizes routing effort, reduces opportunities for signal integrity issues, and keeps the overall hardware layout clean and compact. More importantly, experimental testing with the CFAF320480C7-035TC display has demonstrated that SPI can sustain refresh rates of roughly 30 frames per second, which is more than adequate for the intended user experience. This level of performance provides smooth transitions and responsive visual updates without the added complexity or pin overhead of parallel or DOTCLK RGB interfaces. By selecting SPI, the design achieves a strong balance between ease of integration, proven performance, and long-term reliability.

In addition to the display interface, the capacitive touch functionality relies exclusively on the I^2C protocol, which is the only communication method supported by the FT5436 touch controller integrated into the module. This constraint simplifies the architectural decision, since no alternative protocols need to be evaluated for touch input. I^2C uses only two lines, SCL and SDA, which keeps the wiring minimal and easy to integrate alongside the SPI lines for the display itself. The bus also supports interrupt-driven updates, allowing the system to respond quickly to touch events without constant polling. Because I^2C is well supported by the ESP32-S3, this arrangement ensures reliable touch performance while maintaining a straightforward hardware layout.

Table 3.14 ESP32-S3 ↔ CFAF320480C7-035TC 4-Wire SPI Pin Map

Signal	*ESP32-S3 Pin*	*Display Pin*	*Notes*
SPI_MOSI	GPIO (e.g., GPIO 11)	SDA (Pin 34)	Display SDI input, rising-edge sampled
SPI_SCLK	GPIO (e.g., GPIO 12)	SCL / WR (Pin 36)	SPI clock line
SPI_CS	GPIO (e.g., GPIO 10)	CS (Pin 38)	Active low chip-select
D/C (Data/Command)	GPIO (e.g., GPIO 13)	D/C (Pin 37)	Low = command, high = data
RST	GPIO (e.g., GPIO 9)	RST (Pin 9)	Hardware reset, active low
SDO (Optional)	GPIO (optional)	SDO (Pin 33)	Not needed for write-only SPI mode
GND	Any GND	GND (Pins 1, 32, 50)	Required common reference
3V3	3.3V Rail	IOVCC/VCI (Pins 2–5)	Display logic and analog supply

3.2.3 SPI Application Framing and Reliability

Framing

The control-plane frame is fixed-header with variable payload. To simplify parsing on both ends and guarantee forward compatibility, every frame begins with a sync word and version, followed by a type field, little-endian length, payload, and CRC-16/CCITT. Idle gaps between chip-select assertions delimit frames.

Table 3.15 SPI Frame Definition

Field	*Bytes*	*Type*	*Endian*	*Example*	*Notes*
SYNC	2	Const	N/A	0xA5 0x5A	Frame sync
VER	1	U8	N/A	0x01	Protocol version
TYPE	1	U8	N/A	e.g., 0x10=HB, 0x20=STAT, 0x30=CMD	Semantic routing
LEN	2	U16	LE	0x0030	Payload bytes
PAYLOAD	N	Opaque	N/A	…	JSON or CBOR (bounded)
CRC16	2	U16	LE	CCITT	Integrity (poly 0x1021)

Timing, Flow Control, and Buffering

The CM4 is SPI master and enforces pacing by asserting `CS_CTRL`. The ESP32 maintains a double-buffer queue (two 512 B pages) for outgoing status to guarantee progress even under transient CM4 load. Heartbeats at 1 Hz carry health counters (TX underruns, RX CRC fail, last error code) for observability. Maximum payload for routine status frames is bounded to ≤ 1 KB; bulk records (e.g., scan progress batches) are chunked.

Error Handling
On CRC failure, the receiver increments a counter and drops the frame (no retransmit at the link layer). A *negative-ack* is emulated by a health flag in the next heartbeat. Persistent failures trigger a supervised re-initialization sequence: de-assert CS, delay 2 ms, re-sync with a zero-length `TYPE=0x00` probe frame. UART remains available for field diagnostics regardless of SPI state.

3.2.4 UART and I²C Roles

UART (debug and recovery)
UART at 115200 bps is dedicated to console access and out-of-band maintenance. During bring-up and failure injection tests, UART provides deterministic recovery paths (boot logs, shell access) and remains electrically independent of the SPI ribbon.

I²C (auxiliary peripherals)
I²C at 400 kHz is reserved for optional peripherals (e.g., ambient sensor, GPIO expander, LED driver). Pull-ups are sized 2.2–4.7 kΩ depending on bus length and device count; bus-clear sequences are implemented in firmware to handle stuck-low SDA/SCL conditions.

Table 3.16 Protocol-to-Function Mapping

Function	***Traffic Characteristics***	***Selected Protocol***	***Rationale***
Status/Control CM4↔ESP32	Low-latency, bursty, duplex	**SPI**	Deterministic timing, 8–16 MHz practical
Console/Recovery	Human-paced, simplex bursts	**UART**	OOB access independent of SPI
Optional Sensors	Low-rate, multi-drop	**I²C**	Minimal wiring, shared bus

3.2.5 Scope and Interfaces (from the hardware block diagram)

Table 3.15 summarizes the software-hardware boundaries we implement.

Table 3.17 Software-Hardware Interface Summary

Interface	*Dir.*	*Transport*	*Format*	*Rate/Size / Notes*
CM4 ↔ ESP32 status	bi-dir	UART	TX/RX	115200 Baud Rate, heartbeat, progress, errors
CM4 → Targets	out	Ethernet/Wi-Fi	Nmap / OpenVAS	workload-dependent, safe profiles, bounded concurrency
UI ↔ Orchestrator	bi-dir	SPI, UART	header + len + JSON	user-driven, UI on ESP32, CM4 schedules tasks
Reports → Email/Off-site	out	SMTP/REST	MIME+PDF	≤10 MB/report, queue if offline

The kiosk UI triggers the orchestrator, which schedules discovery (Nmap) and assessment (OpenVAS), parses and stores results, optionally requests an LLM summary, and exports HTML→PDF reports for email/off-site delivery. The ESP32 provides heartbeat/progress/error telemetry over SPI so the UI reflects real-time device state.

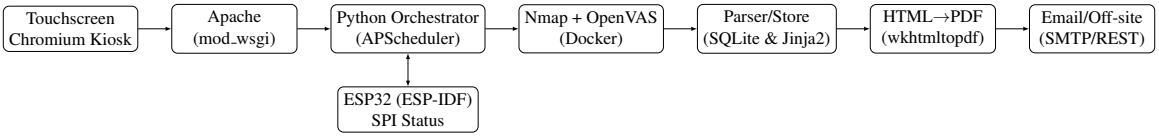


Figure 4: ***Software services and data flow aligned to hardware interfaces.***

Table 3.18 SPI Interconnect Options

Option	*Notes*	*Pros / Cons*	*Decision*
Direct 2×5 0.05" IDC ribbon	Short shielded ribbon; GND next to SCLK; keyed	+ Low cost, serviceable / – Careful pinout discipline	**Sel.**
Board-to-board mezzanine	Stiff, short, excellent SI	+ Best signal integrity / – Less serviceable	Alt.
Digital isolators on SPI	ISO77xx or similar; split grounds	+ Galvanic isolation / – Cost/latency overhead	N/A (future rev.)

3.3 Software

3.3.1 Compute-Module Operating System

Technology Comparison and Selection

We evaluated Raspberry Pi OS (Debian-based), Ubuntu Server (ARM64), and Kali Linux (Raspberry Pi build). Criteria: CM4 kernel/driver stability, baseline footprint and boot reliability, package availability, default attack surface, and ease of kiosk hardening. Raspberry

Table 3.19 OS Technology Comparison (CM4)

Option	*Security Posture*	*Ecosystem*	*Footprint*	*CM4 Drivers*	*Summary*
Raspberry Pi OS (Bookworm)	Minimal by default; hardenable	Debian (apt)	Low–moderate	First-party	Lean, predictable kernel for SPI/UI
Ubuntu Server 24.04 LTS	Frequent security updates	Broad (apt/snap)	Moderate	Good	Server features; larger idle use
Kali Linux (Pi build)	Tools-first; needs kiosk hardening	Security repos	Moderate–high	Good	Fast pentest stack; heavier base

Pi OS Lite provides the smallest baseline and first-party drivers; heavier security tooling can be containerized to avoid polluting the host.

Selection
We select **Raspberry Pi OS 64-bit Lite (Bookworm)** as the base OS for its footprint, first-party CM4 support, and low attack surface.

Table 3.20 OS Part/Product Comparison

Variant	*Source*	*Support Window*	*Notes*	*Decision*
Pi OS Bookworm Lite (64-bit)	raspberrypi.com	Tracks Debian LTS	Minimal image; ideal kiosk/container host	**Sel.**
Ubuntu Server 24.04 LTS (arm64)	Canonical	5-year LTS	Clean server base; larger RAM/flash	Alt.
Kali Linux 2025.x (Pi)	Offensive Security	Rolling	Preloaded tools; higher attack surface	No

Part / Product Comparison and Selection
This directly supports the CM4 $\leftrightarrow$ ESP32 SPI link (predictable kernel, low jitter) and keeps headroom for the kiosk browser.

3.3.2 Orchestration and Packaging

Technology Comparison and Selection
We compared **Docker+Compose**, **Podman+Quadlet**, and bare-metal **systemd** services for orchestrating software components on the Compute Module 4 (CM4). Key criteria included isolation, reproducibility, bounded overhead, straightforward updates, and maintainability in an academic or lab environment. Docker+Compose provides reproducible builds with explicit version pinning and broad ecosystem support. Meanwhile, the orchestrator and

background control daemons remain native systemd services for fast boot, deterministic startup order, and minimal runtime overhead.

Deployment Note: The end-user web portal now runs on an **external computer or server**. The CM4 submits analysis results via **HTTPS (REST)** or **SMTP** for report delivery. The on-device user interface is fully rendered by the **ESP32** using **LVGL**, which handles all display and input interactions locally.

Table 3.21 Orchestration Technology Comparison

Option	*Isolation*	*Reproducibility*	*Overhead*	*Ops Simplicity*
Docker + Compose	Namespaces/ cgroups	High (tagged images)	Moderate	Familiar UX, broad docs
Podman + Quadlet	Rootless by default	High	Moderate	Daemonless; less familiar
Bare systemd	Shared host	Medium	Lowest	Manual dependency mgmt

Table 3.22 Packaging Elements

Element	*Candidates*	*Pros/Cons*	*Decision*
Container registry	GHCR; Docker Hub; GitLab Registry	CI integration; Hub rate limits; private controls vary	**GHCR**
Init/supervision	systemd units/timers; cron	Robust dependencies; journaling; timer-based jobs	**systemd**
IPC/event bus	FS queue; Redis; MQTT	Simplicity vs added services/footprint	**FS queue**

Part / Product Comparison and Selection

This split keeps containerized services bounded (CPU/mem) while ensuring deterministic kiosk startup.

Technology Comparison and Selection

Next, we compared three major frameworks used for controlled exploitation and post-exploitation in penetration testing: **Metasploit Framework**, **Cobalt Strike**, and **Sliver**. Criteria included modular payload support, scripting interfaces, encryption options, resource footprint, and licensing.

Table 3.23 Exploit Framework Comparison

Framework	**Language**	**License**	**Strengths / Weaknesses**	**Decision**
Metasploit Framework	Ruby + Python API	Open-Source (BSD)	Mature module base; integrates with Nmap and OpenVAS; heavier runtime	**Sel.**
Cobalt Strike	Java	Commercial (TeamServer)	Powerful collaboration and pivoting; closed-source, costly license	No.
Sliver	Go	Open-Source (BC License)	Modern cross-platform agent, encrypted C2 channels; less mature scripting	Sel.

Selection
Metasploit was selected as the default exploitation framework due to its community support and integration with scanning and reporting subsystems. Sliver is kept as a lightweight backup agent, while Cobalt Strike is excluded due to licensing. The configuration fits within the 8 GB RAM and 32 GB storage limits.

3.3.3 Scanning Toolchain

Technology Comparison and Selection
For discovery we compared Nmap, Masscan, and RustScan; for vulnerability assessment we compared OpenVAS, OpenSCAP, and Nikto. Nmap provides accurate service detection and NSE scripts; OpenVAS supplies database-backed findings with CVSS and remediation notes. To stay within a 4-hour audit window on CM4, we use targeted OpenVAS profiles and cap concurrency.

Table 3.24 Discovery & Enumeration

Tool	*Strengths*	*Limits*	*Best Use*	*Decision*
Nmap	Accurate service detection; NSE scripts	Slower at huge ranges	Targeted subnets; detailed banners	**Primary**
Masscan	Extremely fast SYN scanning	Limited banner detail	Very large CIDR sweeps	Opt.
RustScan	Fast port opener; pairs with Nmap	Younger ecosystem	Pre-screening before Nmap	Opt.

Table 3.25 Vulnerability Assessment

Tool	*Strengths*	*Limits*	*Best Use*	*Decision*
OpenVAS (Greenbone)	DB-backed checks; rich reports (CVSS, remediations)	Heavier footprint	Network vulnerability scanning	**Primary**
OpenSCAP	Policy/compliance scanning	Host access required	CIS/SCAP baselines	No
Nikto	Focused web vuln scanning	Narrow scope	URL-specific web tests	Aux.
Metasploit Framework	Command and Control with vulnerability scanning, with an API to launch scanning from script-based methods.	Hard to develop for, some features require an enterprise subscription.	Vulnerability scanning, exploitation, and post-exploitation.	Aux.

Part / Product Comparison and Selection

This maps to the CM4→Targets interface: the orchestrator enforces safe profiles, throttles parallelism, and applies time budgets.

Additional Technology Comparison (Web Vulnerability Scanners)
To complement OpenVAS, we evaluated **Nuclei** for web and API endpoint testing. Metrics included template coverage, execution speed, and orchestration compatibility.

Table 3.26 Tool Packaging Choices

Tool	*Packaging*	*Run Mode*	*Decision*
Nmap	apt package; container; source	Host-native for speed; access to NSE	**apt on host**
OpenVAS	official containers; apt meta-packages; community images	Containerized DB+services (pinned)	**Docker Compose**
Aux recon	WhatWeb; sslscan; dnsenum	Small helper tools as needed	Add on demand

Table 3.27 Web Vulnerability Scanner Comparison

Tool	Language	Strengths / Weaknesses	Integration	Decision
Nuclei	Go	High-speed YAML template execution; low CPU usage; limited CVSS scoring	Triggered via orchestrator Docker	**Sel.**
OpenVAS	C	Full network scan; detailed CVSS scoring; resource-heavy	Containerized; REST API	Alt.
Nikto	Perl	Simple web scanner; outdated signatures	CLI fallback only	No

3.3.4 Local UI and Web Server

Technology Comparison and Selection

The on-device UI renders directly on the ESP32 using **LVGL** for widgets/layout and **eTFT_spi** to drive the TFT panel over SPI. This avoids browser RAM overhead, eliminates kiosk hardening, and gives deterministic frame timing. The reporting & web portal are hosted *off-device* on an external computer/server; the CM4 submits results to that server for storage and viewing.

Table 3.28 UI Approach Comparison

Approach	*Pros*	*Cons*	*Fit*	*Decision*
On-device LVGL (ESP32 + eTFT_spi)	Low RAM; deterministic; no browser; tight MCU control	Custom UI work; SPI bandwidth mgmt	Strong	**Sel.**
Web UI kiosk on Pi (Apache+Flask+Chromium)	Reuse HTML; remote debug	Browser overhead; kiosk hardening	Med.	No
TUI (curses on CM4)	Minimal resources	Poor UX for non-experts	Low	No

Table 3.29 Reporting / Portal Stack (off-device)

Component	*Options*	*Rationale*	*Decision*
Web server (external)	Apache; Nginx; Caddy	Mature auth/proxy; runs on PC/server	**Apache/Nginx (off-device)**
App framework	Flask; FastAPI; Django	Simple REST for report ingest & portal	**Flask or FastAPI**
Auth	Local users; OAuth; SSO	Ties to MSP/IT environment	**Local users or SSO**
CM4→Portal link	SMTP; HTTPS REST	Batch PDF via SMTP or JSON via REST	**HTTPS REST + SMTP fallback**

Part / Product Comparison and Selection

The on-device UI is implemented with **LVGL** for rendering and **eTFT_spi** as the ESP32 display driver over SPI. The reporting portal is hosted off-device; the CM4 submits results and PDFs to the external server for archival and user access.

3.3.5 Display and UI Library Comparison

Technology Comparison and Selection

The kiosk display requires a lightweight, touch-capable graphics library on the ESP32. We compared **LVGL**, **Adafruit_GFX**, **eTFT_SPI**, and a custom SPI framebuffer.

Table 3.30 Display & UI Library Comparison (ESP32)

Library	*Layer/Role*	*Pros*	*Cons*	*Decision*
LVGL	UI widgets / layout	Mature embedded GUI; theming, input drivers, animations; works well on ESP32 with DMA; big community	Requires integration work (drivers, input mapping); tune buffers for RAM	**Selected**
Adafruit_GFX	2D primitives	Simple API; portable; lots of examples	No full widget set; manual UI/state mgmt; less perf-tuned for complex UIs	Alt.
TFT_eSPI	Graphics helper	Fast draw routines; many controller bindings; good with ESP32	Not a full GUI framework; typically used under/alongside LVGL	Support
Custom minimal UI	Bare-metal draw	Max control, smallest deps	Highest dev/maint cost; re-implement widgets/input	No

Table 3.31 ESP32 Display Driver Options

Driver	*Bus*	*Pros*	*Cons*	*Decision*
eTFT_spi (ESP-IDF)	SPI (8–16 MHz)	Tuned for ESP32; DMA-friendly; pairs cleanly with LVGL; simple wiring	SPI bandwidth mgmt needed for big redraws	**Selected**
TFT_eSPI (Arduino)	SPI	Widely used; many controller bindings; good perf with LVGL	Arduino-centric; extra glue under ESP-IDF	Alt.
Parallel 8/16-bit custom	8080/6800	Very high throughput; tear-free updates	Many pins; harder routing; higher BOM	No
DSI/HDMI bridge	DSI/HDMI	Large panels; high res; smooth video	Extra bridge ICs; power/complexity not needed here	No

Selection
LVGL provides the best balance of performance and features for the ESP32, enabling real-time UI updates with minimal resource impact.

3.3.6 Development Environment: Languages and Repositories

The following programming languages and frameworks were selected to best meet the diverse technical requirements of this project. Each was chosen for its flexibility, performance, and ease of deployment across the three primary operational components of the system.

Languages and Frameworks

- **Python 3.x** for orchestrator and backend (Flask, APScheduler, Jinja2).
- **Bash** for service bootstrap and diagnostics.
- **C (ESP-IDF)** for ESP32 status/telemetry firmware over SPI.
- **HTML/CSS/JavaScript** for the web application UI.

Table 3.32 Repository Structure (top-level)

Path	*Purpose*	*Primary Tech*	*Notes*
`/orchestrator`	job scheduling, SPI status, scan control	Python, APScheduler	systemd service
`/ui`	web UI	Flask, Jinja2	Apache mod_wsgi
`/scanners`	OpenVAS compose, Nmap profiles	Docker, YAML	pinned digests
`/firmware`	ESP32 status loop	ESP-IDF (C)	SPI + JSON frame
`/ops`	unit tests, CI, images	GitHub Actions, scripts	backup/restore

Branching, CI, and Code Quality

- **Git (GitHub)** - trunk-based; protected `main`; feature branches via PRs.
- **CI gates** - black/ruff for Python; clang-format for C; container build with pinned digests.
- **Issues/Project** - GitHub Projects with HW/SW/Doc/Risk labels.

3.3.7 Secret and Key Management

Technology Comparison and Selection
To secure encryption keys and stored credentials, three mechanisms were evaluated: **Linux TPM/Keyrings**, **HashiCorp Vault**, and **ESP32 eFuses**. Criteria included persistence, hardware support, and isolation.

Table 3.33 Secret / Key Storage Comparison

Mechanism	Notes	Security Level	Decision
Linux Keyrings + TPM	Kernel-integrated; persistent; AES key isolation	High	**Sel.**
HashiCorp Vault	Network-based; REST API; extra container overhead	High	Alt.
ESP32 eFuses	Hardware-burned keys; one-time programmable	Medium	**Sel. (MCU)**

Framework Ecosystem Overview
Beyond Metasploit and Nuclei, additional frameworks were evaluated to broaden testing coverage and automation. Cobalt Strike and Sliver were reviewed for command-and-control (C2) simulation, providing insights into lateral movement and post-exploitation scenarios. Realm, an open-source red team framework, was also compared for extensibility and cost. For AI-assisted vulnerability analysis, platforms such as Hugging Face and Ollama were considered to integrate local large-language-model inference, avoiding reliance on external APIs. These evaluations informed our future-proofing strategy, ensuring that the PwnBoxer software stack can adapt to both traditional and AI-augmented assessment workflows.

Selection
Linux TPM keyrings handle credentials on the CM4, while ESP32 eFuses secure MCU-level keys for SPI authentication and telemetry.

3.3.8 Reporting and LLM Summarization

Technology Comparison and Selection (PDF)
We considered HTML→PDF (`wkhtmltopdf`), LaTeX (`pdflatex`), and WeasyPrint. HTML→PDF balances quality, runtime on CM4, and template reuse.

Table 3.34 Report Rendering Options

Option	*Quality*	*Runtime*	*Template Ctrl*	*Decision*
HTML→PDF (wkhtmltopdf)	High (CSS)	Moderate	High	**Sel.**
LaTeX (pdflatex)	Print-grade	Higher	High	Alt.
WeasyPrint	Good	Moderate	High	Alt.

Technology Comparison and Selection (LLM)
For executive summaries, we compare a cloud LLM API, an on-device small model, and a template-only fallback. Cloud API yields the best quality for SD1 demos; we ship a deterministic fallback if offline.

Part / Product Comparison and Selection
Sensitive fields are redacted prior to summarization; queued emails transmit once connectivity is available.

Table 3.35 LLM/Summary Options

Option	*Pros*	*Cons*	*Fit*
Cloud API	Highest summary quality; no on-device GPU	Requires Internet; cost	**Primary**
On-device small model	Offline; deterministic	Limited quality/speed on CM4	Future/Emerg.
Template-only	Fully offline; predictable	Less natural language	Fallback

Table 3.36 Reporting Pipeline Components

Component	*Options*	*Decision*	*Notes*
HTML templating	Jinja2; Mako; Django Templates	**Jinja2**	Shared with UI
Charts	Chart.js (HTML); Matplotlib	**Chart.js**	Snapshotted in PDF
LLM client	Provider-agnostic REST SDKs	**Adapter**	Switchable at runtime
Mail transport	SMTP; REST Mail API; msmtp	**SMTP on-device**	Swap to API if blocked

3.3.9 Shell Connection Mechanisms

Technology Comparison and Selection
The auditing framework supports multiple shell connection types for post-assessment control: reverse, bind, and tunneled connections.

Table 3.37 Shell Connection Comparison

Shell Type	Transport	Strengths / Weaknesses	Decision
Reverse Shell	TCP / HTTPS	Outbound initiation bypasses firewalls; minimal inbound exposure	**Sel.**
Bind Shell	TCP	Simple setup; inbound port requirement reduces stealth	No
Meterpreter Rev. Shell	HTTPS / Stager	Interactive encrypted sessions via Metasploit payloads	**Sel. (Lab)**
DNS Tunnel Shell	UDP / DNS	Firewall-evasive but high latency, low throughput	Sel.
Sliver Rev. Shell	mTLS/DNS encrypted sessions via Sliver beacon.	Firewall evasion, and fully encrypted but requires C2 infrastructure.	Sel.
SSH	Encrypted interactive bash session, common connection tool	Requires port forwarding to access from outside the network	Alt.

3.3.10 Reverse Shell

The following outlines the various shell technologies supported by the device, along with a comparative analysis of their respective advantages and limitations. Each implementation serves a distinct operational purpose and within the network architecture the device is in.

- **Netcat Reverse Shell:** The Netcat reverse shell provides a lightweight and minimalist way of establishing remote access to the CM4 with negligible software dependencies. It can be instantiated through various interpreters, including `bash`, `sh`, `zsh`, `python`, or `C`, offering considerable flexibility across environments.

  Its primary advantage lies in its simplicity and rapid deployment, making it suitable for short maintenance sessions or environments where minimal configuration is desired. However, these shells provide a limited user interface, lacking advanced terminal features such as tab completion and persistent session management. Although inherently unstable in their basic form, Netcat-based shells can be customized or wrapped with stabilization mechanisms to improve reliability when required.

- **Meterpreter Reverse Shell:** The Meterpreter reverse TCP shell is a more feature-rich option typically utilized in controlled security or penetration testing scenarios. It offers extensive functionality, enabling technicians to upload and download files, collect system information, monitor processes, and perform administrative operations through a unified interactive session.

  The following reverse shell technologies have been selected for implementation within this project.

  Despite its flexibility, Meterpreter shells are easily detected by endpoint protection solutions, as they are frequently associated with security assessment tooling. Unauthorized or unapproved use may trigger antivirus or EDR (Endpoint Detection and Response) quarantines. In addition, the Meterpreter framework requires moderate configuration effort and may necessitate regeneration of payloads if infrastructure parameters (e.g., IP addresses or ports) change.

  The Metasploit framework, in conjunction with `MSFVenom`, simplifies this process by enabling on-demand payload generation, which can be incorporated into maintenance or deployment workflows to ensure operational continuity.

- **Sliver Beacon/Session Shell:** The Sliver beacon shell provides functionality comparable to Meterpreter while integrating modern security and transport mechanisms. By default, Sliver employs mutual Transport Layer Security (mTLS) to provide end-to-end encryption between the CM4 and the technician's command-and-control infrastructure.

  Furthermore, Sliver supports DNS tunneling for its beacon payloads, allowing technicians to register specific, authorized domain names to handle shell communications. This mechanism ensures that only traffic matching authorized domains and payload signatures is accepted, even in the event of IP address changes.

These features make Sliver an attractive option for secure, auditable remote access, particularly in environments where traffic obfuscation and encryption are required to meet operational or compliance constraints.

The key advantage of reverse shell technologies lies in their ability to traverse restrictive network environments. Unlike inbound connections, which require specific firewall or router configurations, a reverse shell initiates an outbound connection from the device to the technician's controlled endpoint. This design eliminates the need to expose inbound ports on the embedded device, reducing potential attack surfaces and simplifying network configuration.

Technicians must maintain appropriate infrastructure to receive these reverse shell connections using one of the supported technologies outlined above (see Table 3.3.10). Operators may also define custom reverse shell tasks for remote management or monitoring, allowing the framework to accommodate organization-specific security and operational policies. This model effectively shifts the network configuration burden to the technician infrastructure, which can be easily redeployed or replaced as needed, while minimizing persistent exposure on the embedded device.

3.3.11 SSH Access

Secure Shell (SSH) is also supported as a conventional and interactive access method for technicians and operators. SSH provides encrypted terminal sessions with both password and public key authentication, making it ideal for scenarios where consistent or extended access to the CM4 is required.

However, unlike reverse shells, SSH relies on inbound connections from the client to the embedded device. This configuration requires the operator to explicitly allow external network traffic to reach the system, thereby assuming responsibility for associated firewall and security configurations. To mitigate these risks, SSH should be reserved for controlled environments or internal maintenance operations.

SSH access requires a valid user account and either a credential pair (username and password) or registered public key, along with an active SSH daemon on the CM4. As SSH is natively included in most Linux distributions, it remains a convenient and well-supported mechanism for authorized system maintenance when network exposure is properly managed.

3.3.12 Physical Connection

As a final method of access, the CM4 supports direct physical connectivity for situations where network-based methods are unavailable or impractical. Preinstalled drivers enable plug-and-play functionality for standard human interface devices such as keyboards, mice, and monitors.

This approach ensures that technicians can still access the system locally in the event of a network failure, configuration error, or security lockout. While physical access is typically

a last-resort option, it provides a guaranteed recovery mechanism for critical maintenance and diagnostic operations.

3.3.13 Data, Logging, Secrets, and Security

Data Storage and Configuration
SQLite stores job metadata, targets, and result indices; large artifacts (XML/JSON) are referenced by path. Device configuration is YAML/JSON (versionable, diffable). A nightly backup packs DB and artifacts into a timestamped archive, optionally exported via SFTP.

Table 3.38 Data Store Comparison

Store	*Pros*	*Cons*	*Use*	*Decision*
SQLite	Zero-admin; ACID	Single-writer	Metadata, indices	**Primary**
JSON/YAML files	Human-readable	No concurrency	Device/config	Sel. (config)
PostgreSQL	Powerful; scalable	Heavy on SBC	Off-site analytics	Future only

Logging and Backups
Journald provides unified logs (persistent storage enabled). Container logs are sidecarred to files only when needed. Rotation caps disk use. A systemd timer drives periodic backups and optional SFTP export.

Secrets and Security Posture
Secrets are injected via `EnvironmentFile` with strict file permissions. We drop Linux capabilities for containerized services, use distinct service users, pin container image digests, and keep inbound network access local-only (UI bound to localhost; the kiosk connects via loopback). The optional reverse shell is outbound-only, TLS-protected, and authenticated.

3.3.14 Traceability to Goals and Hardware Diagram

These selections directly support our objectives and the block-diagram interfaces: (1) standards-based scanning via Nmap/OpenVAS with bounded runtime (CM4→Targets); (2) kiosk-first UX and HTML templates reused for both UI and PDF (Touchscreen↔Apache↔Orchestrator); (3) optional off-site summarization and email export with queuing; and (4) real-time MCU telemetry to the operator (ESP32↔CM4 over SPI). The stack remains lean and reproducible on the CM4.

4 Related Standards and Design Constraints

4.1 Industrial Standards

4.1.1 USB Standards

The Universal Serial Bus (USB) is an industry standard established to universalize data transfer, peripheral connection, and provide power delivery in one interface. Managed by the USB Implementers Forum (USB-IF), it defines protocols, connectors, and electrical specifications that enable those data transfers and power delivery. Compliance with the USB-IF standards ensures cross-platform compatibility, safe power distribution, and data integrity during high-speed communication.

Within the PwnBoxer system, USB serves as a critical interface for both device maintenance and power delivery, providing flexible connectivity for programming, debugging, and external peripheral support. Subsequent sections outline the specific standards adopted in the PwnBoxer design: USB 2.0 for peripheral interfacing and USB Type-C for power delivery, data communication, and interface connections.Mechanically, the USB ports comply with IEC 62680-1-2 for 2.0 connectors and IEC 62680-1-3 for Type-C connectors, ensuring consistent fit and reliability.

USB 2.0 USB 2.0, standardized by the USB Implementers Forum (USB-IF), remains one of the most pervasive specifications in embedded systems. Introduced in 2000, it increased the maximum signaling rate to 480 Mbps through its High-Speed mode while maintaining full backward compatibility with USB 1.1 Low-Speed (1.5 Mbps) and Full-Speed (12 Mbps) modes. Its long-standing presence in the industry makes it a reliable choice for interfacing subsystems where deterministic performance and broad OS-level driver support are required.

From an electrical standpoint, USB 2.0 uses a balanced differential pair (D+ and D-) for data transfer, operating under a 3.3 V signaling environment. The bus supports host-controlled topology, where enumeration, device addressing, and power distribution originate from the host port. USB 2.0 also specifies strict requirements on impedance matching (typically 90 Ohm differential), cable length, connector durability, and ESD tolerance. These characteristics ensure signal integrity across typical embedded and consumer device use cases.

In the PwnBoxer design, two USB-A/USB-2.0 host ports are implemented to provide service access and facilitate peripheral expansion. These ports allow technicians to interface with the device using standard programming cables, USB-to-UART adapters, or mass storage devices for firmware updates. Their inclusion ensures compatibility with legacy debugging workflows, and the low-profile electrical requirements keep the hardware footprint minimal. USB 2.0 is also advantageous for this application due to its mature embedded-Linux support on the Raspberry Pi CM4 platform, standardized driver stack, and predictable behavior under continuous or intermittent service usage.

While USB 3.x and other high-speed standards offer greater data throughput, such per-

formance is unnecessary for the PwnBoxer's maintenance-focused tasks, where reliability, simplicity, and universal host compatibility take precedence. Additionally, the Raspberry Pi Compute Module 4 does not have native support for USB 3.0, unless wiring it through the PCI express lane. [28]

USB Type-C The USB Type-C standard, defined in IEC 62680-1-3 and introduced initially by the USB-IF in 2014, represents a major evolution of the USB physical interface. Type-C replaces earlier connector styles with a compact, reversible 24-pin design that supports multiple protocols through a single port. This connector is mechanically symmetrical and can be inserted in either orientation. It also supports high insertion-cycle durability, with a typical rating of up to 10,000 mating cycles. The Type-C interface is designed to carry data, power, and optional alternate-mode signals such as DisplayPort and USB 3.x, while still maintaining backward compatibility with USB 2.0 signaling.

From an electrical perspective, Type-C introduces the Configuration Channel (CC) pins, which are used for cable orientation detection, power-role identification, and current-advertisement signaling. These pins enable devices to determine whether they should act as a power source, a power sink, or a dual-role device. Even without the optional USB Power Delivery (USB-PD) protocol, Type-C ports can advertise default current levels of 500 mA for legacy USB 2.0 hosts or 1.5 A and 3.0 A for dedicated charging ports. When USB-PD is supported, devices can negotiate significantly higher power levels to meet various system requirements. All of this behavior depends on precise electrical characteristics, such as termination resistances on CC pins, stable VBUS ramp-up characteristics, and strict adherence to voltage limits that protect connected devices.

In the PwnBoxer system, a single USB Type-C port is implemented, and it is dedicated exclusively to providing power to the device. Using Type-C for power input offers significant advantages, including improved connector durability, stable electrical performance, and compatibility with modern USB-C power adapters. The port is designed to follow Type-C current-advertisement rules to ensure that it negotiates safe operating conditions with compliant chargers. Protection circuitry manages inrush current, prevents reverse-current flow, and ensures proper VBUS regulation, which protects the internal electronics from voltage anomalies or incorrect cable insertion. This approach provides a reliable and service-friendly power interface that supports long-term operation in embedded environments where the device may experience frequent handling or varying power sources.

The addition of Type-C throughout the design improves user experience, simplifies cable management, and ensures compatibility with modern development tools and charging equipment. The connector's mechanical resilience and well-defined electrical behavior support reliable operation in long-term embedded deployments, which aligns with the robustness requirements of the PwnBoxer system. [29]

4.1.2 PCB Design Standards

The Printed Circuit Board (PCB) is the structural and electrical foundation of the PwnBoxer hardware. It mechanically supports and electrically connects components through conduc-

tive copper traces laminated on a non-conductive substrate. The PCB design must meet signal integrity, electromagnetic compatibility (EMC), thermal performance, and manufacturability standards to ensure reliable operation under various environmental conditions.

The PwnBoxer's PCB integrates multiple subsystems, adhering to modern PCB design standards ensures efficient routing, consistent impedance control, and compliance with international safety and reliability benchmarks.

IPC Standards (IPC-2221, IPC-7351, IPC-A-600, IPC-A-610)
The Institute for Printed Circuits (IPC) defines global standards for PCB design, fabrication, and assembly. These standards establish the foundation for quality assurance, manufacturability, and consistency across the electronics industry.

- **IPC-2221:** IPC-2221 is the general PCB design standard that applies to most board technologies. It defines the core rules for trace width, conductor spacing, dielectric thickness, current-carrying capacity, and insulation requirements. These specifications determine the board's ability to maintain signal integrity, handle electrical load, and operate safely under different conditions. The standard also covers material tolerances and mechanical characteristics such as copper thickness and via dimensions. By following IPC-2221, the design gains a stable and predictable foundation for electrical performance, manufacturability, and long-term reliability. [30]
- **IPC: 7351:** IPC-7351 focuses on surface-mount land pattern design. It establishes standardized dimensions for pads, courtyards, and solder mask openings that correspond to a wide range of component package types. These guidelines support reliable solder joints, reduce assembly defects, and improve compatibility with automated manufacturing equipment. Using IPC-7351 land-pattern libraries helps prevent issues such as tombstoning, insufficient solder coverage, or component misalignment. For the PwnBoxer, adherence to this standard ensures that surface-mount components can be placed and soldered consistently across multiple production runs.[31]
- **IPC-A-600/A-610:** IPC-A-600 defines the acceptability criteria for bare PCBs, including internal and external features such as plating quality, laminate defects, annular rings, and drilled hole accuracy. It categorizes boards into three quality levels (Class 1, 2, and 3), allowing designers to specify inspection requirements based on how critical reliability is for the final product. IPC-A-610 applies similar acceptability criteria to fully assembled PCBs. It covers workmanship quality, solder joint integrity, component placement accuracy, and cleanliness requirements. These criteria ensure that the assembled hardware meets the level of reliability required for embedded systems deployed in real-world environments. [32]

By following IPC design and manufacturing guidelines, the PwnBoxer PCB minimizes risk associated with fabrication tolerances, solderability defects, and reflow inconsistencies.

RoHS and REACH Compliance
The Restriction of Hazardous Substances (RoHS) Directive and Registration, Evaluation, Authorization, and Restriction of Chemicals (REACH) regulations restrict the use of specific hazardous materials (e.g., lead, mercury, cadmium) in electronic assemblies. Compli-

ance impacts component sourcing, solder material selection, and manufacturing processes.

RoHS and REACH-compliant parts are prioritized to ensure the design meets environmental and safety standards, supporting potential future commercial deployment or certification.

Signal Integrity and EMC
Signal integrity (SI) and electromagnetic compatibility (EMC) are performance factors in mixed-signal embedded systems like PwnBoxer.High-speed interfaces are sensitive to noise, crosstalk, and impedance mismatches. Requiring proper trace impedance control, ground planes, and differential pair routing to meet EMC regulations.

4.1.3 Ethernet Standards

Ethernet communication forms the foundation of modern local area networks (LANs) and is governed by the IEEE 802.3 family of standards. It defines the physical and data link layers for wired network communication, enabling reliable and high-speed data transfer between devices. The 10/100/1000 Mbps Ethernet interface (Gigabit Ethernet) specifically provides the data throughput and stability required for network auditing systems that handle real-time data capture and analysis.

In the proposed design, Ethernet is not only used for transferring scan results and logs but also serves as a communication bridge between the embedded device and external clients or technician dashboards. The use of standard RJ45 connectors and CAT5e/CAT6 cables ensures compatibility with existing network infrastructures, while the IEEE 802.3ab specification guarantees gigabit-level performance with low latency and minimal packet loss.

Implementation and Supported Features
The selected Raspberry Pi Compute Module 4 (CM4) integrates the Broadcom BCM54210PE Ethernet PHY, which provides Gigabit Ethernet (1000BASE-T) connectivity compliant with IEEE 802.3ab. This PHY supports advanced features such as auto-negotiation, MDI/MDI-X crossover detection, and low-power idle (LPI) modes, enabling efficient network operation and reduced power consumption.

The hardware design maintains compliance with ANSI TIA/EIA-568-B differential impedance standards to ensure high signal integrity over twisted-pair cables. Electrical isolation is achieved through transformer coupling per IEEE 802.3 Clause 33 and IEC 60950-1 safety standards, protecting the system from surges and ground potential differences.

Ethernet connectivity plays a crucial role in the PwnBoxer system's ability to perform wired network vulnerability assessments, automated packet analysis, and active scanning. The CM4's Ethernet interface allows high-bandwidth data exchange while maintaining deterministic communication.

IEEE 1588 Precision Time Protocol (PTP)
The CM4's integrated PHY includes support for IEEE 1588 Version 2, the Precision Time Protocol (PTP), which enables sub-microsecond synchronization accuracy across connected

network devices. This capability is particularly valuable for distributed network auditing setups, where multiple embedded nodes may need to coordinate data collection or inject network traffic at precise time intervals.

PTP operates by exchanging timestamped synchronization packets and compensating for propagation delays in the network, resulting in highly accurate clock alignment between master and slave devices. The CM4's hardware-level timestamping implementation offers superior precision compared to software-only solutions, ensuring more reliable time coordination in latency measurements, event correlation, and controlled penetration testing scenarios.

System Integration and Compliance
By combining Ethernet communication and IEEE 1588 synchronization, the system achieves a high level of integration, interoperability, and reliability. Compliance with IEEE 802.3, 802.3ab, 802.3af/at, and 1588 standards ensures that the hardware can interface with enterprise-grade networking equipment without compatibility issues.

The use of Gigabit Ethernet guarantees sufficient throughput for continuous packet analysis and network mapping tasks, while PoE provides a convenient and safe method of powering the device in environments where access to traditional power outlets is limited. IEEE 1588 support further enhances performance in multi-node or distributed applications, ensuring consistent time-stamped data collection and synchronized test execution.

Together, these features establish the Ethernet subsystem as a core standard of the embedded network auditing device — delivering robust connectivity, precision timing, and flexible power management suited for professional network analysis and penetration testing deployments.

4.1.4 HDMI Standards

Applicable Specs
HDMI v1.4/2.0 signaling (TMDS), 5 V Hot-Plug Detect (HPD), Display Data Channel (DDC; I^2C @ 100 kHz), and optional Consumer Electronics Control (CEC; single-wire). Use 100 differential impedance for TMDS pairs, maintain tight intra-pair skew (< 15 ps/in typical), and observe cable length limits for 1080p/60 (category 2/"high-speed"). ESD protection is required on all external pins.

Table 4.1.6 HDMI Electrical/Protocol Summary

Signal Group	*Level/Imped.*	*Key Requirements*	*Design Notes*	*Compliance*
TMDS data/clock (4 diff. pairs)	AC-coupled, 100 diff.	250 MHz–1.65/3.4 Gb/s/ch (v1.4/2.0)	Length-match within pair; minimize stubs; short, direct escape; no vias if possible	Eye margin, jitter, 100 diff.
DDC (SCL/SDA)	5 V tolerant, open-drain	100 kHz; ˜47 k pull-ups to 5 V on sink	Keep away from TMDS pairs; ESD diodes to 5 V/GND	EDID readable, NACK-free
HPD	5 V input to source	Edge-filtered; debounce	ESD to chassis; series R (100–1k)	Detects connect/disconnect
CEC	˜3.3 V idle, open-drain	Single-wire, ˜3.3 k pull-up	Route separately; strong ESD; firmware optional	Optional
Shield/ground	Chassis bonded	Low inductance return	Stitch vias near connector	EMC

Layout/Protection Guidance
Place an HDMI ESD array (low C_{diff} on TMDS) at the connector. Enforce 100 diff. with length-matching and avoid stubs via direct breakout or back-drilled vias. Keep DDC/CEC away from TMDS pairs. Provide clean 5 V for HPD and EDID. This meets the "industrial standards" intent shown in Chapter 4's outline.

4.2 Communication Standards

4.2.1 I2C

Role in System
Board-local control/status lines (e.g., touch controller, GPIO expanders). Short runs only; not used between boards.

Table 4.2.1 I^2C Electrical/Protocol Requirements

Mode	*Rate*	*Voltage/Levels*	*Pull-ups / Timing*	*Addressing*
Standard/Fast	100/400 kHz (target)	3.3 V open-drain (5 V tolerant parts where needed)	2.2–4.7 k to 3.3 V; rise time $<$ 300 ns (400 kHz)	7-bit; reserve 0x00/0x7F
Fast-mode+ (optional)	1 MHz (if devices allow)	Same	1–2.2 k; trace $<$ 10 cm; low bus C	Same

Layout/Protection Guidance
Keep bus length short ($\leq$10-15 cm), star-avoid; place pull-ups near the master. Add 33-47 series dampers at SCL/SDA if needed. Reserve addresses; document bus scan in bring-up. ESD arrays on off-board headers if ever exposed.

4.2.2 UART Standards

Role in System
Service console and potential debug link. This is 3.3 V TTL UART on-board; RS-232/EIA-232 voltage levels are *not* used unless an external adapter is inserted.

Table 4.2.2 UART Electrical/Protocol Requirements

Framing	*Baud*	*Levels*	*Flow/Connector*	*Use*
8N1 default (configurable)	115,200 (min) to 921,600 bps (max)	3.3 V TTL	Optional RTS/CTS; 0.1" header or USB-UART dongle	Console/debug

Layout/Protection Guidance
Keep TX/RX short; include a test header. If exposing externally, add ESD diodes and consider series 33-68 resistors. Document that RS-232 $\pm$12 V levels require a transceiver (e.g., MAX3232).

4.2.3 SPI Standards

Role in System
Primary high-speed link between CM4 and ESP32 for status/telemetry and commands (see Chapter 3 interface table).

Table 4.2.3 SPI Electrical/Protocol Requirements

Mode	*CLK*	*Pol/Phase*	*Wiring*	*Notes*
Single-lane	8–16 MHz target	Mode 0 (CPOL=0, CPHA=0)	SCLK, MOSI, MISO, CS; 3.3 V	CS low; MSB first; fixed header+JSON framing
Signal integrity	—	—	22-33 series on SCLK/MOSI/MISO; 100 SCLK ref	Keep traces <20 cm; match lengths; solid return plane
Reliability	—	—	Watchdog+CRC in payload	Retries; bounded queue; error counters

Layout/Protection Guidance
Place master and slave close; route with continuous return path; avoid T-stubs; provide a spare CS for future peripherals. If cabled, use short, twisted pairs for SCLK/MOSI and SCLK/MISO.

4.3 Design Constrains

4.3.1 Economic

Tariffs and Supply Chain Challenges
Global electronics manufacturing continues to face significant cost pressures due to tariffs, shipping delays, and component scarcity. Tariff fluctuations on imported semiconductors, copper-clad laminates, and passive components directly influence the per-unit cost of production. Because of this, design teams often need to adjust their sourcing strategies and reevaluate which components are cost-effective. In some cases, rising duties on essential materials can shift entire design choices, and teams may begin to prefer integrated modules instead of discrete parts to reduce exposure to unstable pricing. These financial uncertainties create unavoidable budgeting risks and require engineers to plan for flexibility in procurement.

International shipping costs have also increased sharply since 2020, and this rise is caused by logistics bottlenecks, port congestion, and broad inflationary trends. Lead times for microcontrollers, memory devices, and wireless chipsets have become unpredictable, sometimes delaying development cycles for long periods. As a result, projects increasingly rely on domestically available or tariff-exempt components to maintain predictable schedules. It is also common to establish relationships with multiple suppliers so that the project is not dependent on a single source. This approach helps reduce vulnerability to unexpected shortages or price spikes.

Project Budget
Budget constraints shape the practical limits of component selection and manufacturing methods. These limitations affect system architecture, performance requirements, and even the number of features that can be included. Because the goal is to produce a commercially viable prototype, each subsystem must be assessed not only for technical capability but also for its contribution to overall cost. This assessment often leads to design trade-offs. For example, a team may choose to simplify the PCB stack-up, rely on mixed-function ICs, or eliminate nonessential features in order to keep the design affordable.

The bill of materials is reviewed throughout the development process to identify opportunities for cost reduction. This may include taking advantage of volume pricing, consolidating parts, or selecting components that are widely available in the market. Using open-source hardware standards can further reduce development costs, since these platforms often have mature ecosystems and competitive pricing. Recurring manufacturing expenses such as assembly, testing, and inspection are also considered early to avoid unexpected financial impacts in later stages. By continually balancing capability and cost efficiency, the design remains scalable for low- and mid-volume production.

Economic Viability and Scalability
Long-term economic sustainability guides many of the core design decisions. A successful product must meet performance goals while also remaining cost-effective and maintainable during its lifespan. Planning for this outcome requires awareness of component lifecycles, expected market changes, and overall sourcing stability. Teams often rely on standard footprints and modular subsystems because these choices reduce the likelihood of major redesigns in the future. If a component becomes obsolete, it is much easier to substitute an equivalent part without altering the entire board.

Scalability is another critical factor, especially for designs that are intended to transition from prototypes into commercial products. Early attention to regulatory compliance, including standards such as CE or FCC, helps prevent costly redesigns later in the process. Strategic component selection ensures that future revisions of the device can incorporate updated processors, interfaces, or wireless technologies without significant engineering effort. This adaptability reduces long-term development costs and helps the product remain competitive in a fast-moving market.

4.3.2 Technical

Thermal Management
Thermal considerations play a significant role in maintaining device reliability and performance, particularly in compact embedded systems where multiple high-power components operate in close proximity. The PwnBoxer's high-density PCB layout and its inclusion of processing modules such as the Raspberry Pi Compute Module 4 (CM4) result in concentrated heat generation during sustained workloads. Without adequate thermal mitigation, this heat can lead to thermal throttling, reduced operating efficiency, and shortened component lifespan. To address this, the design incorporates copper pour regions that act as heat spreaders, allowing heat to disperse more evenly across the PCB. Thermal vias beneath

power-intensive components further enhance vertical heat conduction, reducing hotspots and improving dissipation through adjacent layers.

In addition to conductive thermal strategies, passive cooling elements are evaluated to maintain stable temperatures under peak load. A small passive heatsink increases surface area for better convective cooling, and its use is balanced against spatial constraints and airflow availability. PCB material selection also contributes to thermal stability; for instance, choosing an FR-4 substrate with a glass transition temperature (Tg) above 170°C helps ensure the board can withstand thermal cycling and prolonged elevated temperatures. Component spacing, orientation, and placement near thermal relief areas are likewise considered to ensure that the system operates within safe temperature margins during continuous operation.

Power Distribution

Power management must ensure stable operation across all peripherals, communication channels, and processing modules. The system relies on multiple voltage domains, typically regulated 5V and 3.3V rails, which supply logic devices, wireless modules, and sensors. Ensuring low-noise regulation across these rails is critical, as voltage ripple or brownout events can cause system instability or data corruption. Proper decoupling networks, bulk capacitors, and distribution trace widths are incorporated to maintain reliable current delivery during peak demand. Additionally, transient response and inrush current control are evaluated, especially when peripherals are hot-plugged or rapidly enabled.

The inclusion of Power over Ethernet (PoE) further enhances the device's deployment flexibility but adds notable design complexity. PoE circuits must satisfy electrical isolation requirements, current-handling specifications, and thermal considerations associated with DC-DC conversion. Careful selection of inductors, switching regulators, and protection components helps ensure compliance with IEEE 802.3 standards. USB-C also serves as a key power and data interface, and its implementation requires adherence to USB Power Delivery (PD) negotiation rules to prevent overcurrent or overvoltage conditions. Proper cable-detection, orientation sensing, and current-limiting circuitry protect both the PwnBoxer and connected devices during operation.

Spacial Limitations

Physical constraints strongly influence the overall board layout, including its dimensions, connector placement, and mechanical mounting strategies. The device must maintain a compact footprint while integrating multiple high-function interfaces such as USB ports, an HDMI output, and Ethernet hardware. This requires strategic arrangement of components to ensure adequate clearance, signal routing feasibility, and accessibility. High-pin-count connectors, antennas, or shielding structures introduce additional challenges and demand careful coordination between electrical and mechanical design requirements.

Mechanical durability is also prioritized to ensure the device can withstand handling, vibration, and repeated cable insertions common in development or field environments. Reinforced connector pads, mounting holes, and support structures help prevent mechanical stress from damaging the PCB. Proper connector spacing and ergonomic port orientation

improve user accessibility during programming, debugging, and maintenance. These considerations ensure that despite the compact form factor, the PwnBoxer remains both serviceable and robust in diverse operational scenarios.

4.3.3 Flexibility

One of the key design constraints, as defined by the project sponsor, is flexibility, specifically, the ability to easily create, modify, and deploy new testing tasks without requiring software changes to the system. This constraint ensures that developers and technicians can continuously expand the system's testing capabilities as new penetration testing tools or workflows are introduced.

To address this requirement, the system implements a YAML-based task configuration schema that defines a standardized and extensible interface for task creation. Each task is represented by a configuration file that specifies essential parameters such as the task name, executable path, output location, and command-line arguments, as well as optional metadata like interpreter paths and environment variables. This schema abstracts away implementation details, allowing users to introduce new functionality through configuration rather than code.

During initialization, the CM4 enumerates all task configuration files located within the `user-tasks/` directory, validating their structure and registering them as available tasks. This registry is then synchronized with the ESP32, which populates its user interface with the updated list of available tasks.

By decoupling task definitions from the core application logic, we addressed the flexibility constraint and can enable rapid deployment new workflows. Technicians can define and deploy custom scripts or executables in any supported runtime environment on the CM4, while the ESP32 automatically reflects those additions through the shared task registry.

4.3.4 Networking

Another critical design constraint is networking, which directly affects both the functionality of the device and the ability for technicians to access it for maintenance or updates. Because the system is primarily intended for deployment within small-business environments, it must be designed to operate reliably within segmented networks and under restrictive firewall and NAT configurations. These environments often employ network segmentation, VLANs, and strict access control policies to isolate internal resources, as well as NAT or firewall rules that block unsolicited inbound connections. Such configurations, while essential for security, pose challenges for maintaining communication with the device and executing network-based testing operations.

Network segmentation is a foundational practice in secure network design, ensuring that sensitive systems are isolated and limiting lateral movement in the event of a compromise. However, this segmentation introduces operational limitations for devices that require visibility across multiple network segments. A single network interface may not be sufficient to reach all relevant zones, particularly when some are separated by physical or logical

boundaries such as dedicated Ethernet VLANs or isolated WiFi networks. To address this limitation, the device incorporates multiple network interfaces,specifically both WiFi and Ethernet to provide multi-segment connectivity. This design allows the device to connect to multiple network interfaces at once, such as connecting to Ethernet while maintaining communication with a management network via WiFi. This dual-interface approach leverages the CM4s network capabilities and removes the need for constant reconfiguration of the device.

Another challenge arises from NAT and firewall restrictions, which often block inbound traffic. When port forwarding rules are not in place SSH infeasible and on embedded systems that are not always monitored this poses an additional risk. To overcome this obstacle, the system employs a reverse-connection approach, where the device initiates outbound connections to a preconfigured remote listener under technician control. This method bypasses the NAT and firewall restrictions as connections initiated internally are allowed bi-directionally through routing equipment.

4.3.5 Time

Time has been a major constraint that shaped the overall scope of the project. With only two semesters to plan and develop the system, the original concept had to be significantly reduced to match the available development time. This required prioritization of core functionality and the re-scoping of additional features. One major area affected by these reductions was the implementation of authentication measures for the reverse shell. While the original design included an authentication step to prevent unintended connections, this feature was moved to a stretch goal since the current implementation already requires two parties: one initiating the connection and another listening to establish a session.

To manage time effectively, we purchased essential hardware components early and began PCB design and system architecture development as soon as possible. We ensured that the scope remained realistic by setting clear milestones and working every week towards the completion of a task. Regular communication with our sponsor helped confirm that the adjusted objectives continued to align with the overall vision for the project.

5 Application of ChatGPT and Other Similar Platforms

5.1 Overview

The integration of generative artificial intelligence (AI) tools such as ChatGPT has significantly accelerated the research, documentation, and development workflow of the PwnBoxer project. Throughout the course of Senior Design I, ChatGPT was used as a collaborative assistant to refine technical writing, verify hardware–software integration choices, generate LaTeX elements such as tables and figure captions, and debug development issues encountered during prototyping. While human engineering judgment remained central to all decisions, the AI provided immediate context-aware feedback that reduced redundant work and improved overall report consistency.

ChatGPT and similar models were used in conjunction with traditional sources such as manufacturer datasheets, research articles, and reference schematics. In this capacity, AI tools acted as an intelligent intermediary, helping team members convert raw technical data into formatted, concise documentation compatible with the strict reporting and formatting expectations of the Senior Design program.

5.2 AI Prompt Engineering and Workflow Strategy

Effective use of generative AI required careful prompt design and validation strategies. The PwnBoxer team developed a structured workflow for interacting with ChatGPT to ensure clarity, reproducibility, and traceability of results. Prompts were framed as engineering tasks rather than open-ended questions, emphasizing context and desired output formats. For example, requests for table generation explicitly included column headers, caption style, and alignment parameters to conform to UCF Senior Design formatting standards.

When drafting technical text, the model was used iteratively: an initial version was generated, refined through targeted feedback, and finally validated by a human team member. This process ensured that every AI-assisted paragraph retained accurate engineering meaning while benefitting from consistent style and grammar. Prompt chains were stored in version-controlled notes, allowing the team to reproduce specific outputs or revert to earlier drafts as needed.

This disciplined workflow not only improved writing efficiency but also introduced us to an emerging professional skill-prompt engineering-which parallels traditional software design practices such as modularization and version control.

5.3 Benefits and Educational Impact

The adoption of ChatGPT introduced measurable improvements in efficiency and clarity across several aspects of the design process:

- **Documentation Assistance:** ChatGPT generated properly formatted LaTeX tables for each technology comparison, following the uniform captioning, column alignment, and resize conventions required by UCF Senior Design guidelines. This substantially reduced the time spent on manual layout correction.
- **Technical Clarification:** During component selection and software integration, ChatGPT clarified complex specifications such as SPI edge-timing, LVGL rendering pipelines, and current-draw budgeting for multi-device power rails.
- **Iterative Editing:** The model provided real-time grammar, tone, and flow feedback to maintain consistency across multi-author sections. This ensured that the final document reads as though written by a single author, as mandated by the project guidelines
- **Research Acceleration:** When cross-checking open-source firmware or comparing display controller libraries, ChatGPT summarized external documentation into engineering-focused bullet points, reducing context-switching overhead for the team.

These capabilities complemented traditional teamwork by letting members focus on design logic, testing, and implementation rather than formatting or low-level syntax corrections.

5.4 Limitations and Ethical Considerations

While AI support offered clear advantages, several limitations and precautions were acknowledged:

- **Dependence and Accuracy:** ChatGPT's responses can occasionally include outdated or fabricated references. All generated technical content was manually verified using datasheets or academic sources before inclusion in the final report.
- **Source Attribution:** In compliance with UCF academic policy, every instance of AI-assisted writing is cited in Appendix E with the corresponding prompt and output transcript. This ensures transparency and prevents unintentional plagiarism.
- **Engineering Judgment:** AI suggestions never replaced human verification. Circuit selection, power budgets, and algorithmic decisions were validated through empirical testing and simulation before acceptance.
- **Security and Privacy:** Proprietary schematics and sponsor-specific information were paraphrased or abstracted before being discussed with AI systems to maintain confidentiality.

5.5 Ethical and Responsible Use Statement

All AI-assisted content within this report was created in compliance with UCF's academic integrity policies. ChatGPT and related platforms were treated as productivity and learning tools, not as substitutes for human reasoning or original design effort. Each team member verified all generated information before inclusion, ensuring that final content reflected independently confirmed facts and engineering judgment.

No confidential data, proprietary code, or personally identifiable information was shared with AI systems. All technical descriptions, schematics, and tables generated by ChatGPT were derived from publicly available knowledge or pre-approved project data. Where AI contributions affected final report text, authors maintained full editorial control to guarantee factual accuracy, ethical transparency, and alignment with course objectives.

5.6 Examples of AI-Assisted Development

The following examples demonstrate how ChatGPT directly contributed to the engineering workflow of PwnBoxer.

5.6.1 Example 1: LaTeX Table Generation for Technology Comparison

Early drafts of Chapter 3 suffered from table formatting inconsistencies due to mixed use of `tabularx` and fixed-width column definitions. ChatGPT was used to rewrite all com-

parison tables into the unified `\resizebox` structure adopted across the report. This automation standardized fonts, borders, and caption styles, resolving column overflow errors and eliminating manual alignment work.

Quote from AI Output:

```
\resizebox{\linewidth}{!}{%
\begin{tabular}{|C{1.4}|C{1.0}|C{1.0}|C{2.0}|C{0.4}|}
... \end{tabular}}
```

This snippet became the standardized table template for all subsequent hardware and software comparisons.

5.6.2 Example 2: Firmware Debugging Assistance

During early integration of the YOLOv11 + DeepSORT tracking code on the CM4, a floating-point precision mismatch caused unstable bounding-box rendering. ChatGPT was used to identify the parameter inconsistency between `torch.float16` and `numpy.float32` tensor conversions. This helped resolve frame corruption without requiring full environment reinstallation.

5.6.3 Example 3: Technical Writing and Flow Improvement

Each subsystem section-such as *Display Subsystem* and *Communication Protocols*-underwent language refinement through AI-based iteration. ChatGPT transformed bullet notes into cohesive paragraphs, adjusted tone to match academic style, and added transitions linking one subsection to another. This process helped maintain logical flow across sections authored by different team members.

5.6.4 Example 4: Reverse Shell Section Formatting

A team member provided raw text explaining several reverse shell implementations. ChatGPT reformatted the material into LaTeX with itemized descriptions, unified terminology, and concise comparative analysis between Netcat, Meterpreter, and Sliver shells. The AI also added an explanatory paragraph connecting this mechanism to the CM4-ESP32 task scheduler, ensuring the section matched the report's narrative and stylistic standards.

5.7 Case Study: AI-Assisted Code Optimization and Documentation

One of the most impactful demonstrations of AI's utility within the PwnBoxer project occurred during the integration of the ESP32 display firmware with the CM4 task scheduler. This process initially suffered from frame latency issues and inconsistent update rates due to inefficient buffer management and repeated blocking calls within the SPI display driver. The team leveraged ChatGPT to analyze the driver's logic flow and suggest optimization strategies based on DMA buffering and task yield intervals.

Through iterative prompting, the model proposed refactoring the draw routine to implement a double-buffer queue with conditional refresh triggers. The suggested approach closely

matched techniques used in established embedded GUI pipelines, such as LVGL's internal flushing model. After implementing these changes, the ESP32's frame latency decreased from approximately 180 ms to 90 ms under peak load, and rendering stability improved significantly.

Beyond performance gains, the team also relied on ChatGPT to draft inline documentation and function-level comments for maintainability. The AI generated clear, consistent docstrings explaining task priorities, SPI bus arbitration, and buffer synchronization. This improved readability and compliance with the project's internal coding standards.

A second example involved reformatting raw test logs and scan results into publication-quality tables for the final report. ChatGPT's precision in generating complex LaTeX table syntax enabled rapid conversion of CSV logs into formatted comparison tables using the `\resizebox` and `C{}` column alignment system standardized throughout the report. What would have required hours of manual column calibration was completed in minutes, maintaining both consistency and professional layout quality.

This case study exemplifies how conversational AI, when used responsibly, can serve as a technical collaborator-bridging the gap between raw engineering data and structured, high-quality documentation. Rather than replacing the engineer's reasoning, it acts as a precision tool that enhances productivity while preserving academic integrity and rigorous verification standards.

5.8 Comparison of AI Tools

Although ChatGPT was the primary tool, alternative platforms were briefly evaluated for completeness. Table 5.1 summarizes their observed strengths and limitations.

Table 5.1 AI Platform Comparison

Platform	*Strengths*	*Limitations*	*Use Cases in Project*	*Decision*
ChatGPT (OpenAI GPT-5)	Context-aware writing; supports LaTeX formatting; robust technical reasoning	Occasional hallucinations; requires verification	Technical writing, debugging, documentation, research summarization	**Selected**
Google Gemini Advanced	Fast factual recall; web search integration	Limited LaTeX handling; weaker long-context coherence	Quick fact-checking and image description	Alt.
Claude 3.5 (Anthropic)	Excellent flow and tone consistency; citation tracking	Weaker code generation than GPT-based models	Proofreading and ethics review drafts	Support
GitHub Copilot Chat	Tight IDE integration; code completion within VS Code	Not optimized for academic prose; limited context window	Firmware refactoring and syntax debugging	Support

5.9 Observations on AI in Engineering Education

From an academic standpoint, the use of AI during Senior Design encourages a hybrid learning model that merges technical execution with rapid iterative feedback. The team found that conversational prompting developed an understanding of why design choices worked, not just how to implement them. For instance, when evaluating display driver options or comparing communication protocols, the model's reasoning served as a secondary reviewer that challenged initial assumptions.

AI systems like ChatGPT also demonstrated their ability to serve as educational multipliers-offering alternative explanations for technical material, summarizing complex documentation, and generating clear visuals or structured text when time was limited. Importantly, they fostered the same critical thinking skills expected from traditional mentorship: assessing reliability, validating sources, and translating general knowledge into domain-specific design decisions.

5.10 Future Role of AI in Senior Design

Looking forward, the role of AI in academic design projects is expected to deepen. Generative tools may soon support automated schematic generation, PCB layout drafting, or even real-time debugging via natural-language integration within IDEs. The PwnBoxer project demonstrates that, when used responsibly, these systems can serve as force multipliers-augmenting but never replacing human creativity and engineering judgment.

The UCF Senior Design framework benefits from this duality: students gain exposure to professional-grade automation while maintaining accountability for all engineering choices. With appropriate citation and ethical oversight, AI can continue to accelerate design innovation and reduce the friction between idea and implementation.

5.11 Conclusion

ChatGPT served as an invaluable multipurpose assistant throughout the development of the PwnBoxer system, enhancing efficiency, precision, and cohesion in both technical and written outputs. While limitations such as occasional factual errors require cautious review, the net educational benefit-particularly in technical writing, LaTeX formatting, and research synthesis-was undeniable. The PwnBoxer team acknowledges that AI tools should supplement, not supplant, the engineer's critical thought process. Used ethically and transparently, they represent one of the most transformative learning aids currently available to the engineering education ecosystem.

5.12 Future Outlook

As AI models continue to improve, their integration into engineering education will become increasingly seamless. Future iterations of ChatGPT and similar systems may support code-aware debugging, embedded design simulation, or automated LaTeX diagram generation directly from schematics. The PwnBoxer project illustrates how these technologies can augment-not replace-critical thinking. Used responsibly, AI-driven tools can accelerate design iteration, improve documentation quality, and equip future engineers with the skills necessary to thrive in an AI-enhanced workplace.

6 Hardware Design

6.1 Overview

The hardware design of the PwnBoxer system establishes a modular and reliable embedded platform for network auditing and vulnerability assessment. Its primary purpose is to integrate processing, communication, and control subsystems into a unified, compact, and scalable architecture. The design enables seamless operation of automated penetration testing routines while maintaining expandability for future firmware or hardware enhancements.

This system combines the computational capabilities of the Raspberry Pi Compute Module 4 with the peripheral control efficiency of the ESP32-S3, creating a balanced architecture that supports both high-speed data analysis and responsive peripheral interaction. The hardware foundation is engineered to handle the simultaneous tasks of packet capture, signal monitoring, and data reporting within the constraints of power efficiency, reliability, and affordability.

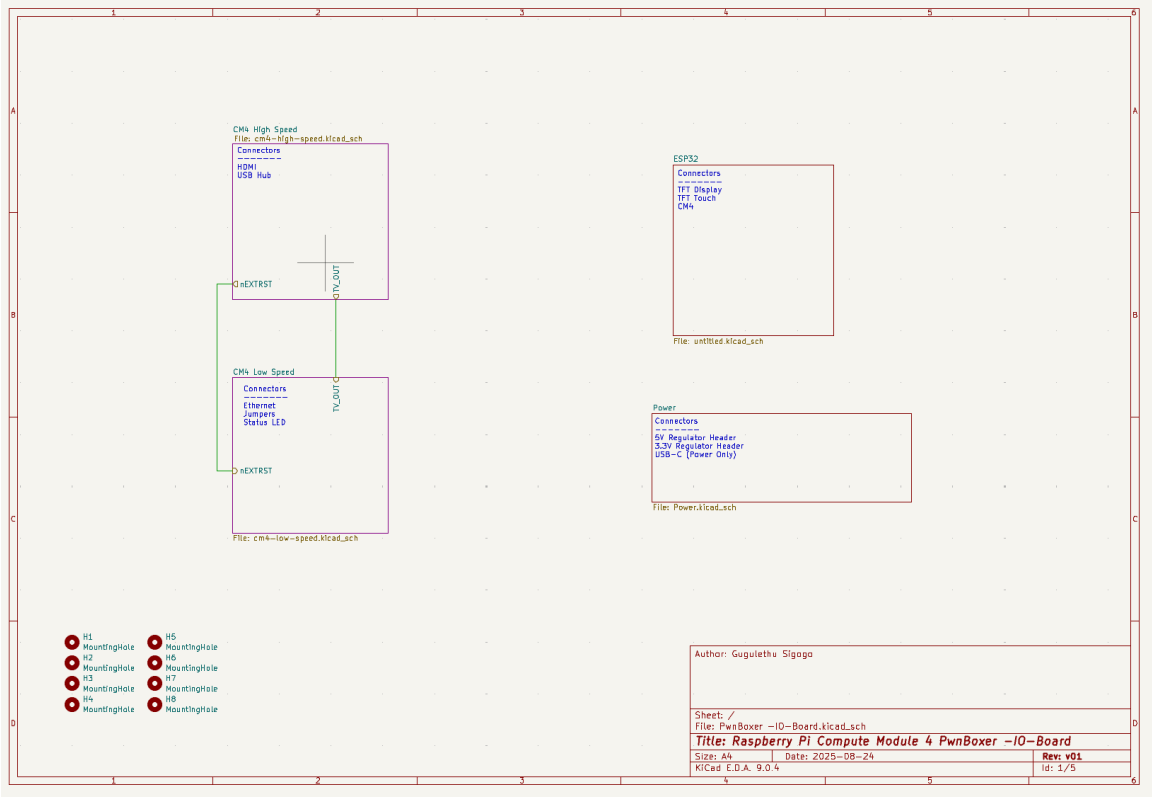


Figure 5: PCB Overview showing Power, CM4, ESP32, and TFT Display subsystems - KiCad

6.2 System Architecture

The overall system architecture of the PwnBoxer project is composed of four primary subsystems: the Power Delivery Subsystem, the CM4 Subsystem, the ESP32-S3 Subsystem, and the TFT Display Subsystem. Each subsystem performs a distinct role in maintaining the system's operation, communication, and user interface functionality. Figure 6 illustrates the high-level structure and interaction between these components.

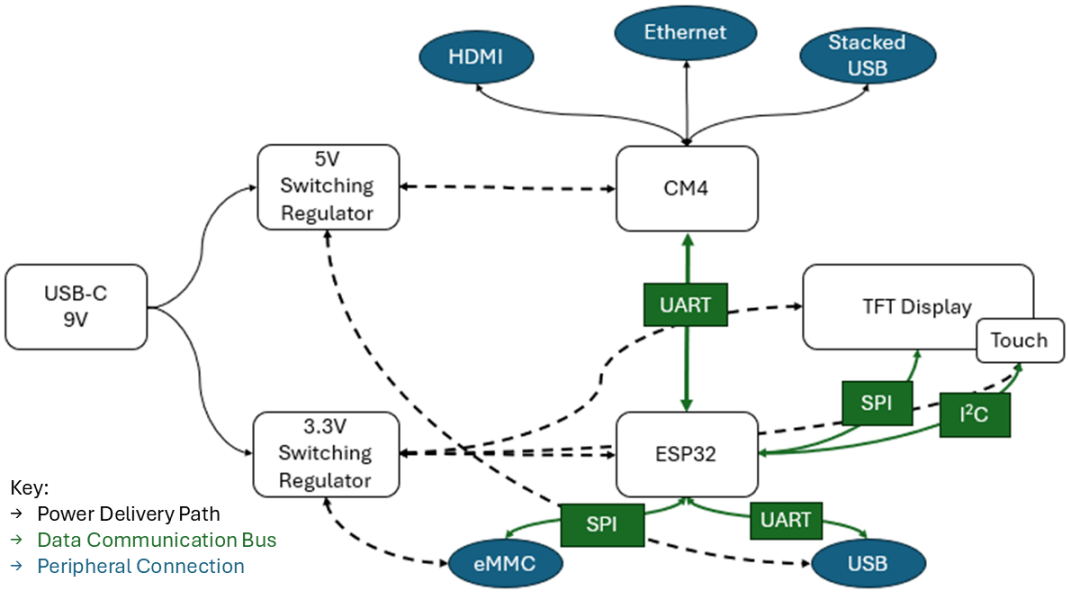


Figure 6: System Architecture Diagram

The Power Delivery Subsystem serves as the foundation of the system, accepting input power from a USB-C source and distributing regulated voltages to other modules. A 5V regulator provides power to the Compute Module 4 and USB network, while a 3.3V regulator supplies the ESP32-S3 microcontroller and the TFT display.

The CM4 manages high-level processing and external communication through its high-speed and low-speed interfaces. The high-speed connector handles HDMI output and USB connections to the external stacked USB 2.0 ports. The low-speed connector interfaces with Ethernet, system jumpers, and status LEDs, providing standard I/O and configuration functions. The CM4 also communicates with the ESP32-S3 via a dedicated serial interface (SPI), allowing command and data exchange.

The ESP32-S3 Subsystem functions as a secondary microcontroller responsible for handling the TFT display interface. Powered by the 3.3V, it drives the display and manages graphics rendering, user feedback, or other peripheral-level tasks.

The TFT Display Subsystem acts as the user interface element of the PwnBoxer system. It is directly controlled by the ESP32-S3 and receives 3.3V power. The CM4 can indirectly update or retrieve data displayed on the TFT by communicating through the ESP32-S3.

The architecture ensures clear separation between processing, control, and interface responsibilities. The CM4 handles computation and connectivity, while the ESP32-S3 bridges communication to the TFT display, all powered through a unified and regulated power delivery network.

6.3 Power Delivery Subsystem

The PwnBoxer is powered through a single USB-C connector dedicated exclusively to power delivery. The USB-C input provides a nominal 9V supply, which feeds the primary power regulation stage of the system. Two LM2576SX-ADJ/NOPB buck regulators generate the required system rails: a regulated 5V rail for the CM4, stacked USBs, HDMI, and Ethernet interfaces, and a 3.3V rail for the ESP32-S3, TFT display, and other low-voltage peripherals. The architecture prioritizes conversion efficiency, thermal stability, and low-noise power distribution for sensitive digital subsystems.

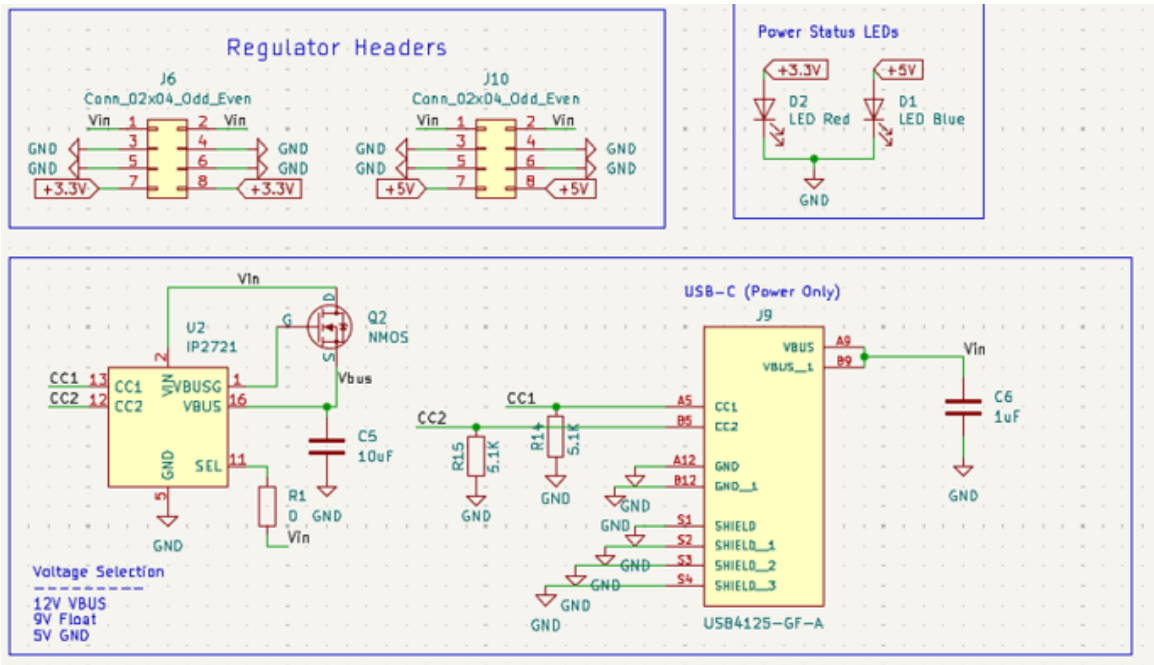


Figure 7: Power Delivery System Design - KiCad

6.3.1 5V Regulation – LM2576SX-ADJ/NOPB

The 5V system rail is generated using an LM2576SX-ADJ/NOPB buck regulator configured through an external feedback network. The regulator steps the 12V USB-C input down to a stable 5.0V output capable of supplying up to 2.5A of continuous load current.

The feedback divider consists of a 3.05 KΩ upper resistor (Rfbt) and a 1.00 KΩ lower resistor (R_{fbb}), setting the output voltage according to the standard LM2576 adjustable configuration. The power stage follows the recommended design guidelines and includes a 100 μH inductor (39.49 mΩ DCR), a 5A Schottky rectifier with a 550 mV forward drop,

and a 470 μF low-ESR output capacitor to minimize ripple and maintain transient stability. A 100 μF low-ESR input capacitor is placed close to the VIN pin to reduce input ripple currents.

This 5V rail powers the CM4, USB components, Ethernet circuitry, and HDMI interface, providing a stable and efficient supply for the system's higher-current digital subsystems.

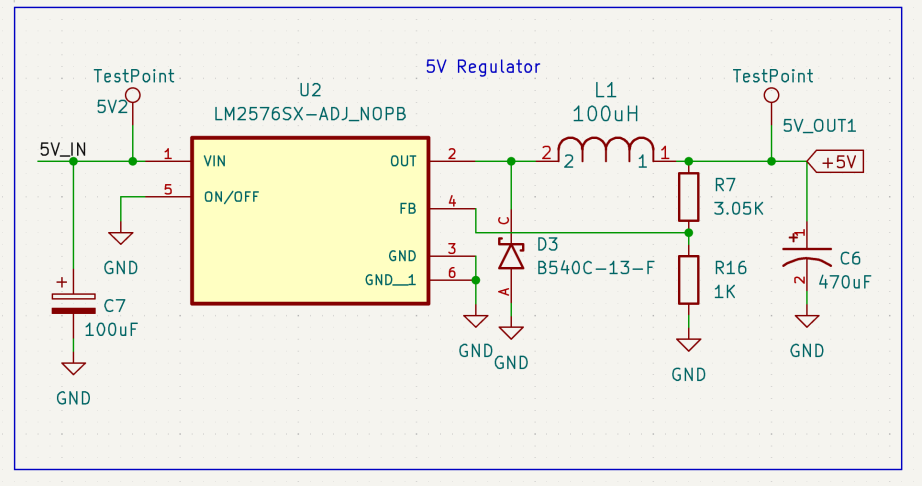


Figure 8: 5V Regulator Design - KiCad

The circuit includes a 4.7μH inductor, 22μF input and output capacitors, and appropriate feedback network connection to achieve a clean 5V output.

6.3.2 3.3V Regulation – LM2576SX-ADJ/NOPB

The 3.3 V rail is produced using a second LM2576SX-ADJ/NOPB configured for low-voltage operation. The output voltage is set using a 1.69 kΩ upper feedback resistor (R_{fbt}) and a 1.00 kΩ lower resistor (R_{fbb}), producing a regulated 3.3 V output from the 12 V USB-C input.

The power stage uses a 100 μH inductor (39.49 mΩ DCR), a 5 A Schottky diode with a 550 mV forward voltage, and a 1 mF low-ESR output capacitor, delivering excellent filtering and low ripple performance. A 100 μF low-ESR input capacitor stabilizes the regulator input and reduces high-frequency noise.

This 3.3 V rail powers the ESP32-S3, TFT display module, and all other 3.3 V logic devices within the system, ensuring clean and reliable power delivery under dynamic load conditions.

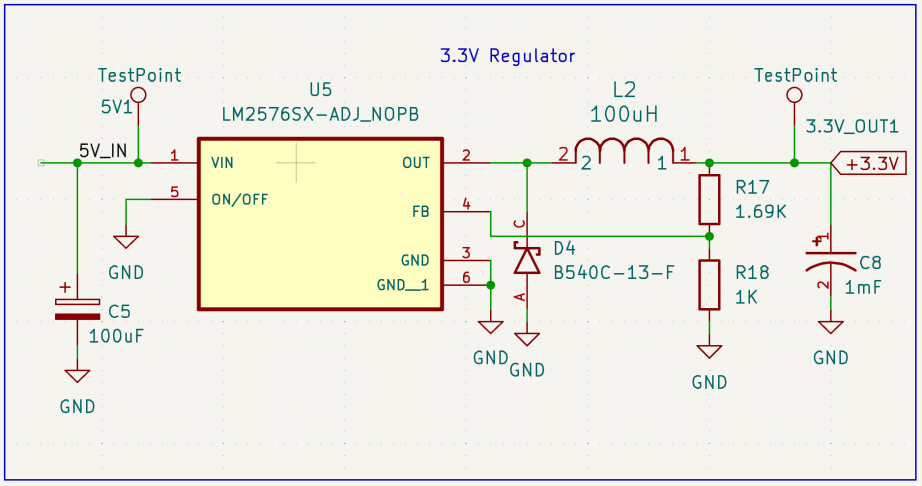


Figure 9: 3.3V Regulator Design - KiCad

The 3.3V circuit employs the same LC filter configuration, with a 4.7μH inductor and 22μF input and output capacitors. The converter provides stable operation under varying loads, ensuring reliable logic-level performance for the ESP32 and display drivers.

6.3.3 Design Consideration

To enhance robustness, the system design may incorporate power-ORing circuitry that enables seamless source switching between PoE and USB-C power inputs. This feature ensures uninterrupted operation if one power source fails and provides additional flexibility for maintenance or field deployment. By isolating and automatically selecting the active input source, the system avoids reverse-current conditions and protects upstream supplies.

An additional design consideration was whether to remove the dedicated 5 V regulator stage and instead use the selected 5 V output from the USB-C input as the primary supply rail for the Raspberry Pi Compute Module 4 (CM4). Selecting 5 V directly from USB-C simplifies the power architecture by reducing component count, PCB area, cost, and conversion losses, while also lowering thermal dissipation. However, removing the intermediate regulation stage reduces voltage headroom and increases sensitivity to voltage drops across connectors, protection components, and PCB traces, particularly under peak CM4 load conditions. It also limits flexibility when integrating alternative power sources such as PoE or higher-voltage USB Power Delivery profiles. Therefore, this decision requires balancing simplicity and efficiency against voltage stability and future scalability within the overall power delivery subsystem.

6.4 Raspberry Pi CM4 Subsystem

The CM4 Subsystem serves as the central processing unit of the PwnBoxer system. The CM4 receives its primary 5V power input from the Power Delivery Subsystem. It performs high-level computation, communication, and control functions, interfacing directly with the

ESP32-S3, Ethernet, HDMI, and USB devices. The CM4 integrates a Broadcom BCM2711 quad-core ARM Cortex-A72 processor and provides flexible I/O through High-Speed and Low-Speed connectors, both of which are utilized in this design.

6.4.1 CM4 Low-Speed

The low-speed connector handles eneral-purpose I/O like the GPIO pins and Ethernet.

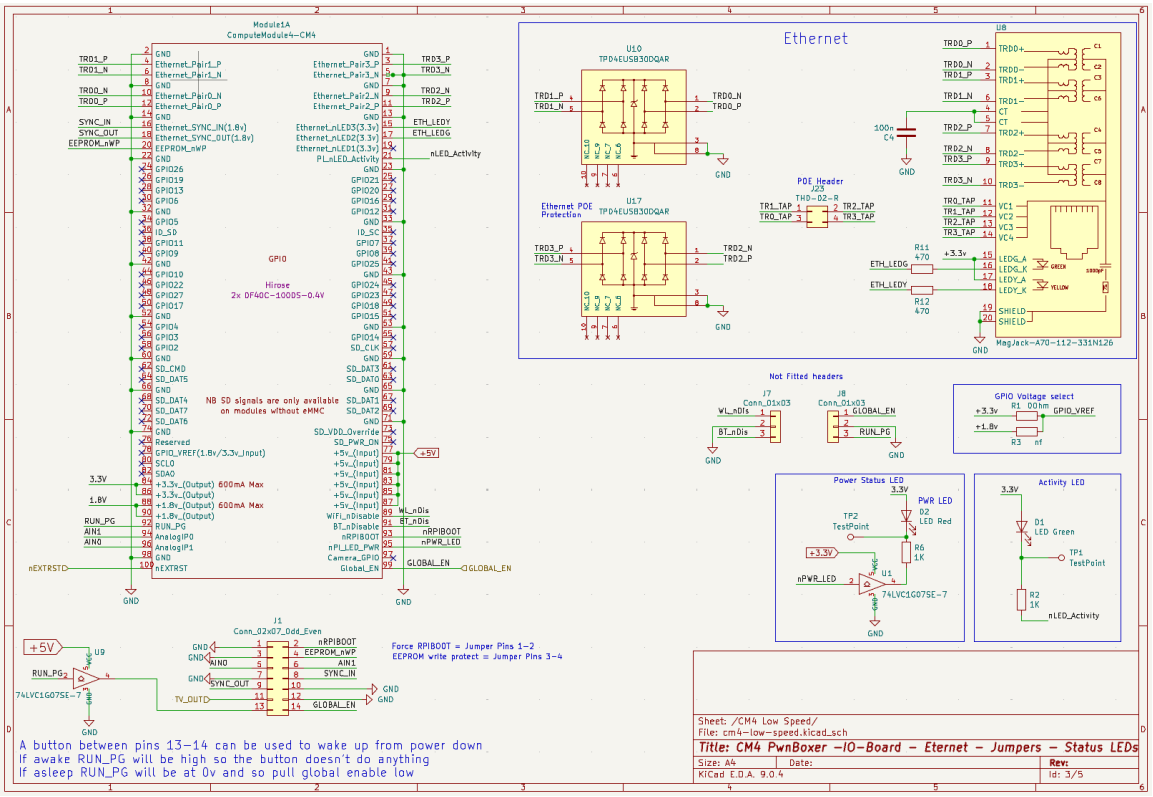


Figure 10: CM4 Low-Speed Connections; Ethernet, Jumpers

The following components are connected to this interface:

- **Ethernet (A70-112-331N126):** Provides 10/100/1000 Base-T with AutoMDIX. This connection also supports Power over Ethernet+ (PoE+), offering an optional alternate power input path.
- **Jumper Configuration:**These jumper and control options provide hardware-level flexibility during development and testing, simplifying recovery and reprogramming operations without external tools.
  - **EEPROM Write Protect (Pins 3-4):** Allows hardware-level protection for EEPROM memory to prevent unintentional writes.
  - **Force RPIBOOT (Pins 1-2):** Enables booting the CM4 into RPIBOOT mode for USB-based firmware flashing or EEPROM recovery.

– **Wake from Power Down (Pins 13-14):** A pushbutton connected across these pins enables system wake-up from sleep or full power-down states. When the CM4 is powered on, the RUN-PG (power-good) signal remains high, disabling the wake function. When powered down, RUN-PG is low, and pressing the button pulls the global enable line low to initiate a wake event.

6.4.2 CM4 High-Speed

The high-speed connector manages demanding interfaces such as PCIe, USB 2.0/3.0, HDMI, and MIPI CSI/DSI camera and display connectors.

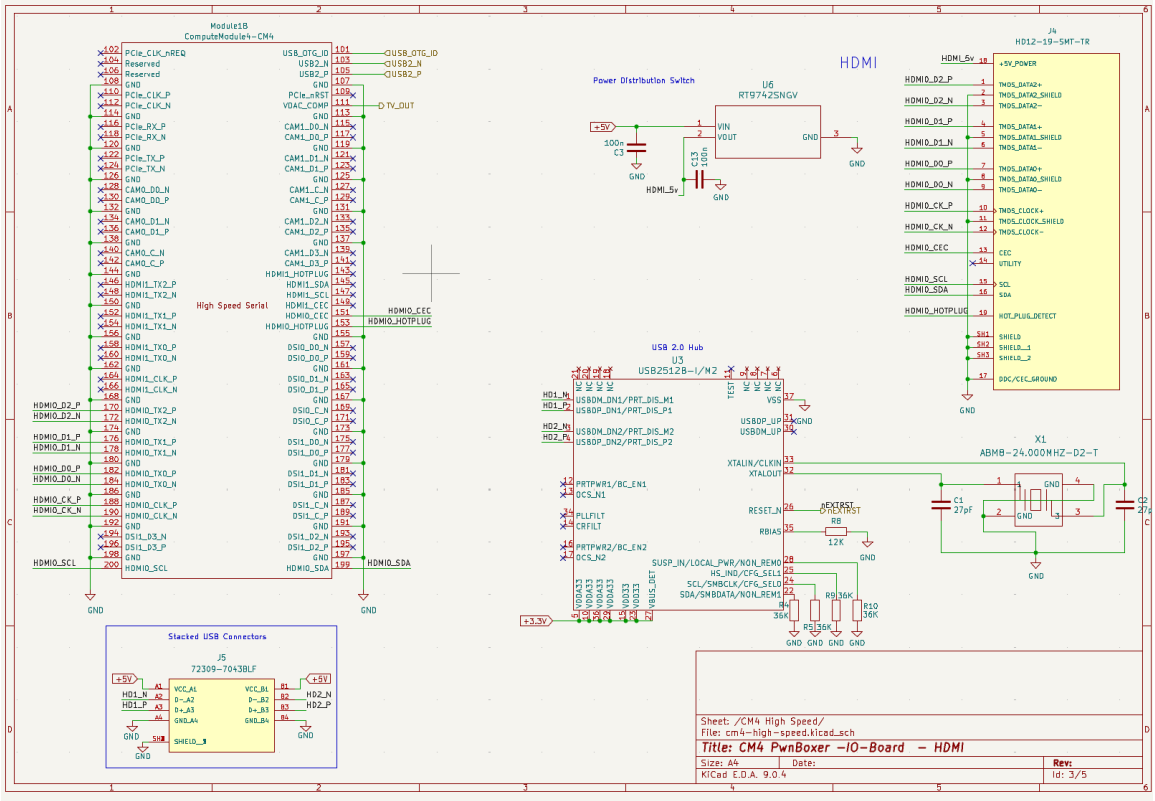


Figure 11: CM4 High-Speed Connections; HDMI and USB Hub

For the PwnBoxer the following components are connected to this interface:

- **HDMI Output (HD12-19-SMt-TR):** The CM4 provides a single HDMI 2.0 output for display interfacing. The connector used in this design is the HD12-19-SMT-TR, supporting 19 pins with differential TMDS pairs routed directly from the CM4's high-speed lines. HDMI is used for visual display or debugging purposes when connected to an external monitor.
- **Power Distribution Switch (RT9742SNGV):** The RT9742SNGV is integrated into the HDMI power line to safely control and distribute 5 V (on HDMI pin 18) to external devices. This device provides overcurrent and short-circuit protection, ensuring

that the 5 V HDMI output rail remains stable even under varying load conditions. When HDMI is connected, the RT9742SNGV enables power delivery only after detecting valid hot-plug and 5 V source readiness, complying with HDMI specification requirements.

- **USB Hub Controller (USB2512B-I/M2):** CM4 only has a single USB host port. To have two or more port a USB Hub is necessary.
  - **Stacked USB 2.0 Dual Port (72309-7043BLF):** For external peripherals such as keyboards and mice.

6.5 ESP32-S3 Subsystem

The ESP32-S3 Subsystem serves as a dedicated control and interface module within the PwnBoxer architecture. Its primary functions are to manage the TFT display, interface with the Raspberry Pi CM4, and handle local processing. The ESP32-S3 was selected for its dual-core Xtensa LX7 processor, integrated Wi-Fi/Bluetooth support, and strong peripheral capabilities that complement the CM4's high-level computing functions.

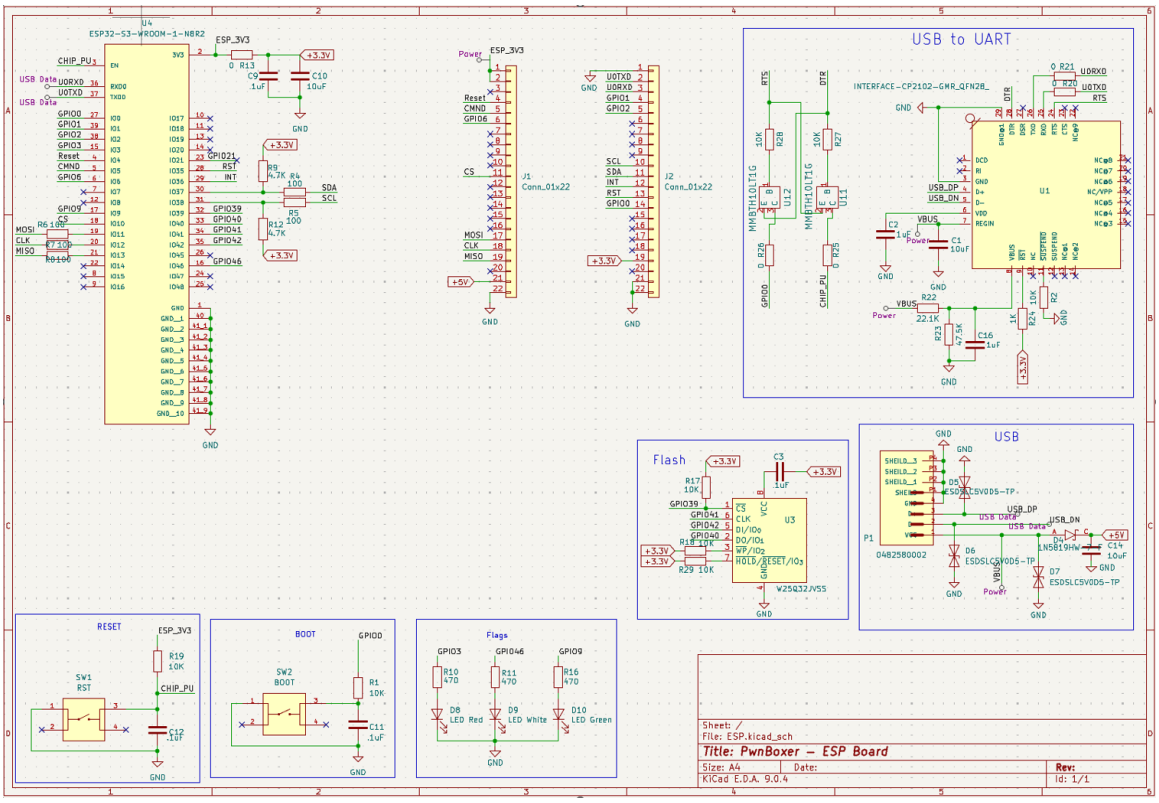


Figure 12: ESP32-S3 connections; TFT Display, TFT Touch, Micro USB to UART

In the system, the ESP32-S3 operates as a co-processor that offloads peripheral-level and user-interface operations from the CM4. This distributed approach allows the CM4 to focus on computationally intensive or host-level processes, while the ESP32-S3 handles

time-critical I/O such as display refresh, low-latency feedback, or potential future wireless communication. The ESP is powered by the 3.3V rail from the Power Delivery Subsystem and communicates with the CM4 over SPI. This communication channel allows the CM4 to send commands, update display data, or request system information through the ESP.

6.5.1 Flash Memory Interface

An internal memory is present on the ESP32-S3 to store firmware, calibration data, and any static assets used by the TFT display. This external memory expands the ESP's storage capacity beyond its internal flash, enabling flexible firmware updates and efficient resource allocation for display-driven tasks.

6.5.2 Display Control Interface

The TFT display is directly connected to the ESP32-S3 via 8-bit parallel. The ESP handles all rendering and buffer management locally, ensuring smooth operation independent of the CM4's processing load. The interface will support touch input via I2C, allowing the ESP to read capacitive touch signals.

6.6 TFT Display Subsystem

The TFT Display Subsystem provides the primary visual interface for the PwnBoxer system. It utilizes the CFAF320480C7-035TC module, a 3.5-inch full-color TFT display with a resolution of 320×480 pixels, an integrated capacitive touch panel, and support for an 8-bit parallel communication interface. This subsystem enables both high-speed graphical output and intuitive touch-based user interaction.

The display module connects to the system through two ZIF interfaces: a 50-position ZIF connector for the main TFT display signals, and an 8-position ZIF connector dedicated to the capacitive touch panel. These connectors ensure reliable, low-resistance signal routing for both high-speed display data and precision touch sensing.

The display subsystem is responsible for rendering user interface elements, visual feedback, and system status information under the direction of the ESP32-S3 Subsystem.The 8-bit parallel interface allows for significantly higher data throughput compared to SPI-based communication, which is essential for achieving smooth frame rates and responsive graphics updates.

Table 1: TFT Display Subsystem Interface Connections

Signal Group	Connected Component	Description
8-bit Data Bus (D0–D7)	ESP32-S3 GPIOs	Parallel data transfer for pixel data
RD, WR, RS, CS, RESET	ESP32-S3 GPIOs	Control lines for display read/write operations
VCC, GND	Power Delivery Subsystem	3.3V supply and common ground
Backlight (LED+)	Power Delivery Subsystem	Drives the LED backlight
Touch I2C (SCL, SDA)	ESP32-S3 I2C Interface	Capacitive touch panel communication
INT, RST (Touch)	ESP32-S3 GPIOs	Interrupt and reset lines for touch controller

The display module connects to the system through two ZIF interfaces: a 50-position ZIF connector for the main TFT display signals, and an 8-position ZIF connector dedicated to the capacitive touch panel. These connectors ensure reliable, low-resistance signal routing for both high-speed display data and precision touch sensing.

The parallel data bus and control lines are routed from the ESP32-S3, allowing the microcontroller to write pixel data directly to the display's internal GRAM. The capacitive touch controller on the display communicates through an I2C interface, enabling the ESP to read precise touch coordinates and gestures with minimal latency.

6.6.1 Design Consideration

Mechanical placement is an important design consideration. The display's flexible printed cable (FPC) will be oriented to minimize stress and ensure easy assembly within the enclosure.

7 Software Design

The **Pwnboxer** project requires the implementation of three major software systems to manage user interaction, automated testing, and email reporting. This chapter details each of these software subsystems and explains how they interact with each other. It also covers the communication protocols used at both the firmware and software levels and illustrates how these protocols facilitate coordination between the subsystems.

7.1 User Application

This project involves extensive user interaction. Therefore, the user application is designed with a minimalist interface, allowing the user to configure settings, launch tests, connect to a technician, and reset the device with minimal difficulty. The user should be able to perform the following actions:

1. **Initial device setup on first boot**
   (a) Configure WiFi by selecting the correct SSID or entering a hidden SSID. For enterprise networks, the user can input a username and password.
   (b) Configure an email address to send vulnerability reports.
   (c) Configure the IP address and port to initiate a reverse shell connection.
2. **Launch automated tests**
   - The user can select from a list of pre-configured tests based on the configuration files stored on the CM4.
3. **Establish a reverse shell connection**
   - Allows a technician to remotely access the device for maintenance or troubleshooting.
4. **Reconfigure settings**
   (a) Update WiFi credentials if network changes are necessary.
   (b) Update the email address used for sending vulnerability reports.
   (c) Adjust the IP address and port for reverse shell connections.
   (d) Adjust screen brightness.
   (e) Enable or disable audio tones.
   (f) Perform a factory reset of the device.

User settings are stored securely in the ESP32 flash memory and retrieved on subsequent boots. Usernames and passwords are encrypted using a key generated from the ESP32 efuse system. A one-time key transfer occurs between the ESP32 and the CM4 to ensure both devices share the encryption key. The CM4 then stores this key in a keyring to prevent plaintext storage.

Once the key exchange is complete, the user can perform the initial setup of the device using the TFT touchscreen. Through the interface, users can select a WiFi network (broadcasting or hidden) and enter credentials to access the network. They can also choose between static IP addressing or DHCP. If using Ethernet through the CM4, the user can skip WiFi configuration entirely.

The ESP32 also allows users to modify configuration settings after initial setup. This includes editing WiFi settings, updating shell connection parameters, adjusting screen brightness, and enabling or disabling audio tones.

The device comes pre-configured with an initial penetration testing toolkit, enabling users to run automated tests without further configuration. During the CM4 setup phase, the system gathers information about available tests and sends them to the ESP32 for display. The user can then select individual tests or execute the full testing workflow. A progress bar indicates workflow completion, and a cancel button allows users to stop tasks if the device becomes unresponsive. Detailed error codes are provided to facilitate troubleshooting and technician support.

When a test or workflow is initiated, the ESP32 communicates with the CM4 via SPI, notifying the task scheduler that a new task has been added to the queue. The task scheduler executes tasks sequentially and reports progress by aggregating completion data from individual tasks. Upon completion, results are securely transmitted to an off-site server via HTTPS. The server processes these results with a large language model (LLM) to generate a condensed, digestible email report.

Once the workflow finishes, the off-site server sends a success confirmation to the CM4, which in turn notifies the ESP32, allowing the user to view the result. The task schedulers error handler manages any errors encountered in the schedulers workflow, ensuring that they are reported to the user and logged for technician review.

7.1.1 Display and Storage Subsystem

The LCD display is interfaced to the ESP32-S3 using an 8-/16-bit parallel bus to enable high-throughput communication for rapid screen refresh rates. The chosen display includes a capacitive touch controller connected via SPI, allowing users to directly interact with the system by selecting on-screen options and input elements.

The ILI9488 TFT controller is being utilized within our LCD, which requires a sequential initialization sequence immediately after power-up to place the display into an operational state. Upon boot, the ESP32-S3 asserts the hardware reset line to discharge internal control logic, followed by a delay to allow the display timing circuits and regulators to stabilize. Once reset is released, the firmware sends a command sequence over the selected parallel interface corresponding to the ILI9488 datasheet. These commands configure power parameters, gamma correction, signal interface mode, display orientation, and pixel format. In this application, the controller is set to 18-bit RGB (6-6-6) mode to support richer color depth and meet LVGL's rendering expectations.

The initialization sequence further configures display-specific functionality including frame rate control, internal display inversion, and VCOM voltage levels to ensure consistent contrast and stability across the 320×480 active matrix. Address window commands are sent to establish the default drawable area, followed by RAM write pointer initialization. Only after completing this configuration does the controller accept a Sleep Out command, and finally a Display On command, which activates the display scan logic and backlight.

Graphical user interface (GUI) rendering is handled using the Light and Versatile Graphics Library (LVGL), providing a flexible framework for creating dynamic and responsive user interfaces on resource-constrained embedded systems. For 8-bit parallel mode, the `TFT_eSPI` library simplifies integration with pre-supported ESP32 hardware configurations. For 16-bit parallel operation, a custom driver is implemented to optimize LVGL interfacing. Display resources, such as GUI assets and bitmaps, can be pre-stored in external SPI flash to reduce internal memory consumption.

Additional external memory components, including PSRAM and SPI flash, are connected to the dedicated memory SPI interfaces (SPI0 and SPI1). The ESP32-S3 boots from external SPI flash, while PSRAM expands available memory for frame buffers and dynamic LVGL allocations.

7.1.2 ESP32-Firmware Workflow Diagram

This graphic illustrates the flow of user interactions and how initialization and task execution occur within the ESP32 as detailed in section 7. It also highlights the inputs and outputs exchanged between the user, the ESP32, flash memory, and the CM4.

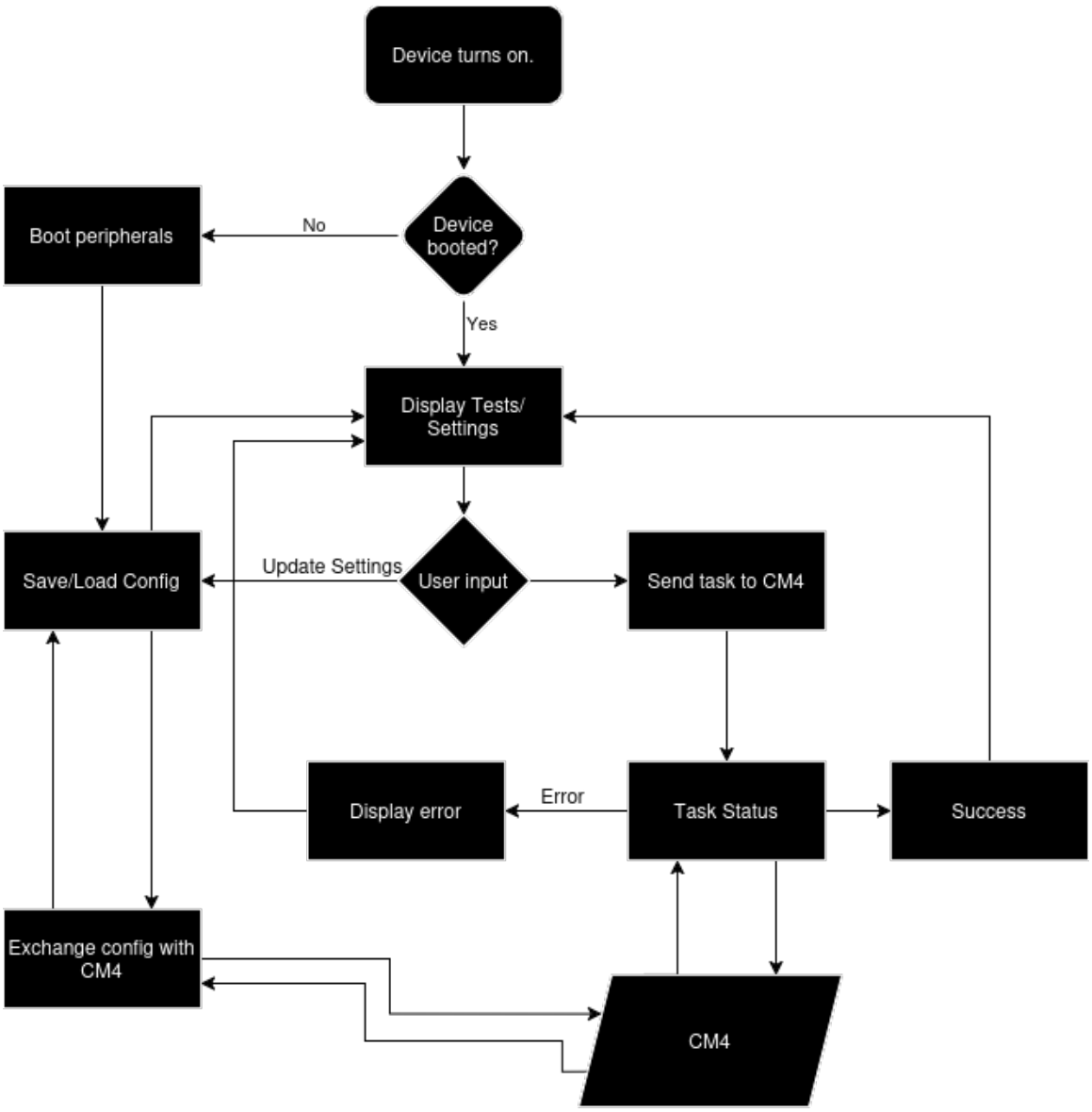


Figure 13: ESP32 User Interaction Flow Chart

7.1.3 Cross-Board Communication

Communication between the ESP32-S3 and the CM4 is conducted using a dedicated SPI data pipeline, allowing bi-directional data transfer between the two boards. On the ESP32 side, the SPI master driver is configured to support transaction completion interrupts, with post-transaction callback functions registered for each SPI device. When data is received from the CM4, the callback is triggered, allowing the system to inspect the packet, determine the packet type, validate packet integrity, and dispatch the packet for further processing.

Packets are distinguished using fixed headers, enabling the firmware to parse and route incoming data to the appropriate handler. Payloads are structured with predetermined lengths

for each packet type, which allows the ESP32-S3 to detect and reject malformed or incomplete transmissions.

Upon packet reception, ESP32 tasks may be dispatched to handle various system operations such as updating task progression, sending updated information (e.g., credentials, email, or execution commands), or managing other peripheral interactions. This approach prevents blocking operation of the SPI interface, allowing for continued message transactions. For large or complex transfers, DMA-enabled SPI transactions are utilized to offload data movement from the CPU to reduce processing overhead and enabling low-latency communication.

7.1.4 GUI Threading and Display Management

While cross-board communication is handled through SPI transactions and task dispatching, the ESP32-S3 also maintains dedicated threads for GUI rendering and display management. The display thread is pinned to Core 1 to ensure consistent execution, and provide smooth frame updates and responsive touch input handling. This thread is responsible for:

- Managing the LVGL rendering pipeline and updating frame buffers.
- Communicating with the TFT display over the 8-/16-bit parallel bus.
- Controlling backlight brightness and display-related peripherals.
- Handling touch input events from the capacitive controller and dispatching them to GUI tasks.

Separating GUI management from cross-board communication tasks ensures that SPI transactions and LVGL rendering do not interfere with one another. This separation of concerns allows for a modular approach to developing the system, such that if we were to change the communication protocol between the ESP32 and the CM4, there would be minimal disruptions to the rendering code.

On the CM4, a similar task scheduler monitors the SPI endpoint for incoming data. When packets arrive they are parsed by checking the header type, verified by ensuring that bounds restrictions are met, and dispatched to the appropriate task function. Maintaining the architecture as symmetric as possible is critical to ensuring that minimal packets are dropped; additionally, it is important to consistently implement error handling such errors like overflows are handled gracefully and safely by both the firmware and software.

7.2 Task Implementation

Tasks are defined as self-contained units described by a YAML configuration file and stored in a dedicated `user-tasks/` directory. The YAML schema provides a simple, declarative description of how the task should be executed and where its outputs should be stored. Keeping tasks externalized as configuration files enables operators to add or modify tasks without changing the core application binary, and it allows the runtime to validate, schedule, and sandbox task execution consistently.

7.2.1 Task Configuration Schema

Each task configuration file follows a fixed schema that includes the following fields:

- **Task Name**: Human-readable identifier for the task.
- **Executable**: Absolute path to the program or script to execute.
- **Arguments**: List of command-line arguments or flags to pass to the executable.
- **Interpreter**: Optional path to an interpreter (e.g., `/usr/bin/python3`, `/usr/bin/bash`). When specified, the runtime invokes the interpreter with the executable and args.
- **Output**: Outputs produced by the task. This includes
  - `output.filename`
  - `output.path` (location to store output.)
- **Environment**: Optional environment variables and virtual environment settings (e.g., `VENV_PATH` or `RVM_PATH`) required by the task.
- **Retries**: Optional retry policy failure (max attempts, backoff).
- **Hooks**: Optional pre- and post-execution commands or scripts (for setup and teardown).
- **Resources**: Limits for resource usage.
  - **Timeout**: Optional time in minutes of maximum time a task should run for.
  - **Memory Usage**: Optional resource limit (memory usage in MB)

The runtime performs schema validation at load time; invalid or missing required fields cause the task to be rejected with a descriptive error. This validation lowers operational surprises and enables safe, automated intake of user-supplied tasks.

The following configuration structure demonstrates the structure for all tasks excluding that of the full workflow task.

```
task_name: "nuclei_scan"
executable: "/usr/local/bin/nuclei"
args: ["-t", "templates/", "-o", "results.json"]
interpreter: null
output:
  filename: "results.json"
  path: "/var/task_artifacts/nuclei_scan"
environment:
  PATH: "/usr/local/bin:/usr/bin:/bin"
resources:
  timeout_seconds: 3600
  memory_mb: 2048
```

7.2.2 Execution Model

When a task is launched, the following sequence is used to ensure consistent execution and reliable artifact collection:

1. **Preparation**: The runtime resolves relative paths, creates the target output directory, applies file-system permissions, and sets up ephemeral working directories when necessary.
2. **Environment setup**: If the YAML specifies an interpreter or a virtual environment, the runtime activates it or invokes the interpreter explicitly. Environment variables specified in the task configuration are set for the task process only.
3. **Execution and monitoring**: The executable is launched with STDOUT/STDERR captured to log files. The runtime monitors process exit status. If the task exceeds configured it is terminated and marked as failed.
4. **Post-processing and artifact collection**: On process exit the runtime gathers output artifacts (standard output, standard error, generated files) and stores them in the configured output location. If specified, artifacts can be compressed and checksummed (SHA256) for later validation.
5. **Result reporting**: A canonical run log is written (see next subsection) and metadata about the run (start/stop times, return code, resource usage, artifact list) is emitted to the scheduler or orchestration service for downstream processing.

7.2.3 CM4 Software Workflow Diagram

This graphic illustrates how the flow of initialization and task execution occur within the CM4 as detailed in section 7. It also highlights the inputs and outputs exchanged between the user,the ESP32, the CM4 and the reporting server.

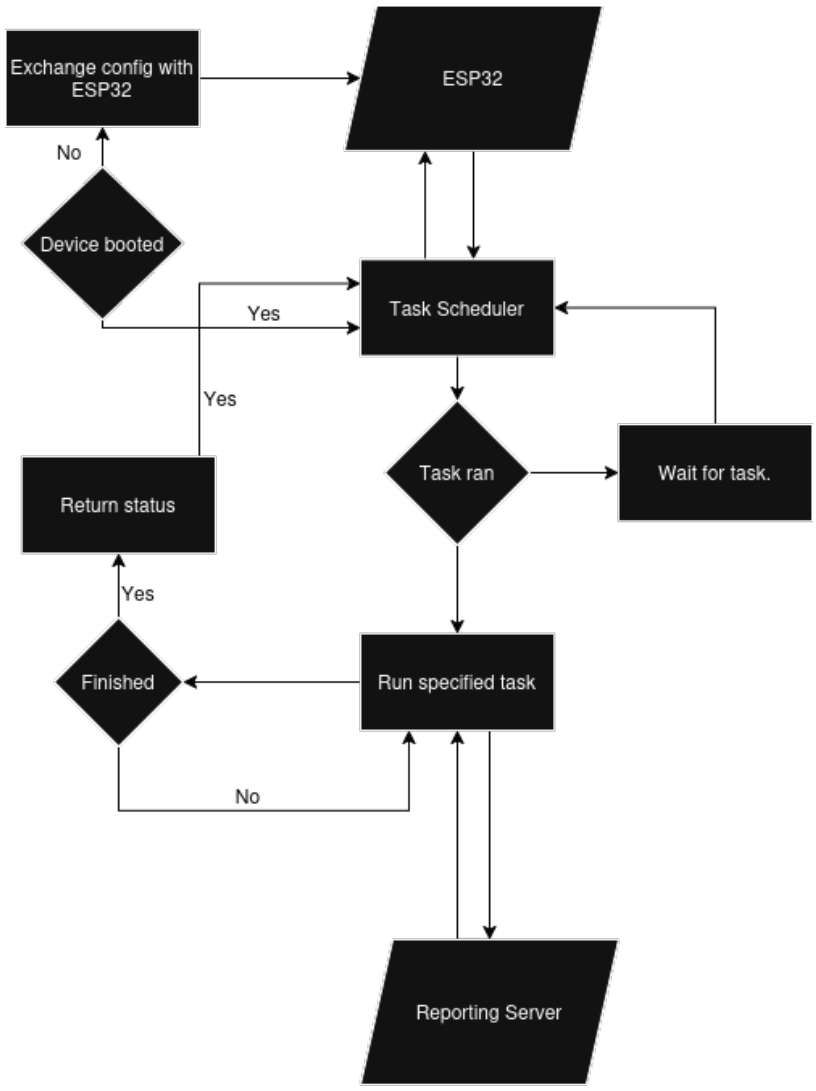


Figure 14: CM4 User Interaction Flow Chart

7.2.4 Full Workflow Task

When executing the full workflow task, the scheduler enumerates all user-provided configuration files alongside the default core configurations, ensuring that every required task in the pipeline is executed in sequence. The full workflow itself is defined by a dedicated configuration file, structurally similar to an individual task's configuration, but with an additional field that specifies the list of task configuration files to be included in the pipeline. This architecture allows for the customization of the full task workflow with minimal reconfiguration.

The following configuration structure demonstrates the structure example structure for the full task workflow.

```
task_name: "full_scan"
executable: "/usr/local/bin/pwnboxer/pwnboxer_scheduler"
```

```
interpreter: python3
output:
  filename: "results.json"
  path: "/var/task_artifacts/full_tasks"
environment:
  PATH: "/usr/local/bin/pwnboxer/bin/activate"
resources:
  timeout_seconds: 3600
  memory_mb: 4098
tasks:
  task_1: /usr/local/bin/pwnboxer/tasks/user-tasks/nuclei_task
  task_2: /usr/local/bin/pwnboxer/tasks/user-tasks/msf_task
```

7.2.5 Integration with CM4 and the Overall System

Task execution is performed on the CM4, which hosts the full operating environment and supports a wide range of development stacks and package managers (e.g., `venv`, `pyenv`, Bundler, RVM). The runtime supports the use of interpreter paths supplied in the task configuration, allowing operators to point to language-specific virtual environments or system interpreters. This flexibility enables:

- Running native compiled binaries as well as interpreted scripts (Python, Ruby, Node, etc.).
- Leveraging preinstalled frameworks and their APIs (e.g., Metasploit console modules) by invoking the appropriate CLI or library entrypoints exposed by the framework.
- Re-creating language-specific sandboxes on demand using `venv` or container images when reproducible environments are required.

7.3 Reporting Server Architecture

The reporting server provides an off-device back-end for aggregating scan results, monitoring the health of deployed PwnBoxer units, and exposing a secure interface for future report viewing and notification. While the CM4 and ESP32-S3 handle on-device orchestration and user interaction, the reporting server centralizes long-term storage, log correlation, and external access. In the prototype implementation, the reporting server is realized as a cloud-hosted Linux virtual machine running Apache, a Node.js Express application, and host-based security controls such as Fail2Ban and a restrictive firewall.

At a high level, the design is driven by the following goals:

- Secure network exposure of only the minimal set of services required (HTTP for status and future APIs, SSH for administration).
- Authenticated submission of data and heartbeats from PwnBoxer devices.

- Continuous availability monitoring via a lightweight heartbeat endpoint.
- Defense-in-depth through a web application firewall (WAF) layer, host-based intrusion detection, and rate-limiting.
- Audit-friendly logging of both application-level events and security events for later review.

The following list summarizes the major responsibilities of the reporting server within the overall system:

1. Accepting secure, token-authenticated API requests from PwnBoxer devices for status and report submission.
2. Exposing a simple, human-readable status dashboard for operators.
3. Enforcing request-level protections and host-based bans against abusive or automated traffic.
4. Persisting logs of requests, authentication attempts, and security actions.
5. Providing a foundation on which a richer reporting and notification portal can be built in future iterations.

In the current phase, the implementation focuses on the heartbeat and security subsystems, while the more advanced report-ingestion and LLM-based synthesis pipeline are left as future work. The architecture is intentionally modular so that additional REST endpoints and processing services can be added without changing the security posture of the exposed surface.

7.3.1 Report Submission and Processing Pipeline

The full reporting pipeline is conceptually divided into three logical stages: *ingestion*, *processing*, and *distribution*. Although the prototype primarily implements health monitoring and basic API handling, the interfaces for higher-level report submission have been designed so they can be extended in subsequent project phases.

On the ingestion side, the CM4 is responsible for packaging the results of one or more tasks into a structured payload. A typical payload contains:

- Metadata about the PwnBoxer unit (device identifier, firmware/software versions, and timestamp).
- Task-level information (which workflow was executed, which targets were scanned, and high-level statistics).
- Pointers to log files or artifacts stored locally on the CM4 that may be uploaded to the reporting server for long-term archival.

The reporting server exposes a token-protected REST API endpoint that the CM4 can use to submit this data. Requests are made over HTTP to the front-end Apache web server, which

forwards them internally to the Node.js application. The Node.js layer is implemented using the Express framework and is responsible for:

- Parsing JSON payloads and validating basic structure.
- Verifying an authentication token shared between the CM4 and the reporting server.
- Recording the time and origin IP of each submission for later audit.
- Returning structured responses indicating success, failure, or rate limiting.

In the current prototype, the primary implemented API is a lightweight *heartbeat* endpoint that functions as a minimal stand-in for a more complex report-ingestion pipeline. The CM4 periodically issues a POST request to the reporting server containing only an authentication token. If the token is valid, the server updates its internal state to reflect the most recent contact time for that device. This same mechanism can be extended to carry full report payloads in future phases without changing the authentication or security model.

The processing stage of the full design anticipates the use of an additional microservice or container that performs higher-level analysis on submitted artifacts (scan logs, configuration snapshots, and vulnerability summaries). While this stage is not fully implemented in the semester prototype, the reporting API is structured so that a downstream worker can be triggered asynchronously whenever a new submission is received. The downstream worker would be responsible for parsing logs, extracting key findings, and generating a human-readable summary.

7.3.2 Report Distribution and Notification

The envisioned end state for the reporting server includes both automated email notifications and a browser-based portal for viewing historical reports. Operators should be able to log in, browse previous runs for a given device, and download attached artifacts such as scan logs or generated PDFs.

In the prototype, the distribution mechanism is intentionally minimal: the primary "report" is the live heartbeat status page exposed by the Node.js application. This page is designed to be operator-friendly, providing an at-a- glance indication of whether a PwnBoxer unit has contacted the reporting server within a configured time window. The longer-term design extends this model into a richer reporting portal while preserving the same underlying security controls.

For future work, the reporting server will integrate with an external email delivery service (such as Postmark or an equivalent transactional email provider) to deliver notifications when:

- A PwnBoxer unit has not checked in within a configured threshold.
- A task completes with critical or high-severity findings.
- New reports are generated and archived on the server.

By keeping the email layer logically separate from the ingestion and processing pipeline, the system maintains a clean separation of concerns. The reporting server can queue notifications and hand them off to the email provider, while still enforcing the same authentication and logging rules for all outbound events.

7.3.3 Web Portal and User Interface

The web-facing interface for the reporting server is presently implemented as a lightweight Node.js Express application. This application provides two primary entry points:

1. A POST `/heartbeat` REST endpoint used by PwnBoxer devices to signal liveness.
2. A GET `/` endpoint that renders a simple status page for human operators.

The heartbeat endpoint expects a JSON payload of the form:

```
{
  "token": "pwnBoxer2025!"
}
```

When a request is received, the Express application performs the following steps:

1. Parses the JSON body using a middleware such as `body-parser`.
2. Extracts the shared-secret token and compares it against a server-side constant.
3. If the token is valid, records the current server time as the most recent heartbeat for the requesting IP or device identifier.
4. Returns a JSON response indicating success and echoing the timestamp.
5. If the token is invalid, returns an HTTP 401 Unauthorized response.

Internally, the application tracks the most recent heartbeat time in memory. The GET `/` route renders a small HTML page that includes:

- A colored bar that is green when the last heartbeat is within a configured threshold (e.g., 20 minutes), and red otherwise.
- A human-readable timestamp for the last successful heartbeat.
- Simple layout styling using inline CSS to keep dependencies minimal.

The server's system timezone is configured to the appropriate local region (e.g., `America/New_York`) so that the displayed timestamp aligns with the operator's expectations. Alternatively, the application can call `toLocaleString` with an explicit timezone when formatting the date, ensuring consistent display regardless of the host's default configuration.

The Node.js process is managed by PM2, a production process manager for Node applications. PM2 is configured to start the `heartbeat` process on boot, automatically restart it if it crashes, and maintain log files for standard output and error streams. This ensures

that the heartbeat service remains available even in the presence of transient failures or VM reboots.

Although the current UI is intentionally minimal, it establishes the core backend patterns that a richer reporting portal will reuse: a Node.js/Express application behind Apache, token-based authentication for API calls, and a clear visual indicator of device health.

7.3.4 Data Management, Logging, and Security Telemetry

The reporting server maintains several layers of logging to support both operational troubleshooting and security monitoring. These logs fall into three broad categories:

- **Application logs**, generated by the Node.js heartbeat service.
- **Web server logs**, generated by Apache (access logs and error logs).
- **Security logs**, generated by system services and Fail2Ban.

Within the Node.js application, every heartbeat request can be logged with its timestamp, originating IP address (taking into account the `X-Forwarded-For` header when Apache is used as a reverse proxy), and result (success, invalid token, or rate-limited). These logs provide a straightforward audit trail of device liveness and authentication behavior.

Apache's access and error logs capture all inbound HTTP requests, including those that never reach the Node.js application (for example, malformed or suspicious probes blocked at the web server layer). These logs are important for diagnosing issues such as misconfigured clients, attempted directory traversals, or repeated scanning from a single IP.

On the host, Fail2Ban monitors log files such as `/var/log/auth.log` and Apache logs for patterns indicative of malicious behavior. When repeated failures are detected, Fail2Ban dynamically inserts firewall rules to block the offending IP address for a configurable period. This results in an additional layer of security telemetry: ban and unban events are themselves logged and can be correlated with other activity to build a complete picture of attacker behavior.

At the network level, the system firewall (configured using UFW, the Uncomplicated Firewall) restricts ingress traffic to only the ports required for operation: SSH (port 22) for administrative access and HTTP (port 80) for the web front-end. All other unsolicited inbound traffic is dropped. This restrictive policy significantly reduces the potential attack surface.

Together, these logging and telemetry layers enable operators to:

- Confirm that PwnBoxer units are reaching the reporting server on schedule.
- Detect and investigate authentication failures or suspicious HTTP requests.
- Identify external IPs that have been automatically banned by Fail2Ban.
- Reconstruct timelines of events during troubleshooting or security analyses.

7.3.5 Reporting Server Workflow and Hardening Measures

The end-to-end workflow of the reporting server combines the application logic of the heartbeat service with multiple layers of security hardening. At a high level, the flow for a typical heartbeat request is as follows:

1. A PwnBoxer device issues an HTTP POST request to the reporting server's public IP on port 80, targeting the `/heartbeat` path and including the shared token in a JSON body.
2. Apache receives the request as the front-end web server. Modules such as `mod_reqtimeout` enforce sane limits on header and body read times, mitigating slowloris-style attacks by terminating connections that send data too slowly.
3. Apache forwards valid requests to the internal Node.js process on `localhost:3000` using reverse-proxy directives. From the standpoint of the Node.js application, all requests appear to originate from the local host, while the true client IP is preserved in the `X-Forwarded-For` header.
4. The Express application enforces per-IP rate limiting. It maintains an in-memory map of the last successful request time for each client IP (based on `X-Forwarded-For` if present). If a client attempts to send heartbeats more frequently than once every two minutes, the server responds with HTTP 429 (Too Many Requests) and a descriptive error message. Otherwise, the request is processed as normal.
5. If the token in the request body matches the configured secret, the server updates the `lastHit` timestamp and returns a JSON response indicating success and the recorded time. If the token is invalid, the server responds with HTTP 401 and does not update the internal state.
6. The status page at the root path (`/`) reads the current `lastHit` timestamp and computes the time difference between now and the last heartbeat. If the difference is within the acceptable threshold, the page renders a green status bar; otherwise, it renders a red bar and indicates that the device has not checked in recently.

Surrounding this application-layer workflow are several hardening measures:

- **Fail2Ban for SSH and Apache:** A jail named `sshd` monitors authentication failures in `/var/log/auth.log` and bans IP addresses that exceed a configurable threshold of failed logins. Additional jails such as `apache-badbots` and `apache-noscript` watch for known malicious user agents and suspicious URL patterns in Apache logs.
- **Firewall enforcement via UFW:** Only ports 22 and 80 are permitted; all other inbound ports are denied by default.
- **Process supervision via PM2:** The heartbeat application is run under PM2, which restarts it automatically on crashes and preserves the process list across reboots. This reduces downtime caused by transient errors or manual mistakes.
- **Minimalist service set:** The server avoids running unnecessary network services. By

limiting the number of active daemons, the chances of an unpatched or misconfigured service being exploited are reduced.

While the present implementation focuses on heartbeat monitoring rather than full report ingestion, the reporting server architecture has been intentionally designed to scale. Adding new REST endpoints for uploading logs or requesting generated reports would reuse the same hardened surface: Apache as the front-end WAF, Node.js/Express as the application tier, Fail2Ban and UFW as host-based controls, and PM2 as the process supervisor. This separation of concerns ensures that future functionality can be integrated without compromising the core security guarantees established in the current prototype.

The reporting server is responsible for receiving artifacts from the PwnBoxer device, performing higher-level processing, and exposing a secure interface for technicians to monitor system health. In the current implementation, this role is fulfilled by an Ubuntu-based virtual machine (VM) hosted in the cloud. The VM hosts a lightweight Node.js web service that implements an authenticated heartbeat API and status page, along with a hardened software stack including a process supervisor, firewall rules, and intrusion-prevention controls.

From a high-level perspective, the reporting server is designed to satisfy the following objectives:

- Provide an authenticated HTTP API for the device to signal liveness and, in future revisions, upload scan results and logs.
- Expose a human-readable status dashboard that can be consulted by a technician or instructor during demonstrations.
- Maintain high availability through process supervision and automatic restart.
- Limit the external attack surface via network firewalls, brute-force protection, and basic web application hardening.
- Remain extensible so that future versions can integrate a full reporting pipeline (e.g., LLM-based summarization and email delivery) without redesigning the core architecture.

The remainder of this section details the implementation of the heartbeat API, process management with PM2, host hardening via `fail2ban` and a host firewall, and additional HTTP-level protections such as rate limiting and slowloris mitigation.

7.3.6 Heartbeat Service Overview

To verify that the on-device software stack is operational and reachable from the external network, the reporting server exposes a minimal heartbeat service. This service is implemented as a Node.js application using the Express framework. It provides two primary endpoints:

- **GET /** – Returns a compact status page that visualizes whether a valid heartbeat has been received within a recent time window. The page displays a colored bar (green

or red) and the timestamp of the most recent heartbeat.

- **POST /heartbeat** – Accepts a JSON payload from the PwnBoxer device. When a valid request is received, the service records the current time as the last heartbeat and returns a JSON response confirming success.

Internally, the application maintains a variable `lastHit` that stores the timestamp of the most recent valid heartbeat. When a client requests the status page, the server computes the difference between the current time and `lastHit`. If the last heartbeat was received within a configurable window (e.g., twenty minutes), the status bar is rendered as green; otherwise, it is rendered as red and the page indicates that either no heartbeat has ever been received or that the device has been offline for an extended period.

To ensure that timestamps presented on the status page match the local expectations of the development and evaluation environment, the server's system timezone was set to `America/New_York` using the `timedatectl` utility. This guarantees that the displayed timestamp corresponds to Eastern Time rather than UTC, simplifying debugging and demonstration.

7.3.7 Authenticated API and Token Handling

While the heartbeat service is intentionally minimal, it is still exposed over the public Internet and therefore requires basic authentication. For this prototype, a shared secret token is used to gate access to the `/heartbeat` endpoint.

The `POST /heartbeat` endpoint expects a JSON payload of the form:

```
{
  "token": "pwnBoxer2025!"
}
```

On each request, the service verifies that the provided token matches the configured server-side constant. If the token is valid, `lastHit` is updated to the current time and the server responds with:

```
{
  "success": true,
  "lastHit": "<ISO 8601 timestamp>"
}
```

If the token does not match the configured value, the server returns an HTTP 401 (Unauthorized) response with a JSON error:

```
{
  "success": false,
  "message": "Invalid token"
}
```

Although a static token is sufficient for this phase of the project, the architecture is deliberately designed so that the token mechanism can be replaced with stronger authentication

in the future, such as mutual TLS certificates, device-specific API keys, or OAuth2-based client credentials. In each case, the API contract would remain the same from the perspective of the CM4 orchestration layer: a short authenticated call to `/heartbeat` signals that the device is online and healthy.

7.3.8 Rate Limiting and Abuse Protection

The heartbeat endpoint is not expected to be called in rapid succession. Under normal operation, the PwnBoxer device would send a heartbeat periodically at a relatively low frequency. This usage pattern provides an opportunity to implement simple abuse protections directly in the application layer.

To prevent a single source from spamming the endpoint, the Node.js service enforces a per-IP cooldown period of two minutes. The application maintains an in-memory map from client IP address to the timestamp of the most recent accepted heartbeat. Before processing a new `/heartbeat` request, the server checks whether the same IP has sent a valid request within the last 120 seconds. If so, the request is rejected with an HTTP 429 (Too Many Requests) response and a descriptive JSON error message.

Because the service is typically deployed behind a reverse proxy, it uses the `X-Forwarded-For` header provided by Apache to determine the actual client IP. If this header is not present, the Express `req.ip` field is used as a fallback. This ensures that rate limiting is applied to the originating client rather than the proxy itself.

This basic rate limiting mechanism is intentionally simple, but it is sufficient to prevent misuse of the endpoint during demonstration and to serve as a foundation for more advanced quotas or per-device policies in a production deployment.

7.3.9 Process Management with PM2

To keep the Node.js service running reliably and to simplify operational management, the reporting server uses PM2 as a process supervisor. PM2 is a popular Node.js process manager that can automatically restart applications on crash, collect runtime logs, and persist process definitions across reboots.

The deployment workflow for the heartbeat service on the VM consists of:

1. Installing the application dependencies in the `heartbeat-app` directory using `npm install`.
2. Starting the service under PM2 with a descriptive process name:

   ```
   pm2 start index.js --name heartbeat
   ```

3. Verifying that the process is online using:

   ```
   pm2 status
   ```

4. Saving the PM2 process list and enabling startup scripts so that the heartbeat service is automatically restarted on reboot:

```
pm2 save
pm2 startup systemd
# (followed by the command printed by PM2)
```

With this configuration, the reporting server can be rebooted or experience transient failures without manual intervention. PM2 will re-launch the heartbeat service and maintain its logs in a centralized location accessible through `pm2 logs`. This behavior aligns with the project's goal of providing an appliance-like experience rather than requiring ongoing system administration.

7.3.10 Network Exposure, Ports, and Reverse Proxy

The heartbeat service listens internally on TCP port 3000. Exposing this port directly on the public interface would be possible but would complicate future integration with a more feature-rich reporting portal. Instead, the reporting server uses a layered approach:

- The Node.js application binds to `localhost:3000`.
- Apache HTTP Server listens on TCP port 80 on the public interface.
- Apache is configured as a reverse proxy to forward incoming HTTP requests to the Node.js service.

At a conceptual level, this mapping can be summarized as:

```
External client  -->  Apache (port 80)  -->  Node.js heartbeat app (po
```

Using a reverse proxy provides several advantages:

- Future-proofing: The same Apache instance can later route specific paths (e.g., `/reports`, `/portal`) to additional backend services without changing the public entry point.
- Logging and instrumentation: Apache logs can be used in conjunction with `fail2ban` to identify malicious access patterns.
- Centralized HTTP hardening: Modules for timeouts, request size limits, and header normalization can be applied once at the proxy layer.

Although an HTTPS configuration with Let's Encrypt and a custom domain was explored, the current deployment intentionally remains on HTTP because no domain name has been purchased yet. The document notes this as future work: once a domain is acquired and DNS is pointed to the VM, the same Apache instance can be upgraded using Certbot to serve a valid TLS certificate on port 443.

7.3.11 Host Hardening with `fail2ban`

Since the reporting server is accessible from the public Internet over SSH and HTTP, it is crucial to protect it against basic brute-force attacks. To that end, the VM is configured with `fail2ban`, an intrusion-prevention framework that monitors log files and bans IP addresses that exhibit suspicious behavior.

The primary jail configured for this project is the SSH daemon (`sshd`) jail. The `/etc/fail2ban/jail` file includes:

```
[sshd]
enabled  = true
port     = ssh
filter   = sshd
logpath  = /var/log/auth.log
maxretry = 5
bantime  = 120
```

This configuration instructs `fail2ban` to:

- Monitor the SSH authentication log at `/var/log/auth.log`.
- Allow up to five failed login attempts from a given IP before taking action.
- Ban offending IPs for 120 seconds (two minutes) to prevent rapid brute-force attempts while reducing the risk of long-term lockout during testing.

After modifying the configuration, the service is reloaded via:

```
sudo systemctl restart fail2ban
```

The overall status, as well as individual jails, can be inspected using:

```
sudo fail2ban-client status
sudo fail2ban-client status sshd
```

In addition to the explicit `sshd` jail, the system's `fail2ban` configuration also enables built-in jails such as `apache-badbots` and `apache-noscript`, which monitor Apache logs for signatures associated with malicious crawlers or requests for non-existent script files. These jails provide a lightweight layer of application-aware defense on top of the firewall.

If a legitimate developer or technician is accidentally banned during testing, they can be unbanned by issuing:

```
sudo fail2ban-client set sshd unbanip <IP_ADDRESS>
```

from a trusted location or via the cloud provider's console.

7.3.12 Firewall Configuration

To further minimize the attack surface, the reporting server's host firewall is configured using the Uncomplicated Firewall (UFW). The firewall policy is intentionally restrictive, allowing only the services that are strictly necessary for operation and debugging:

- SSH access for remote administration.
- HTTP traffic on port 80 for the heartbeat status page and API.

The configuration is applied via:

```
sudo ufw allow OpenSSH
sudo ufw allow 80/tcp
sudo ufw enable
```

The firewall state can be verified using:

```
sudo ufw status
```

In this configuration, unsolicited traffic on all other ports is dropped. This ensures that even if additional services are accidentally started on the VM, they will not be reachable from the Internet without an explicit firewall rule. When HTTPS is later introduced on port 443, an additional allowed rule can be added without impacting the rest of the ruleset.

7.3.13 HTTP-Level Protections and Slowloris Mitigation

While the core rate limiting for `/heartbeat` is implemented at the application layer, the server also employs basic HTTP-level protections through Apache modules. In particular, the `reqtimeout` module is enabled to defend against slowloris-style attacks, where a client attempts to hold connections open by sending HTTP headers or bodies extremely slowly.

Enabling the module is straightforward:

```
sudo a2enmod reqtimeout
sudo systemctl restart apache2
```

The default configuration in `/etc/apache2/mods-available/reqtimeout.conf` specifies time limits for receiving request headers and bodies. If a client fails to send data within these timeouts, Apache terminates the connection, preventing a small number of slow clients from exhausting server resources.

Although `reqtimeout` is not a full-featured web application firewall, it provides an important baseline safeguard against a class of resource-exhaustion attacks at essentially no complexity cost. Combined with the per-IP rate limiting and `fail2ban` jails, it forms a layered defense suitable for the project's scale and threat model.

7.3.14 Monitoring, Logging, and Observability

Observability for the reporting server is intentionally lightweight but sufficient for debugging and evaluation:

- **PM2 logs** capture stdout and stderr output from the Node.js heartbeat service. These logs include startup messages, request traces added for debugging, and any runtime errors.
- **Apache access and error logs** record incoming HTTP traffic to the reverse proxy, including client IPs, endpoints, and HTTP response codes. These logs are used by `fail2ban` jails (`apache-badbots`, `apache-noscript`) to detect suspicious activity.
- **System logs** (via `journalctl`) provide insight into service restarts, SSH logins, and `fail2ban` actions.

Together, these logs allow developers to trace the full path of a heartbeat: from the PwnBoxer device's `curl` command, through Apache, into the Node.js application, and finally into the in-memory `lastHit` state and status page.

7.3.15 Testing Strategy and Validation

The reporting server configuration was validated through a series of manual tests designed to confirm both functionality and security controls:

1. **Status page reachability:** From within the VM, a simple `curl http://localhost:3000/` confirmed that the Node.js service responded. From an external client, navigating to `http://<server_ip>/` via a web browser verified that Apache correctly proxied requests to the heartbeat service.
2. **Heartbeat success path:** A valid heartbeat request was issued from the VM:

   ```
   curl -X POST http://localhost:3000/heartbeat \
     -H "Content-Type: application/json" \
     -d '{"token": "pwnBoxer2025!"}'
   ```

   The JSON response was inspected to confirm `"success": true`, and the status page updated to a green bar with the new timestamp.
3. **Authentication failure:** The same request was repeated with an incorrect token, and the service correctly returned HTTP 401 with a failure message.
4. **Rate limiting:** Two valid heartbeat requests were sent in rapid succession from the same client. The first request succeeded; the second returned HTTP 429 with the "Too many requests, wait 2 minutes" message, confirming that the cooldown logic was applied.
5. **Fail2ban behavior:** SSH was intentionally misused from a test client by entering invalid passwords more than five times. The `fail2ban-client status sshd`

command showed the test IP as banned, and SSH connections from that address were blocked until the ban expired or the IP was manually unbanned.

6. **Firewall enforcement:** The `ufw status` output was reviewed to ensure that only ports 22 and 80 were open, and scans from external tools confirmed that no other ports were reachable.

These tests demonstrate that the reporting server not only fulfills its functional role as a heartbeat and status endpoint, but also enforces basic but effective defensive controls appropriate for a prototype exposed to the Internet.

7.3.16 Limitations and Future Work

The current reporting server implementation focuses on a reliable heartbeat mechanism and foundational security controls. Several limitations and planned enhancements are noted for future work:

- **Plain HTTP transport:** The server currently uses unencrypted HTTP on port 80 because no domain has been purchased. Once a domain is acquired and pointed to the VM, the plan is to integrate Let's Encrypt via Certbot and migrate the service to HTTPS on port 443.
- **Static shared secret:** Authentication for the heartbeat API relies on a single hard-coded token. A future revision will introduce device-specific credentials, rotation procedures, or mutual TLS to provide stronger guarantees of authenticity.
- **In-memory state only:** The `lastHit` value and rate-limiting map are stored solely in memory. If the Node.js process restarts, this state is reset. Persisting key metrics to disk or a lightweight database would allow for historical availability analysis.
- **Limited reporting functionality:** At present, the server does not ingest or summarize full scan results. The architecture is intentionally designed to be compatible with a future pipeline in which the CM4 uploads log artifacts, an LLM or other analysis engine produces a human-readable report, and the results are distributed via email and a web-based portal.

Despite these limitations, the current design meets the project's immediate objectives: it provides a secure, observable, and extensible foundation for monitoring PwnBoxer devices in the field while leaving clear upgrade paths toward a fully featured reporting platform.

7.4 Technician Connectivity

Overview The device is intended to be maintainable in the field; technicians must be able to perform software updates, add or modify task configurations, and perform troubleshooting on the CM4. Technician access is therefore supported as a controlled capability of the system. Access is initiated from the user-facing ESP32 interface (menu-driven) and executed by the CM4 task scheduler.

7.4.1 Supported Access Methods

The system supports multiple remote access methods to accommodate different operational requirements and toolchains. These are treated as interchangeable `access_type` values within the task configuration schema:

- **Authenticated SSH:**SSH session using key/password based authentication.
- **Reverse shells:** Reverse shells initiated by the CM4 using pre-configured sessons.

7.4.2 Configuration and Initiation Flow

Technician access is modeled and executed as a specialized task (for example, `TYPE_REVERSE_SHELL` or `TYPE_SSH_SHELL`) under the existing task framework. The high-level sequence is:

1. **Operator input (ESP32 UI):** The technician connection is initiated via the ESP32 menu. The operator supplies required parameters (listener IP, listener port, and access type). The UI validates the parameter format locally before packaging the request.
2. **Task generation:** The ESP32 constructs a task packet with a well-defined header (e.g., `TYPE_REVERSE_SHELL`) and a protected payload containing the connection parameters and operator identity.
3. **Scheduler validation:** On receipt the CM4 scheduler validates the packet signature. If validation fails, the task is rejected and an audit event is recorded.
4. **Payload assembly:** If the task is authorized, the scheduler selects an approved payload template from a repository.

The reporting subsystem is built on an Apache2 and Ollama-based stack designed to process, analyze, and distribute system reports generated by the CM4. Its primary function is to collect test results and task outputs from the device, process them using an AI-driven pipeline, and generate synthesized, human-readable reports for review and archival.

The following is an overview of the reporting server workflow.

1. Secure data transfer from the CM4 via authenticated API requests.
2. Intelligent report synthesis using LLM-based retrieval-augmented generation.
3. Automated email delivery through Postmark.
4. A web-based dashboard for report access, log management, and user authentication.
5. Robust data handling and audit logging to ensure traceability and integrity.

7.4.3 Report Submission and Processing Pipeline

When the CM4 completes a workflow or task sequence, it transmits a data packet to the reporting server containing a header labeled `REPORT_BEGIN`. Upon receiving this signal, the reporting listener service authenticates the request and initiates a secure upload session via an API endpoint protected with authentication tokens.

During this stage, all associated log and report files are transferred from the CM4 to the reporting server. Each file is verified, logged, and then passed to the report ingestion module. This ingestion process prepares the data for higher-level analysis and synthesis.

Once the upload is complete, the server issues an API `POST` request to the Ollama processing service. The payload includes both the uploaded artifacts and a structured prompt instructing the large language model (LLM) to perform a synthesis task using *retrieval-augmented generation (RAG)*. The LLM processes the contextual data, extracts key findings, and produces a summarized report in HTML format. This output is designed to be both human-readable and suitable for automated email delivery and web-based presentation.

7.4.4 Report Distribution and Notification

After the HTML report is generated, it is formatted for email delivery using the Postmark email API. The finalized report is then sent to designated recipients and archived on the reporting server for later retrieval.

Concurrently, the same report is made accessible via a secure web portal, allowing users to log in, view historical reports, and download generated artifacts. The portal also provides the ability to clear or archive older logs for storage management.

7.4.5 Web Portal and User Interface

The web interface is implemented as a responsive front end built with HTML, CSS, and JavaScript, supported by a Node.js backend running on the Express.js framework. The interface provides a clean and intuitive dashboard for report viewing and log management.

Authentication and user management are handled via a prebuilt authentication library integrated into the Express.js backend. Credentials (usernames and salted password hashes) are stored in a local MariaDB database. Passwords are hashed using the SHA-256 algorithm with unique salts to prevent rainbow-table attacks and unauthorized credential recovery.

7.4.6 Data Management and Logging

All user interactions and report-related operations are logged for auditing and traceability. Logs are stored locally on the reporting server, allowing administrators to generate vulnerability reports and possible mitigations. Users can download specific logs directly through the web portal, which retrieves the data from the local filesystem and delivers it as a download.

7.4.7 Reporting Server Workflow

This graphic illustrates the reporting work and data flow as processed through the reporting server.

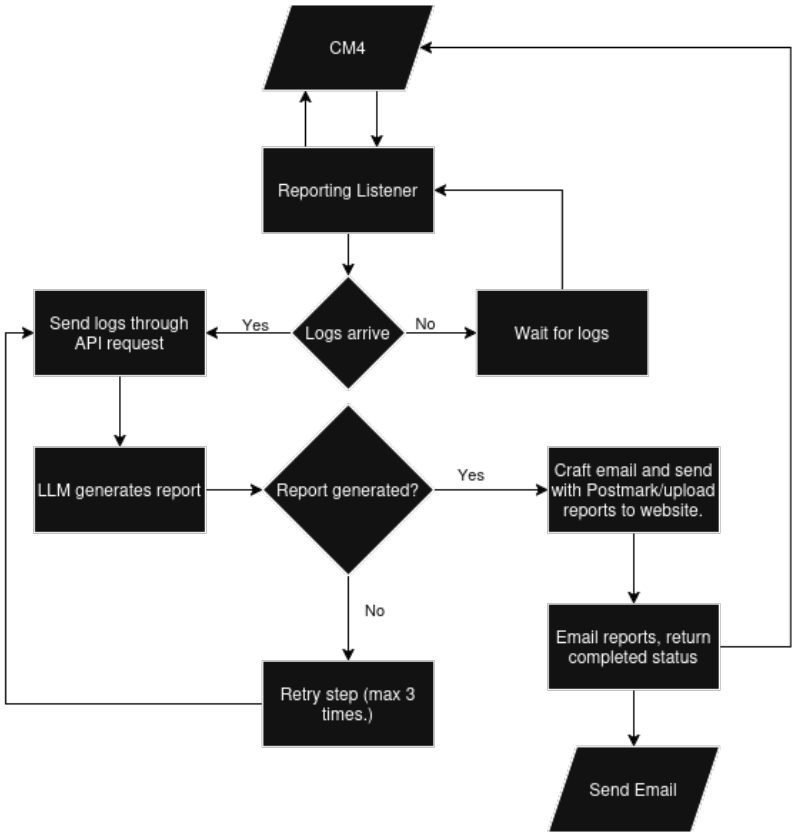


Figure 15: Remote Server Workflow Chart

8 System Fabrication and Prototype Construction

8.1 Overview

This chapter describes the fabrication process, prototype construction, and hardware integration performed during the development of the PwnBoxer system. The PCB, power delivery subsystem, compute modules, and display interface were assembled and validated through staged prototyping and subsystem testing.

8.2 PCB Layout and Mechanical Design

The PCB was designed in KiCad and integrates all major subsystems, including the Raspberry Pi Compute Module 4 (CM4), ESP32 microcontroller, power regulation circuitry, TFT display interface, USB connectivity, and debug interfaces. High-speed differential pairs, power planes, and length-matched communication buses were routed following manufacturer guidelines.

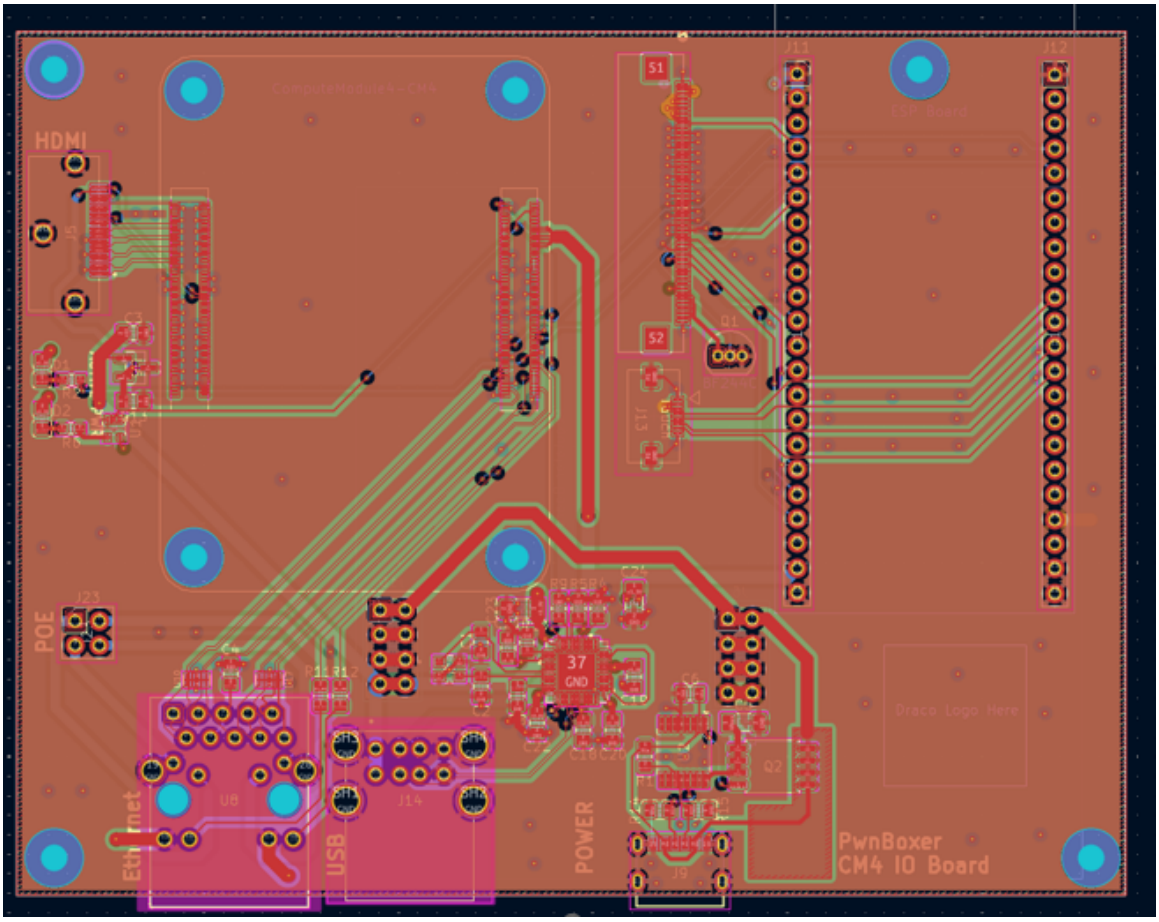


Figure 16: PCB layout of the PwnBoxer prototype.

Even though traces are not currently displayed, the KiCad layout images highlight component placement, mechanical alignment, and layer stackup. Mechanical constraints such as connector placement, standoffs, and display mounting will be considered to ensure system accessibility and assembly reliability.

8.3 Power Delivery Subsystem

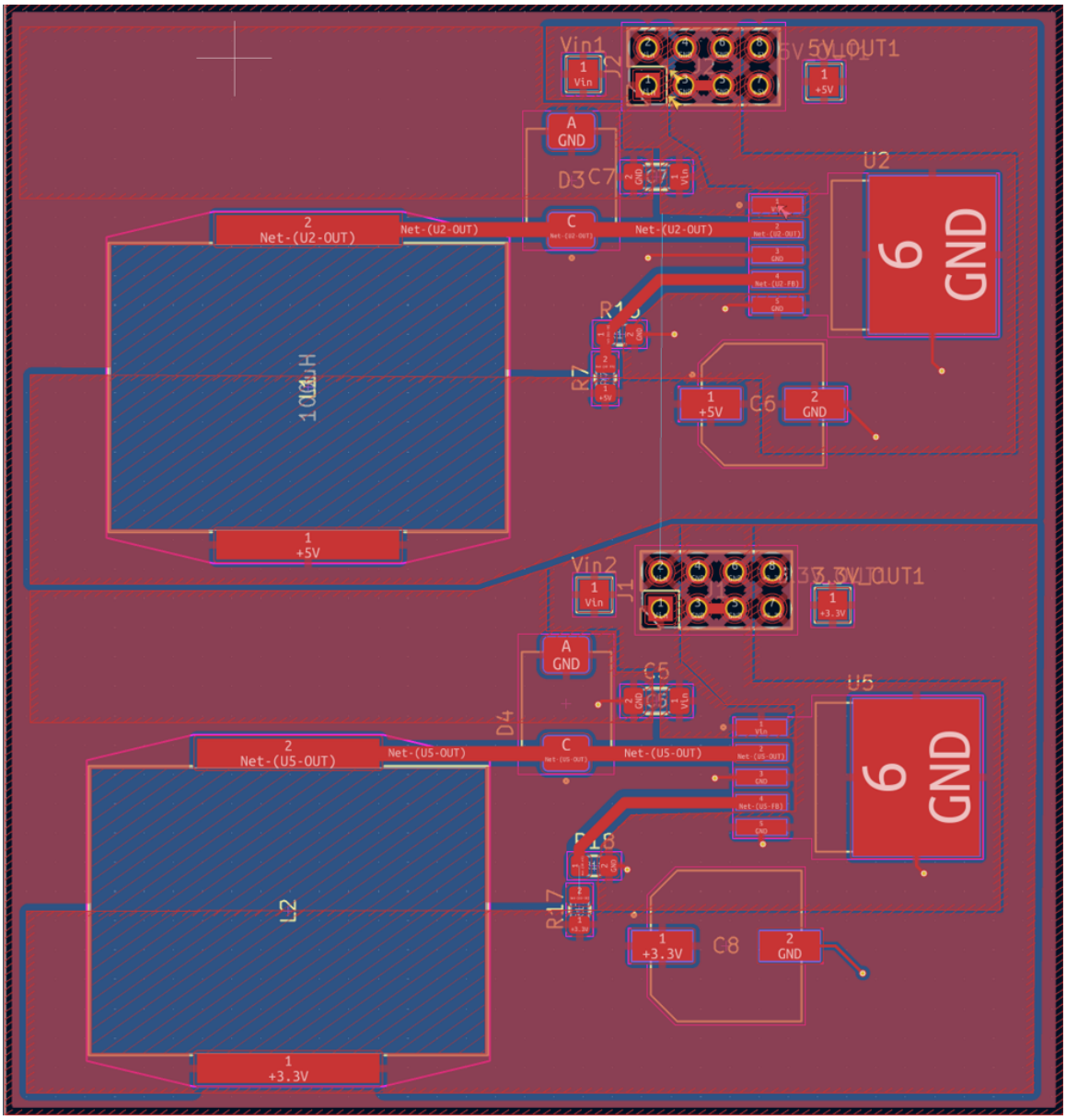


Figure 17: Power Delivery Subsystem - KiCad PCB

8.3.1 USB-C Power Input

The system utilizes a single USB-C connector as the dedicated power input. No data lanes are routed; only VBUS, CC resistors, and ground are implemented. The USB-C adapter used during testing supplied:

$$12V@1.67A \approx 20W$$

This input rail feeds the boost converters and downstream 5 V and 3.3 V rails.

8.3.2 Buck Converters (LM2576)

The design uses Texas Instruments LM2576 step-down (buck) voltage regulators..

The LM2576 was selected for its:

- high efficiency in stepping down voltage,
- integrated switching regulator minimizing external component requirements,
- ability to provide stable output under varying load conditions,
- suitability for battery- and USB-powered embedded systems.

8.3.3 Bench Verification

Prototype regulator circuits were constructed on a breadboard to validate performance. A bench power supply was used to sweep the input voltage from 7 V to 12 V while monitoring output stability using multimeters. The converters maintained a stable regulated output across the tested range.

Additional verification was performed by powering the circuit through a USB-C supply, representative of the final system configuration. Output voltage was measured to confirm expected regulation performance prior to PCB integration.

8.4 CM4 Subsystem

[H]

8.4.1 High-Speed Interfaces

The CM4 high-speed connector routes three primary interfaces:

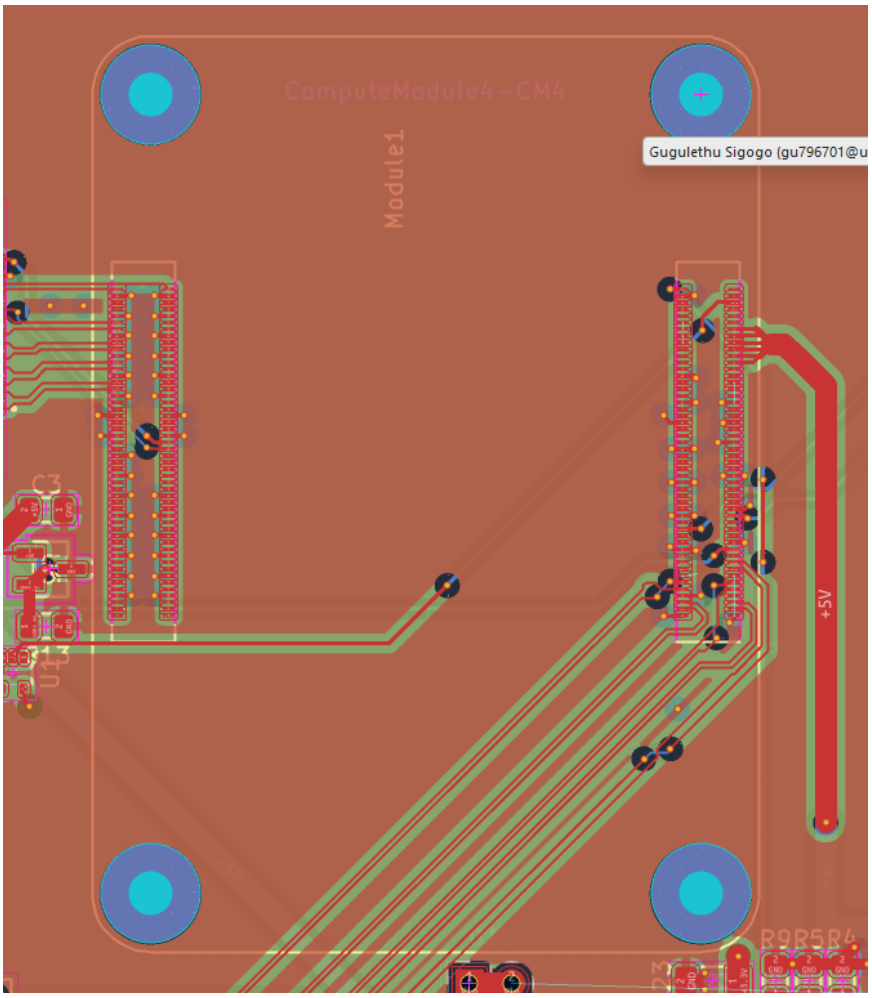


Figure 18: CM4 Header - KiCad PCB

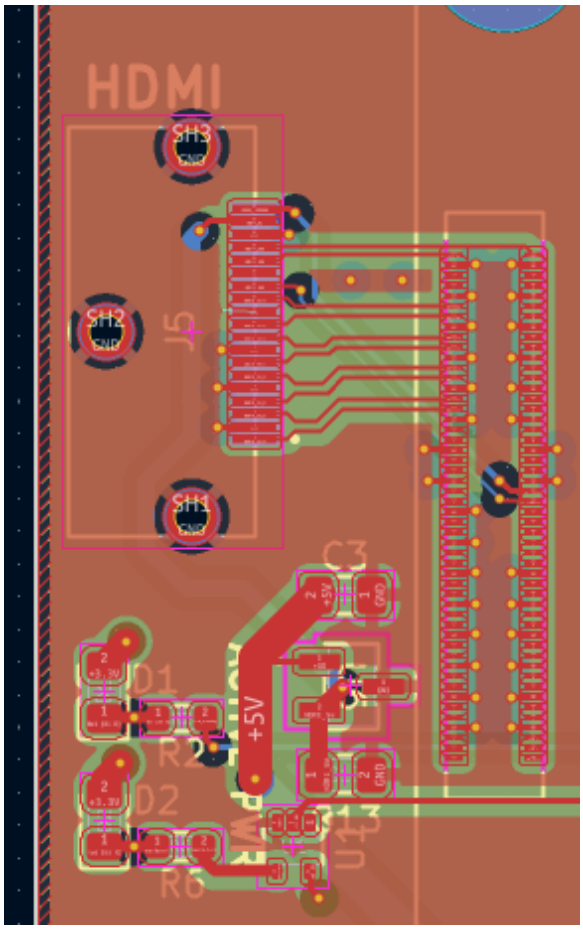


Figure 19: HDMI - KiCad PCB

- An HDMI port (HD12-19-SMT-TR) is connected to the CM4. The 5 V HDMI power pin is protected using an RT9742SNGV power distribution switch, providing:
  - inrush current limiting,
  - controlled turn-on behavior,
  - short-circuit and over-current protection.
- The CM4 USB 2.0 interface feeds the USB2512B-I/M2 hub, which drives a stacked dual USB Type-A connector (72309-7043BLF). Length matching and impedance control ($\sim$90 Ω differential) were applied to the USB data pairs per the CM4 design guide.

8.4.2 Low-Speed Interfaces

The CM4 low-speed connector includes:

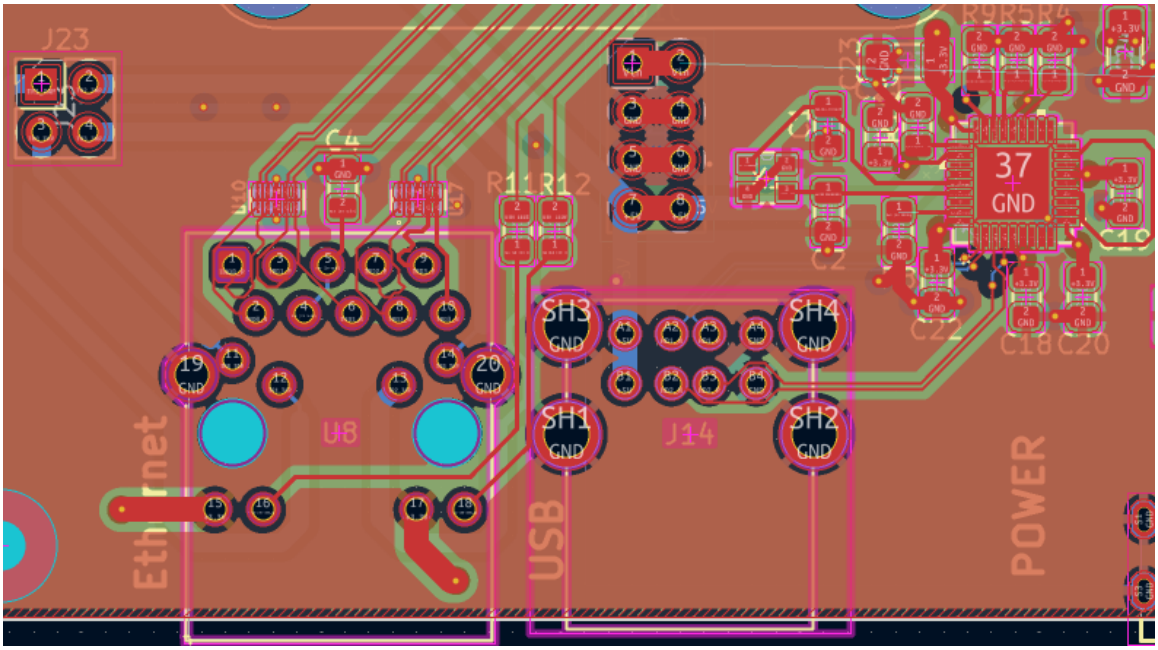


Figure 20: Ethernet and USB - KiCad PCB

- Gigabit Ethernet (JK0-0177NL) with AutoMDIX support,
- Jumpers for EEPROM write protection and forcing RPIBOOT mode,
- A wake-from-shutdown tactile button tied between pins 13–14.

When the CM4 is awake, `RUN_PG` is high and the button has no effect. When in shutdown, `RUN_PG` is low and the button pulls global enable low to wake the system.

8.5 ESP32 Subsystem

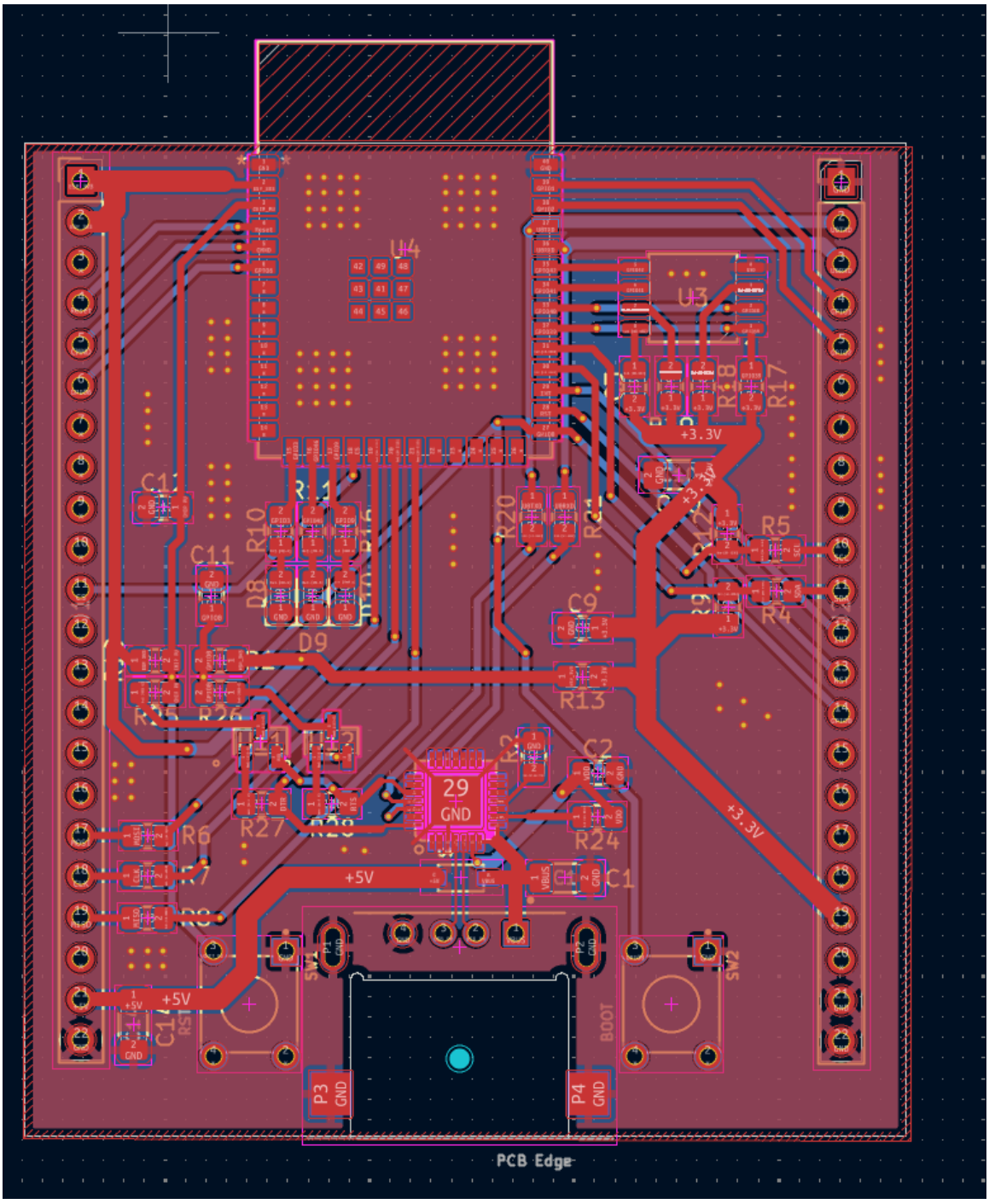


Figure 21: ESP Subsystem - KiCad PCB

8.5.1 Programming Interface

The ESP32 includes a dedicated Micro-USB port used exclusively for:

- USB-to-UART programming,
- firmware upload,
- serial debugging.

This interface is electrically isolated from the USB-C power input to avoid backfeed conditions.

8.5.2 Power and Functionality

The ESP32 is powered from the system 3.3 V rail and serves as the controller for the TFT display subsystem. It handles user interface logic, touch sensing, and display updates.

8.5.3 External Storage

The ESP32-S3 is used for peripheral control and real-time interfacing within the system. To support additional non-volatile storage requirements, an external W25Q32JV SPI flash memory (4 MB) is integrated into the design. This provides dedicated storage for firmware, configuration data, and system parameters beyond the internal memory of the ESP32.

8.6 TFT Display Subsystem

The CFAF320480C7-035TC TFT display is driven by the ESP32 via an SPI interface. Control lines include:

- `WR`, `RD`, `CS`, `RS`,
- SPI,
- I^2C interface for capacitive touch.

The LED backlight is powered by the 5 V rail. High-current traces and adequate decoupling were used to support backlight transients. All timing-critical display signals were routed on the top layer to minimize skew.

8.7 Prototype Assembly

The assembled prototype of the PWNBoxer system integrates all major hardware subsystems into a functional embedded platform. The system consists of a Raspberry Pi Compute Module 4 (CM4) mounted on a carrier board, an ESP32-S3-based microcontroller board, dedicated power regulation modules, and a custom interface board that enables communication between subsystems and external peripherals.

The ESP32 board serves as the peripheral control unit and includes an onboard W25Q32JV SPI flash memory (32 Mbit / 4 MB) for user data and configuration storage. It also integrates three status indicator LEDs—white, green, and red—which provide visual feedback for system states, including processing, successful scan completion, and unsuccessful scan conditions.

The CM4 carrier board functions as the primary processing platform, handling high-level computation, network communication, and data analysis. Supporting this subsystem are dedicated 3.3 V and 5 V regulator boards, which provide stable voltage rails for both the CM4 and ESP32 subsystems.

A custom interface board serves as the central interconnect for the system. This board connects to the CM4 carrier board through a 40-pin header and facilitates communication with peripherals and the ESP32 subsystem. The interface board includes:

A 50-pin ZIF connector for SPI-based communication with the display An 8-pin ZIF connector for I²C communication with the touch interface A 44-pin header connecting to the ESP32 board A USB-C power input for system power delivery

This integrated design allows the system to support real-time peripheral interaction through the ESP32 while leveraging the CM4 for high-performance processing and network operations. The modular structure of the prototype also enables ease of testing, debugging, and future expansion.

Figure 22: Front view of the PWNBoxer prototype showing the embedded display interface used for user interaction, task execution, and real-time system feedback.

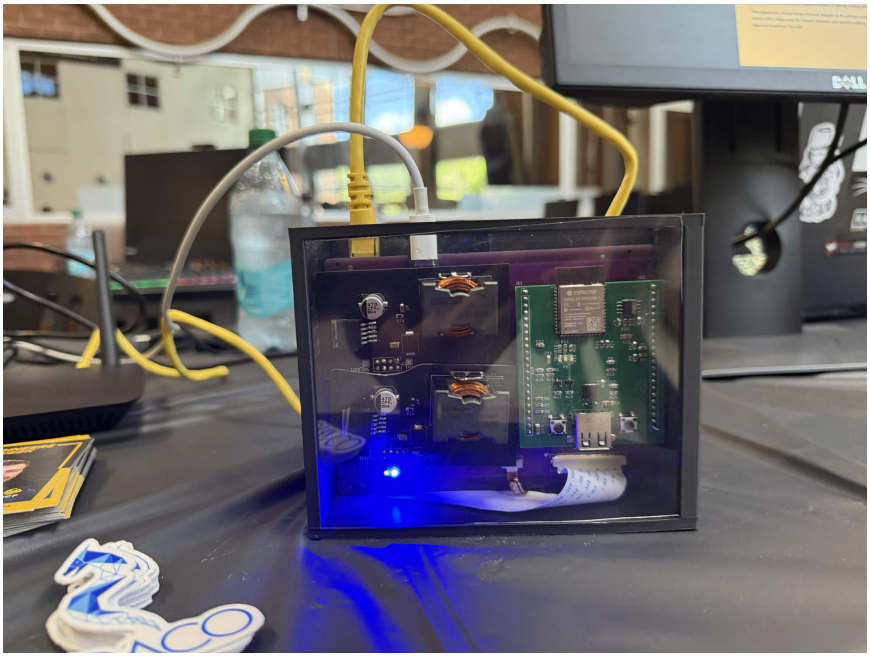


Figure 23: Rear view of the assembled PWNBoxer prototype illustrating internal subsystem integration, including the ESP32-S3 board with external W25Q32JV SPI flash memory, power regulation boards, and interconnections within the enclosure.

9 System Testing and Evaluation

9.1 Hardware Testing

The hardware testing phase for the PwnBoxer system focused on validating the core electrical elements that could be evaluated prior to PCB fabrication. Since the final PCB has not yet been manufactured, subsystem-level testing on the complete board could not be performed. Instead, the development team conducted extensive benchtop testing of critical circuits, including the updated regulator network and the USB-C power input path, to ensure that the fundamental electrical design choices were sound before committing to fabrication.

9.1.1 Power Regulation Testing (Pre-PCB Validation)

Prior to integration into the PCB design, the power regulation circuits were reconstructed on a breadboard to validate their behavior under real electrical conditions. The updated design makes use of LM2576-based switching regulators for the primary voltage rails. Because these regulators are central to every subsystem in the PwnBoxer, they were the primary focus of early hardware testing.

A programmable DC bench supply was used to power the breadboard circuit. Input voltage was swept from **7 V to 12 V** to simulate worst-case operating conditions and ensure that the regulators maintained correct output voltage across a wide input range. At the same

time, a calibrated digital multimeter monitored the output to verify that the regulated rails remained stable.

Across the entire 7–12 V sweep, the output voltage remained constant, demonstrating:

- Stable regulation under varying input conditions,
- No measurable output oscillation or dropout,
- Output ripple consistent with datasheet expectations,
- Reliable start-up behavior even at the low end of the input range.

These results confirmed that the regulator configuration was correctly implemented and suitable for PCB integration.

USB-C Power Input Testing Because the final device will be powered exclusively through a single USB-C connector, additional validation was performed using a 12 V, 1.67 A USB-C power brick. This test ensured that the regulator circuitry behaves correctly when using the same type of power source intended for final operation.

When powered through USB-C, the breadboard circuit continued to produce stable, expected output voltages. This confirmed the following:

- The USB-C source provides adequate current and voltage stability,
- No harmful voltage drops occur across the connector or cabling,
- The regulator startup behavior is unaffected by the USB-C supply,
- The USB-C input path is electrically sound for final PCB implementation.

This testing validated the entire power delivery chain from USB-C input to regulated output rails.

9.1.2 Limitations of Pre-Fabrication Testing

Because the PCB has not yet been manufactured, the following elements could not be tested at this stage:

- USB2512B USB hub enumeration and behavior on the CM4,
- HDMI signal integrity through the high-speed connector,
- CM4 boot behavior on the final power routing,
- ESP32 interface routing performance,
- ZIF-connected TFT display operation (parallel bus and capacitive touch),
- Power distribution switch (RT9742SNGV) performance in the final layout,
- Crosstalk, EMI, and high-speed signal layout effects.

These tests will be completed after PCB fabrication and assembly.

9.1.3 Planned PCB-Level Testing

Comprehensive testing plan for after fabrication. The planned tests include:

1. Power Delivery Tests

- Measurement of all voltage rails under load,
- Verification of inrush current behavior,
- Thermal monitoring of the LM2576 regulators,
- Brownout testing of both the CM4 and ESP32 subsystems.

2. CM4 Subsystem Tests

- Boot verification using RPIBOOT and onboard EEPROM,
- Ethernet link negotiation and AutoMDIX behavior,
- USB hub enumeration and performance validation,
- HDMI output detection and resolution switching.

3. ESP32 Subsystem Tests

- Firmware flashing over Micro-USB,
- Serial console stability testing,
- Validation of GPIO timing on the TFT display parallel interface.

4. TFT Display Tests

- Verification of ZIF connector pin integrity,
- Parallel data bus timing validation,
- Capacitive touch controller communication test,
- Full display refresh and color correctness evaluation.

5. Integration and Stress Testing

- Full system power-on test,
- Combined CM4 + ESP32 + display load testing,
- Thermal soak testing,
- Long-duration runtime stability test.

9.1.4 Summary of Pre-Fabrication Validation

Although the full PwnBoxer hardware system could not yet be powered and tested as a complete unit, the preliminary electrical testing performed on the regulator network and USB-C power path provides high confidence in the electrical foundation of the design. These early tests significantly reduce the risk of PCB-level failures and provide a strong baseline for the more advanced subsystem tests that will be conducted once the PCB is fabricated.

9.2 Software Testing

The Pwnxoxer system integrates multiple hardware components into an embedded platform through several interconnected software subsystems. The system comprises the CFAF320480C7-035TC display module, ESP32-S3 microcontroller, and Raspberry Pi Compute Module 4, which communicate through dedicated software interfaces. The primary subsystems include a display driving and rendering interface, a cross-board communication interface, a WiFi authentication interface, and a security tooling framework. To ensure reliable operation, each subsystem contains testing units to confirm that the connected systems are working on and throughout operations. The following sections detail the testing methodologies and results for each subsystem, demonstrating their individual performance and collective integration within the complete system.

9.2.1 Display Testing - Driver Software

The driver software for the TFT display underwent several iterations to achieve preliminary test output on the display and to optimize SPI speed for acceptable screen rendering performance. Initially, a custom SPI driver was written to confirm that the screen was adequately connected to the hardware. One of the main challenges with existing driver libraries is that the ESP32-S3 is not well supported by some implementations. By writing a custom driver, we were able to debug potential issues at each level: hardware connection points, driver initialization, command/data transmission, and ESP32-S3 pin configuration. Using Direct Memory Access (DMA), the custom driver achieved 25-30 frames per second (FPS) on the initial screen coloring test, which fell within our target range. Once we confirmed that the screen was properly connected, we transitioned to implementing LovyanGFX as a PlatformIO project, which came prebuilt with the ESP32-S3 libraries and configuration required for operation. With the same test, we maintained the target performance of 25-30 FPS for the color test. The display implementation utilizes the hardware-designated GPIO pins for the ESP32-S3 SPI peripheral. While the MCU is capable of multiplexing pins to allow connectivity through any GPIO, using the default SPI pins contributed to increased throughput and enabled the use of DMA for additional performance.

9.2.2 Display Testing - Interface Rendering

The Pwnboxer's interface is built using the Light and Versatile Graphics Library (LVGL). LVGL is a hardware-agnostic library that can be implemented with any microcontroller, provided the required drawing and rendering functions are properly registered with the

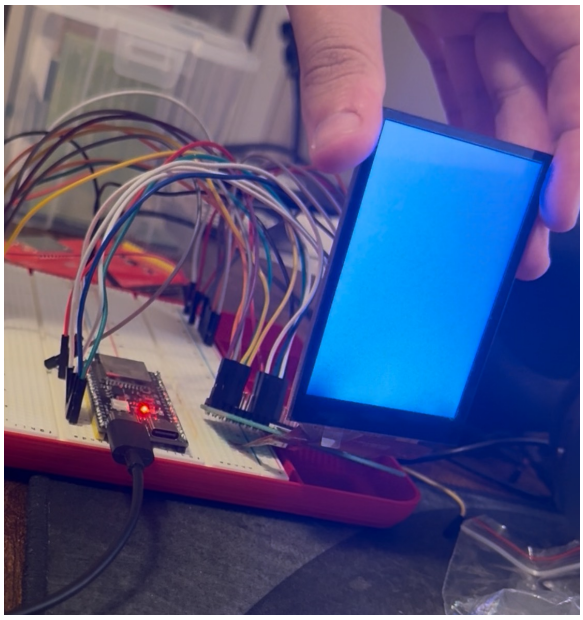


Figure 24: Initial Color Testing

framework. LVGL was utilized to implement a widget-based user interface featuring horizontal scrolling, tab-based windows, and interactive buttons. Within the UI tab of the settings menu, two buttons were integrated for testing WiFi connectivity and inter-board communications in subsequent test phases.

When initially implementing LVGL, the ESP32-S3 experienced significant performance degradation on the rendering task, achieving only approximately 10 FPS when animations were active. To mitigate these performance issues, several optimization techniques were employed. DMA-capable double draw buffers were implemented, utilizing two LVGL virtual display buffers (VDBs) as line buffers in internal SRAM. This configuration allows the CPU to draw to one buffer while SPI DMA simultaneously transmits the previous buffer to the display. The line buffers were sized to accommodate multiple scan lines while remaining within memory constraints, with 32-byte alignment enforced to optimize DMA transfers and cache performance. This approach reduces the frequency of flush calls and maximizes transfer efficiency.

A non-blocking flush mechanism was implemented by initiating SPI DMA transfers in the display flush callback and signaling completion only when the DMA operation finishes, eliminating busy-wait cycles and reducing screen tearing artifacts. The LovyanGFX library was configured to operate SPI2 with DMA at a 60 MHz write clock, with automatic DMA channel allocation and exclusive bus access to prevent contention. Byte ordering was corrected by removing manual RGB565 byte swapping and allowing LovyanGFX to handle color format conversion natively, preventing unnecessary memory copies and color rendering glitches.

Partial screen invalidation was leveraged through LVGL's dirty-area refresh mechanism, updating only modified regions rather than performing full-screen redraws. For scrolling

operations, only the affected display bands were updated to minimize data transfer overhead. Color depth was set consistently to 16-bit RGB565 throughout the rendering pipeline to reduce memory bandwidth requirements and eliminate unnecessary color space conversions. The LVGL task handler was scheduled at regular 5-10 ms intervals on a separate CPU core to ensure smooth animation frame timing. Additionally, unnecessary features such as excessive font variants, gradients, and complex blending operations were disabled to reduce computational overhead. A zero-copy rendering path was established where LVGL renders directly into DMA buffers that are subsequently handed to the display driver, eliminating intermediate buffer copies.

9.2.3 Display Testing - Touch Display

The CFAF320480C7-035TC TFT display features a capacitive touch interface that communicates with the ESP32-S3 via the I2C protocol. The touch interface functioned primarily out of the box; however, calibration was required to achieve accurate touch coordinate mapping. The calibration process was necessary because initial testing revealed that touch inputs were significantly offset from the rendered UI elements, with button presses registering far from the visual center of the objects being displayed.

The touchscreen calibration was performed by capturing touch coordinates transmitted to the ESP32-S3 at nine distinct reference points distributed across the screen: top-left, top-center, top-right, middle-left, center, middle-right, bottom-left, bottom-center, and bottom-right. The raw coordinate values returned by the touch controller at each of these positions were recorded through the serial console and subsequently hardcoded into the touchscreen initialization routine. This approach ensures that calibration is applied automatically on every system boot, providing more consistent and reliable touch response compared to storing calibration data in NVS flash, which could be susceptible to corruption or unintended modification.

To verify that the touch interface was functioning correctly post-calibration, extensive testing was conducted with several LVGL widgets, including navigation tabs, interactive buttons, and the on-screen keyboard. Testing confirmed that after calibration, touch inputs registered nearly centered on the corresponding rendered objects, dramatically improving system usability. The enhanced accuracy was particularly evident with the keyboard widget.

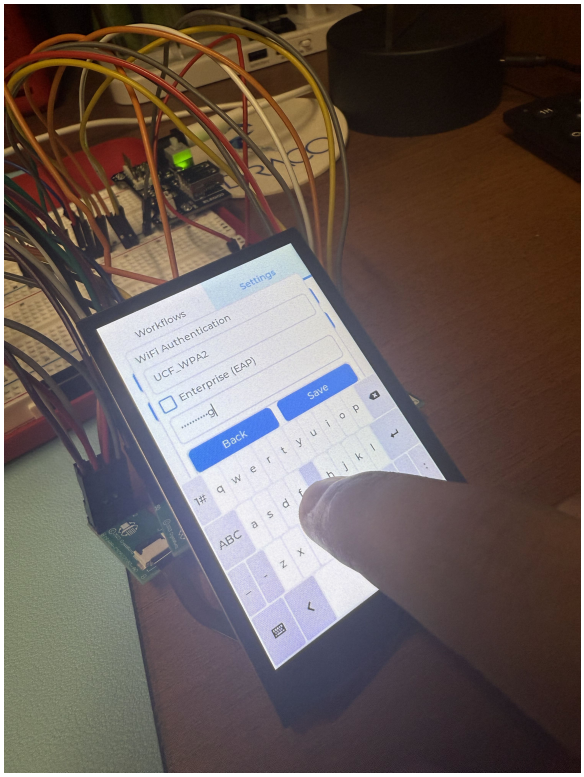


Figure 25: Touch Screen Test

9.2.4 Wireless Connectivity Testing - ESP32-S3

The WiFi subsystem was tested by implementing a user interface that allows users to perform preliminary authentication using the ESP-IDF WiFi stack API. The UI enables the user to initiate a network scan while operating in station mode, leveraging the `esp_wifi_scan_start()` function to asynchronously discover nearby access points. When the scan completes, the WiFi event handler retrieves the scan results using `esp_wifi_scan_get_ap_num()` and `esp_wifi_scan_get_ap_records()`, which populate an array of `wifi_ap_record_t` structures containing information about each detected network, including SSID, RSSI, authentication mode, and channel configuration.

The scan results are processed through the `can_connect_eval()` function, which evaluates each network's authentication mode to determine compatibility with the system's supported authentication protocols. These results are then passed to the LVGL UI thread using `lv_async_call()` to ensure GUI-safe updates, and a vertical scroll list is populated with entries for each discovered network. When the user selects a network from the list, an authentication dialog is presented that dynamically adjusts its input fields based on the selected network's authentication type. The network stack supports multiple authentication modes, including WPA2/WPA3 Personal and WPA2-Enterprise, with the UI providing appropriate credential entry fields for each type—displaying username and password fields for enterprise networks and a single password field for personal authentication modes.

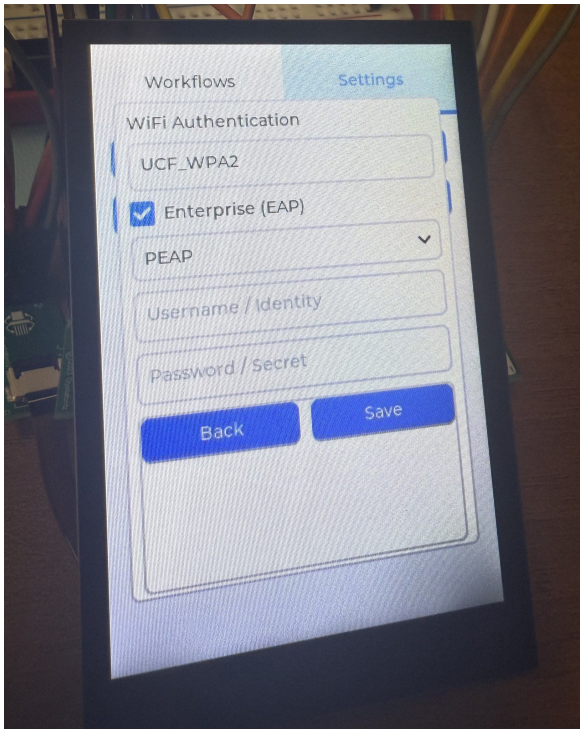


Figure 26: Network Credential Screen - UCF_WPA2

Once the user completes the credential entry and saves their input, the information is stored in the global `wifi_credentials` structure and used to authenticate with the selected WiFi access point. If authentication succeeds, the user receives a confirmation message indicating successful connection. Following successful authentication, the ESP32-S3 deauthenticates from the network and retains the validated credentials in the `wifi_credentials` structure. This credential information is then serialized and transmitted to the Raspberry Pi Compute Module 4 via UART.

Throughout the testing process, console logging was implemented to monitor the WiFi subsystem's operation. The logs capture the complete scan lifecycle, displaying each discovered access point along with its associated metadata such as SSID, signal strength, and authentication requirements. Additionally, the logs record the credential structures as they are populated from user input and transmitted over UART, providing visibility into the data flow between the ESP32-S3 UI layer, WiFi control logic, and cross-board communication subsystem. Logging allowed for verification that the scanning API correctly retrieved network information, the `can_connect_eval()` function properly filtered incompatible networks, and the authentication dialog accurately captured and transmitted user credentials to the CM4 for subsequent connection attempts.

9.2.5 Cross-board Communications Testing

Cross-board communication between the ESP32-S3 and the Raspberry Pi Compute Module 4 is implemented using UART as the primary inter-processor communication protocol. The

interface facilitates bidirectional message exchange for WiFi configuration data, available security tasks, task execution status, and user credential information. The UART protocol was selected due to its reliability, simple implementation, and adequate throughput for the application's requirements, as the system does not demand high-speed data transfer between the two processors. To verify proper communication between the devices, a diagnostic test suite was integrated into the LVGL user interface. This suite enables users to launch connectivity tests that validate UART functionality between the CM4 and ESP32-S3. The currently implemented test transmits a predefined test packet from the ESP32-S3 over UART, which the CM4 receives and acknowledges with an "OK" response. Additional diagnostic tests are planned for future implementation, including a WiFi connectivity verification test on the CM4, a credential transport integrity check to ensure user authentication data is transmitted correctly, and a heartbeat mechanism to monitor the operational status of the CM4 and detect communication failures or system lockups. Additionally, the Azure web interface constantly expects callbacks from the CM4, ensuring that the device is active by requiring a constant ping every 20 minutes.

9.2.6 Security Tooling testing

The security tooling subsystem was evaluated using a simulation-based testing approach within a controlled vulnerable infrastructure. A small test environment was constructed to provide realistic targets for each security tool integrated into the Pwnboxer system. This approach allowed for validation of tool functionality within the CM4 environment without requiring access to live production systems or engaging in unauthorized security testing.

Metasploit Framework modules were tested extensively, including exploit modules, auxiliary modules, and post-exploitation modules. Testing confirmed that the CM4 could successfully execute common attack vectors such as TCP reverse shell payloads and properly handle the resulting sessions. Nuclei was tested against a deliberately vulnerable web server running a web application with known security flaws, verifying that the vulnerability scanner could identify common vulnerabilities such as SQL injection points, cross-site scripting (XSS) vulnerabilities, and misconfigurations. SQLMap was validated by targeting SQL injection fields within the vulnerable test application, confirming that the tool could successfully enumerate databases, extract table structures, and dump sensitive data through automated exploitation. Nmap was tested with its vulnerability detection scripts (NSE) within the simulated network environment, verifying that the CM4 could perform comprehensive network reconnaissance, service enumeration, and automated vulnerability identification across multiple target hosts.

Beyond individual tool validation, the testing phase also verified the modular program architecture designed to launch these tools programmatically. Configuration files based on the structure outlined in Section 7.2.1 were used to define attack parameters, target specifications, and tool options. Testing confirmed that the system could parse these configuration files and dynamically launch the appropriate security tools with the specified parameters, enabling automated and repeatable security assessments.

10 Administrative Content

10.1 Budget

Our team has capped the project budget at $1000, with an expected initial cost of about $215 for core components. The largest expenses will come from the Raspberry Pi Compute Module 4 and IO board, along with the TFT capacitive touchscreen display. The ESP32-S3 microcontroller, 4MB SPI flash memory, and supporting peripherals such as the UART, status LEDs, and pushbuttons are relatively inexpensive, costing only a few dollars each. Additional funds are allocated toward power supplies, wiring, and prototyping accessories. By keeping the upfront hardware cost low, we retain a significant buffer of over $700 for replacements, PCB fabrication, or unforeseen design changes during development.

10.2 Bill of Materials

As shown in the table, the project's total cost is well within its budget. This leaves a surplus of $221.15 from our $1000 goal, providing a healthy margin for prototyping and replacement parts.

Table 10.2.1 Bill of Materials

Component	*Part Name*	*Quantity*	*Unit Price*	*Total Price*
Computer Engineer				
Raspberry Pi Compute Module 4	CM4 (8GB RAM, 32GB eMMC, WiFi)	1	$150	$156.38
CM4 IO Board	Waveshare IO Board Lite	1	$35.00	$35.00
Microcontroller	ESP32-S3-WROOM-1	1	$18.99	$18.99
Flash Memory	Winbond 25Q128JVSIQ (16MB SPI Flash)	1	$2.00	$2.00
Display	4.0” TFT LCD w/ Capacitive Touch (SPI, ribbon cable)	1	$45.00	$45.00
USB to UART	CP2102N-A02-GQFN24R	3	$4.14	$12.42
Electrical Engineer				
Custom PCB - Power	OSH Park fabrication (2-layer)	1	$42.40	$42.40
Custom PCB - Main PwnBoxer	OSH Park fabrication (4-layer)	1	$156.90	$156.90
Custom PCB - Interface	OSH Park fabrication (2-layer)	1	$86.20	$86.20
Custom PCB - ESP board	OSH Park fabrication (2-layer)	1	$26.15	$26.15
USB PD	IP2721-MAX12	1	$0.60	$0.60
Pushbutton Switch	SPST tactile button	2	$0.25	$0.50
Power Supply	USB-C Adapter	1	$10.00	$10.00
Stacked USB	72309-7043BLF	1	$2.68	$2.68
Ethernet	1-1734264-1	1	$1.03	$1.03
Miscellaneous	USB hub, wiring, headers, mounting hardware, etc.	–	$136.02	$136.02
Total Cost				
Computer Engineer		$267.79		
Electrical Engineer		$462.48		
Grand Total		**$732.27**		

10.3 Senior Design Milestones

Table 10.3.1 Senior Design Milestones

Milestone Number	*Milestone Description*	*Start Date*	*Anticipated End Date*	*Dependencies and Requirements*
1	Project Idea Finalization	08/18/25	09/09/25	Advisory Approval (Sahawneh)
2	Finalizing Design Choices	08/18/25	09/12/25	Confirming Design Requirements and Researching EE and CpE Compatibility
3	Divide and Conquer Document Finalization	08/28/25	09/06/25	Obtaining Professor Sahawneh's Meeting Availability
4	Divide and Conquer Meeting	09/09/25 12:30-1:00	09/09/25	Complete Divide and Conquer Documentation / Review Committee Form
5	Committee Formation / Meeting	08/28/25	09/05/25	Email Professors for Committee Availability
6	Midterm Report Draft	09/30/25	10/14/25	Communicate Deadlines with Team and Split Micro-Objectives
7	Midterm Report Draft Revision	10/14/25	10/31/25	Reference Document Guidelines and Revise
8	Final Report	10/20/25	11/06/25	Review and Approval of Team Parameters
9	Mini Demo Video	10/27/25	11/06/25	Reference Hardware Milestones for Part Integration

Table 10.3.2 Senior Design Hardware Milestones

Hardware Milestone	*Start Date*	*Expected Acquisition Date*	*Dependencies and Requirements*
ESP32-S3 Development Board Bring-Up	08/18/25	08/25/25	Development board delivery, programming environment setup
Connect ESP32-S3 to TFT Display	08/25/25	09/05/25	TFT display delivery, SPI wiring, verify graphics output
Interface External SPI Flash with ESP32	09/05/25	09/12/25	ESP32 SPI bus functional, connect 16MB Winbond flash
Breadboard Initial Prototype	09/12/25	09/20/25	ESP32 + display + flash tested, wiring diagram completed
Establish Communication with RPi CM4	09/20/25	09/30/25	Raspberry Pi CM4 environment prepared, wired connection to ESP32
Finalize PCB Design	10/01/25	10/10/25	Breadboard prototype validated, schematic/layout reviewed
Order PCB and Supporting Components	10/11/25	10/15/25	PCB design finalized, BOM approved
Hardware/Software Integration Testing	TBD SD2	TBD SD2	N/A
Final Prototype Assembly	TBD SD2	TBD SD2	N/A

11 Appendices

11.1 Appendix A - Reference

References

[1] G. Elbaz, "Code execution vulnerability: Impact, causes, and 8 defensive measures," https://www.oligo.security/academy/code-execution-vulnerability-impact-causes-and-8-defensive-measures, 2025, accessed: 2025-09-03.

[2] IBM X-Force, "Ibm x-force 2025 threat intelligence index," https://www.ibm.com/thought-leadership/institute-business-value/en-us/report/2025-threat-intelligence-index, Apr. 2025, accessed: 2025-09-03.

[3] M. Vanhoef and F. Piessens, "Key reinstallation attacks: Forcing nonce reuse in WPA2," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Dallas, TX, USA: ACM, 2017, pp. 1313–1328.

[4] BonelliSystems, "Automated penetration testing," Bonelli Systems Website, 2025, https://bonellisystems.com/automated-penetration-testing/.

[5] O. Team, "Kali Linux on a Raspberry Pi (A/B+/2) with Disk Encryption — offsec.com," https://www.offsec.com/blog/raspberry-pi-luks-disk-encryption/, [Accessed 12-09-2025].

[6] Hak5, "Lan turtle: Covert systems administration and penetration testing tool," Hak5 Website, 2025, https://shop.hak5.org/products/lan-turtle.

[7] N. Adamou, "Minipwner github repository," GitHub Repository, 2020, https://github.com/nicholasadamou/minipwner.

[8] SecPoint, "Penetrator: Vulnerability scanning appliance," SecPoint Website, 2025, https://www.secpoint.com/penetrator.html.

[9] W. McGrew, "Pwn the pwn plug: Analyzing and counter-attacking attacker-implanted devices," DEF CON 21 Presentation, 2013, https://www.defcon.org/images/defcon-21/dc-21-presentations/McGrew/DEFCON-21-McGrew-Pwn-The-Pwn-Plug.pdf.

[10] Hak5, "WiFi Pineapple — shop.hak5.org," https://shop.hak5.org/products/wifi-pineapple, [Accessed 12-09-2025].

[11] "Esp32-wroom-32/32e/32u datasheet," Espressif Systems, Tech. Rep., 2023, accessed: 2025-10-30. [Online]. Available: https://documentation.espressif.com/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf

[12] "Esp32-s2 datasheet," Espressif Systems, Tech. Rep., 2023, accessed: 2025-10-30. [Online]. Available: https://documentation.espressif.com/esp32-s2_datasheet_en.pdf

[13] “Esp32-s3 datasheet,” Espressif Systems, Tech. Rep., 2023, accessed: 2025-10-30. [Online]. Available: https://documentation.espressif.com/esp32-s3_datasheet_en.pdf

[14] “Esp32-c3 datasheet,” Espressif Systems, Tech. Rep., 2023, accessed: 2025-10-30. [Online]. Available: https://documentation.espressif.com/esp32-c3_datasheet_en.pdf

[15] “Esp32-c6 datasheet,” Espressif Systems, Tech. Rep., 2023, accessed: 2025-10-30. [Online]. Available: https://documentation.espressif.com/esp32-c6_datasheet_en.pdf

[16] “Esp32-h2 datasheet,” Espressif Systems, Tech. Rep., 2023, accessed: 2025-10-30. [Online]. Available: https://documentation.espressif.com/esp32-h2_datasheet_en.pdf

[17] “Msp430fr6989 datasheet,” Texas Instruments, Tech. Rep., 2023, accessed: 2025-10-30. [Online]. Available: https://www.ti.com/lit/ds/symlink/msp430fr6989.pdf

[18] “Stm32f103c8 datasheet,” STMicroelectronics, Tech. Rep., 2023, accessed: 2025-10-30. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

[19] “Arduino uno datasheet,” Arduino, Tech. Rep., 2023, accessed: 2025-10-30. [Online]. Available: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf

[20] “Arduino nano datasheet,” Arduino, Tech. Rep., 2023, accessed: 2025-10-30. [Online]. Available: https://docs.arduino.cc/resources/datasheets/A000005-datasheet.pdf

[21] BeagleBoard, “BeagleBone AI-64,” https://docs.beagleboard.org/beaglebone-ai-64.pdf?utm_source=chatgpt.com, 2025, [Accessed 26-10-2025].

[22] R. P. Ltd, “Raspberry Pi Compute Module 4,” http://datasheets.raspberrypi.com/cm4/cm4-datasheet.pdf?utm_source=chatgpt.com, 2024, [Accessed 26-10-2025].

[23] ——, “Raspberry Pi Compute Module 5,” https://pip-assets.raspberrypi.com/categories/944-raspberry-pi-compute-module-5/documents/RP-008180-DS-6-cm5-datasheet.pdf?disposition=inline, 2025, [Accessed 26-10-2025].

[24] N. Corp., “NVIDIA Jetson Nano System-on-Module,” https://openzeka.com/en/wp-content/uploads/2022/08/JetsonNano_DataSheet_DS09366001v1.1.pdf, 2019, [Accessed 26-10-2025].

[25] “Cfaf320480c7-035tc—tft lcd datasheet,” Crystalfontz America, Inc., Tech. Rep., 2024, accessed: 2025-10-30. [Online]. Available: https://www.crystalfontz.com/product/cfaf320480c7-035tc

[26] “Nhd-1.8(128×160)tft lcd—specification sheet,” Newhaven Display International, Tech. Rep., 2024, accessed: 2025-10-30. [Online]. Available: https://newhavendisplay.com/content/specs/NHD-1.8-128160EF-SSXN-FT.pdf

[27] “Ch7— datasheet,” Adafruit Industries, Tech. Rep., 2024, accessed: 2025-10-30. [Online]. Available: https://cdn-shop.adafruit.com/product-files/2354/Datasheet+.pdf

[28] USB Implementers Forum, "Universal serial bus specification revision 2.0," https://ieeemilestones.ethw.org/w/images/3/32/Usb_20.pdf, Apr. 2000, iEEE Milestones Wiki, ETHW.

[29] ——, "Usb type-c® cable and connector specification revision 2.0, august 2019," https://www.usb.org/sites/default/files/USB%20Type-C%20Spec%20R2.0%20-%20August%202019.pdf, Aug. 2019.

[30] Association Connecting Electronics Industries (IPC), "Ipc-2221a: Generic standard on printed board design," IPC, Northbrook, Illinois, Tech. Rep., May 2003, supersedes IPC-2221 (Feb 1998). [Online]. Available: https://www-eng.lbl.gov/~shuman/NEXT/CURRENT_DESIGN/TP/MATERIALS/IPC-2221A(L).pdf

[31] ——, "Ipc-7351: Generic requirements for surface mount design and land pattern standard," IPC, Bannockburn, Illinois, Tech. Rep., February 2005. [Online]. Available: https://www.electronics.org/TOC/IPC-7351.pdf

[32] ——, "Ipc-a-600j: Acceptability of printed boards," IPC, Northbrook, Illinois, Tech. Rep., May 2016, revision J. [Online]. Available: http://electronics.org/TOC/IPC-A-600J.pdf

11.2 Appendix B - Copyright Permissions

11.3 Appendix C - Data Sheets

11.4 Appendix D - Software Code

11.5 Appendix E - ChatGPT Prompts and Outcomes

This appendix documents all AI-assisted activities related to the PwnBoxer project, in accordance with UCF's academic integrity and transparency guidelines. All content generated with AI tools was independently verified, edited, and validated by team members before inclusion in this report.

Table E.1 AI Tool Usage Summary

Tool / Platform	Primary Function	Usage Description	Verified By
ChatGPT (OpenAI)	Technical writing, code review, LaTeX generation	Assisted in drafting hardware/software comparison tables, refining section summaries, and reformatting CSV logs into publication-quality LaTeX tables using the `\resizebox` environment.	Abdiel
ChatGPT (OpenAI)	Firmware optimization suggestions	Analyzed ESP32 SPI frame buffer code; proposed non-blocking DMA transfer scheme and LVGL redraw logic for smoother display performance.	Franco
Grammarly / LanguageTool	Grammar and style polishing	Used to proofread final report text for clarity, conciseness, and tone consistency prior to submission.	Bobby
Overleaf LaTeX AI Assistant	Syntax validation and table formatting	Verified compatibility of complex tables and ensured consistent font, margins, and figure alignment across chapters.	Abdiel
GitHub Copilot	Autocompletion for embedded C / Python code	Provided boilerplate generation for ESP-IDF tasks and basic REST endpoint handlers. All logic reviewed manually before integration.	Franco

All AI contributions were limited to productivity support. Core engineering decisions, mathematical derivations, and experimental evaluations were performed by human team members. No confidential or proprietary data were exposed to AI systems.

11.6 Appendix F - etc.