# Engineering Portfolio

Alex J. Zimmermann

- Website: ajzimmermann.github.io
- E-mail: ajzimmermannapple@gmail.com
- LinkedIn: www.linkedin.com/in/ajzimmermann

## About this portfolio

This portfolio complements my resume by showcasing a few hands-on engineering projects. Mostly **personal projects** I pursued for the joy of making, plus one from my coursework. These projects are meant to show off my experience in certain topics. They cover a range of topics, from designing my own circuit boards, building products using off the shelf parts, server/backend understanding and much more.

I take ideas **end-to-end**: define requirements, **reverse-engineer dimensions**, design for **manufacturability and service**, validate with **real measurements**, and iterate quickly.

I keep this portfolio updated as I refine designs and add new work.

Questions or feedback are welcome—feel free to reach me by email.

# Table of Contents

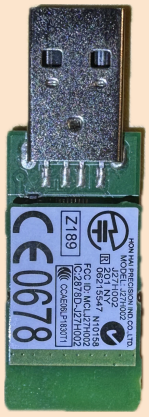

**Engineering Portfolio** **1**
About this portfolio 1
**Table of Contents** **2**
**Custom Wii Bluetooth Module Computer Interface** **3**
Goal 3
Overview 3
Role & Timeline 3
Real-world issues I set out to solve 3
Design choices 4
Version history 5
Validation & results 6
Conclusion and Reflection 7
**100W LED Flashlight** **8**
Overview 8
Role & Timeline 8
Design 8
Conclusion & Reflection 12
**Linux Home Server** **13**
Overview 13
Hardware (current) 13
Role & Timeline 13
Services — What, Why, and How 14
1) NAS (TrueNAS CE in Docker) 14
2) Network-wide Ad Blocking (AdGuard Home) 14
3) Nextcloud Office 14
4) VPN / Remote Access (Tailscale) 15
5) Home Assistant 15
6) Pterodactyl Panel (Game Server Hosting) 15
7) “Lab” — Containers & VMs for Try-outs 16
Design Choices 16
Validation & Results 16
Conclusion & Reflection 17

# Custom Wii Bluetooth Module Computer Interface

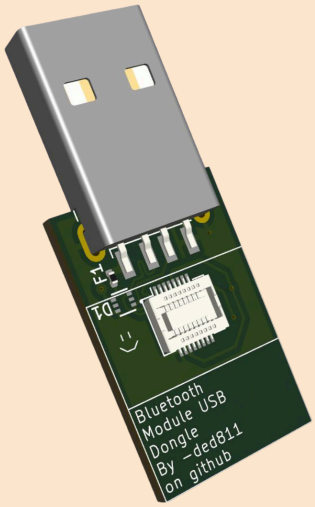
## Goal

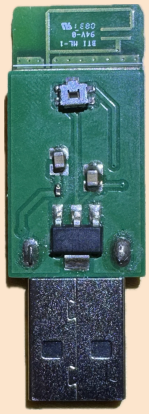
Create a custom PCB that exposes an authentic **Wii Bluetooth module** to a PC **over USB** for emulator passthrough—eliminating the pairing/audio quirks seen with off-the-shelf bluetooth modules and making Wii remotes behave like they do on real hardware.

## Overview



- PCB that hosts the authentic Wii BT module and presents it to Windows/Linux as a pass-through device for emulators
- Designed around **clean power**, **RF-aware layout**, and **simple setup** (sync button, shield ground, proper terminations)
- Addresses the practical issues I saw with normal Bluetooth adapters: unreliable pairing, **speaker audio only on some remotes**, and “works on one machine but not another”
- This project is still actively being worked on by me in my spare time and is constantly improving.



## Role & Timeline

**Role:** Solo design, research & build
**Scope:** research ➔ Kicad PCB➔ assembly ➔ testing
**When:**
V1 (Sept 2024): Concept, PCB design in KiCad, ordering, assembly, and first working prototype
V2 (Oct–Nov 2024): Board redesign, fabrication, assembly, and verification testing
V3 (Dec 2024 ➔ Sept 2025, WIP): Layout/EMI refinements, ergonomics updates, and simulation-backed reviews

## Real-world issues I set out to solve

- **Adapter compatibility:** Built-in and off-the-shelf PC Bluetooth adapters often struggle with **Wii Remote variants** and third-party remotes—pairing can be inconsistent or fail outright
- **Speaker audio quirks:** On non-TR (RVL-CNT-01) remotes, speaker audio is typically garbled/broken; TR (RVL-CNT-01-TR) can be somewhat better, but still not like real hardware.
- **Link stability:** Connections can be **drop-prone** or weak at normal distances, especially when compared to real hardware.

> *Footnote (my testing): On my desktop, only genuine Nintendo remotes would connect to the computers included bluetooth module; on my laptop, a third-party remote paired but **speaker audio didn't work**, and the genuine Wiimotes refused to pair. The pass-through board resolved these across both machines.*

## Design choices

**Power & regulation** **Version**

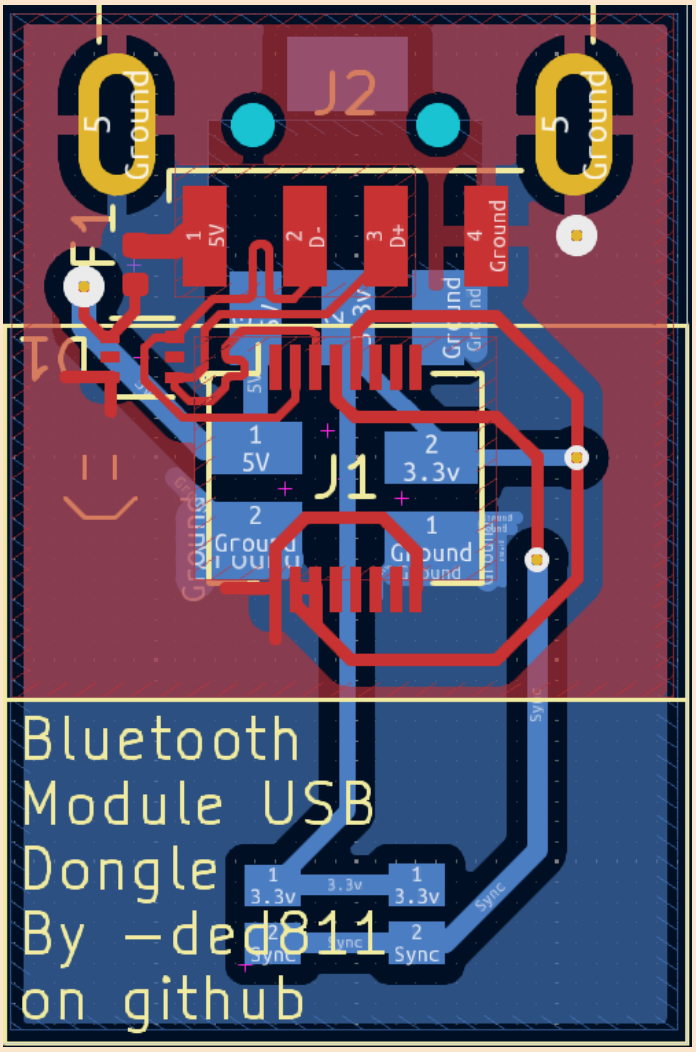
- On-board **step-down/buck** set to the Wii module's required voltage of 3.3v (V1+)
- Tight **decoupling capacitors** around the module to stabilize the power rails and protect the circuit (V1+)
- A **TVS diode** purpose built for usb devices to clamp **ESD** and **plug-in surge** events on **VBUS** and **D±**, reducing cross-device interference (V2+)
- **eFuse 500 mA hold** over-current protection; module listed at **~300 mA max**, and my measured draw topped out at **~180 mA → ample headroom** without nuisance trips (V2+)

**Signal integrity & RF hygiene**

- **Length-matched differential data pairs** and impedance control on the USB side (V2+)
- **Ground plane** reworked to contain as much emf as possible while staying away from the antenna (V2+)
- **Power rails:** Relocated **well beyond the antenna keep-out**, prioritizing **maximum physical separation** and routing behind the module's shield whenever possible (V2+)
- **Component placement:** Pushed **noisy/fast components as far from the antenna aperture as the layout allowed**—preferencing locations **behind/below the shield can** to minimize coupling (V2+)
- Added **explicit shield ground pad** for the module's can/EMI foam to give a solid RF reference (V3)

**Assembly and placement**

- **Two-sided SMT:** The Bluetooth module must sit on the **top side**—its height plus the connector stack-up would otherwise extend **below a modern laptop's chassis**. To avoid **blocking adjacent USB ports**, the **sync button** is placed on the **bottom**, which necessitates a **two-sided PCB**.
- **Antenna orientation considerations**: The antenna on the Bluetooth module's PCB is semi-directional and would normally face the user. The way that the Wii minimizes the semi-directional nature is by having it protrude from the Wii's motherboard, facing the user, and by having no components around it. When plugged into a laptop, it will be facing sideways to the user. Thankfully its only being slightly semi-directional means that it will still work like this, but we need to have it protrude from the PCB and take special care in reducing interference and moving components as far away from it as I can.




**Software**

- **Windows:** On Windows, you need a driver to pass the USB device through to your emulator. I opted to use the open source USB driver Libusbk.
- **Linux:** On Linux no driver is needed, but if you are using Dolphin Emulator installed from a flatpak or snap package you will need to grant the app permission to access USB devices, due to the sandboxed nature of these packages.

## Version history

**V1 — Proof of concept (Sept 2024)**

- Connector, **buck regulator**, sync button, decoupling capacitors
- Focus on **software path** and pass-through bring-up; confirmed behavior on Windows and Linux
- Worked, but **less than stable** and orientation-sensitive performance

**V2 — Robustness & protections (Oct–Nov 2024)**

- Added **TVS** + **eFuse**, **length-matched pairs**, and cleaned return paths
- Pushed switching nodes and power rails **away from the antenna**, kept them **behind the shield**
- **Result: significantly fewer dropouts**, strong connections at typical distances

**V3 — EMI & Optimizations(WIP; resumed Sept 2025)**

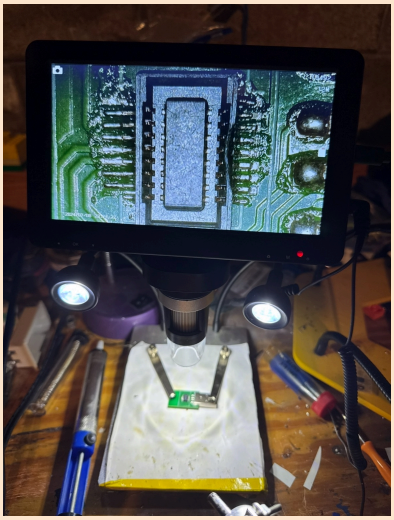
- **Impedance matching,** based on what modders have reverse engineered
- **Sync button repositioned**, pushed slightly back for better signal integrity
- **Exposed shield-ground pad** for proper EMI foam contact
- Components **pushed back** to be kept clear of the antenna window; basing component placement distance on theoretical interferences
- Started using **ngspice in KiCad** (after learning OrCAD X/PSpice in class) to simulate and validate supply filtering and transient response

## Validation & results

- **Pairing & audio:** Now pairs **reliably** with both **genuine** and **third-party** remotes; **Wiimote speaker audio works** where it previously failed
- **Stability:** After the V2 layout changes, **dropouts disappeared** at normal desk/living-room ranges. I verified this with **black-box tests** (range/orientation sweeps, A/B board revisions) and **Wireshark** captures of **Bluetooth HCI/USB** traffic to estimate **drop/retry rates**—rates stayed near zero under typical use.
- **Platform notes:**
    - **Windows:** switch the device to **LibusbK** (using Zadig) for Dolphin pass-through
    - **Linux:** works without extra drivers; ensure user USB access

## Conclusion and Reflection

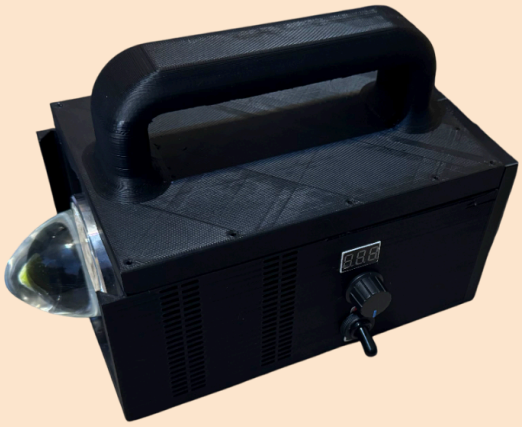
This project gave me hands-on, real-world skills I can use immediately. I moved from curiosity to a reliable, repeatable solution—building real skills along the way: **KiCad** (schematic ➔ layout), reading and applying **datasheets,** building clean **BOMs, ordering custom PCBs,** and a big leap in **microsoldering** (two-sided hot air, paste control). I learned to think like the RF: **keep returns clean, give the antenna space, move noisy rails, add protection at the connector**, and verify with testing—Windows driver swaps, Linux permissions, real remotes in real rooms. Without module-level debug hooks, I validated changes using **black-box testing** and **Wireshark** captures of **Bluetooth HCI/USB** traffic to estimate **drop/retry rates**, then correlated those with orientation, distance, and board revisions. Each iteration (V1➔V3 WIP) simplified the design and improved pairing, audio, and stability.



But most importantly, this is more than an adapter to play games. It's a **passion project**—something I build because I love making things work the way they *should*. Every revision (V1➔V3) has been an excuse to learn, simplify, and improve. I'm proud of how far it's come, and I'm excited to keep pushing it—not because I have to, but because the process itself is the reward.

# 100W LED Flashlight

## Overview



- Final project for **ENGR 240** (sophomore-level), **Illinois Central College (ICC)**
- **100 W LED handheld light** with **adjustable brightness**
- **Near-silent** cooling at full power
- **High-frequency PWM** (camera-safe, no visible flicker)
- **Hot-swappable Milwaukee M18** battery power
- **Focused yet practical beam**—clear reach with gentle spill.

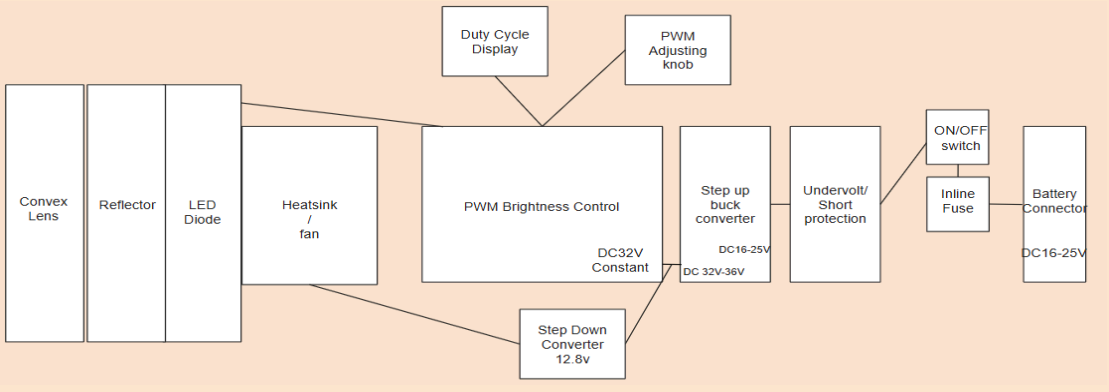
## Role & Timeline

**Role:** Lead design, research & build
**Scope:** concept → CAD → electronics → testing
**When:** January-May 2025

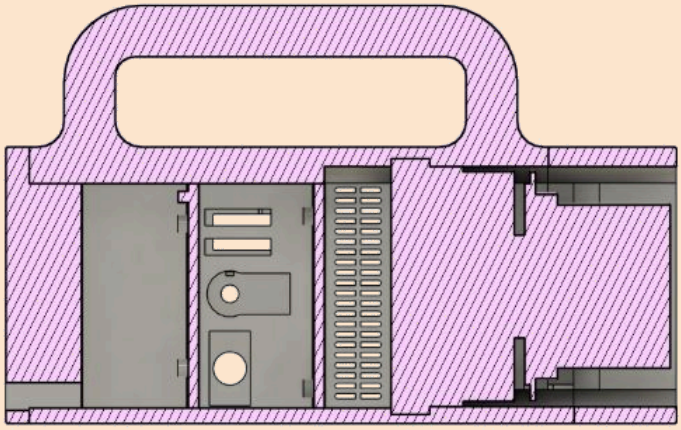
## Design

### 1) Requirements & Research

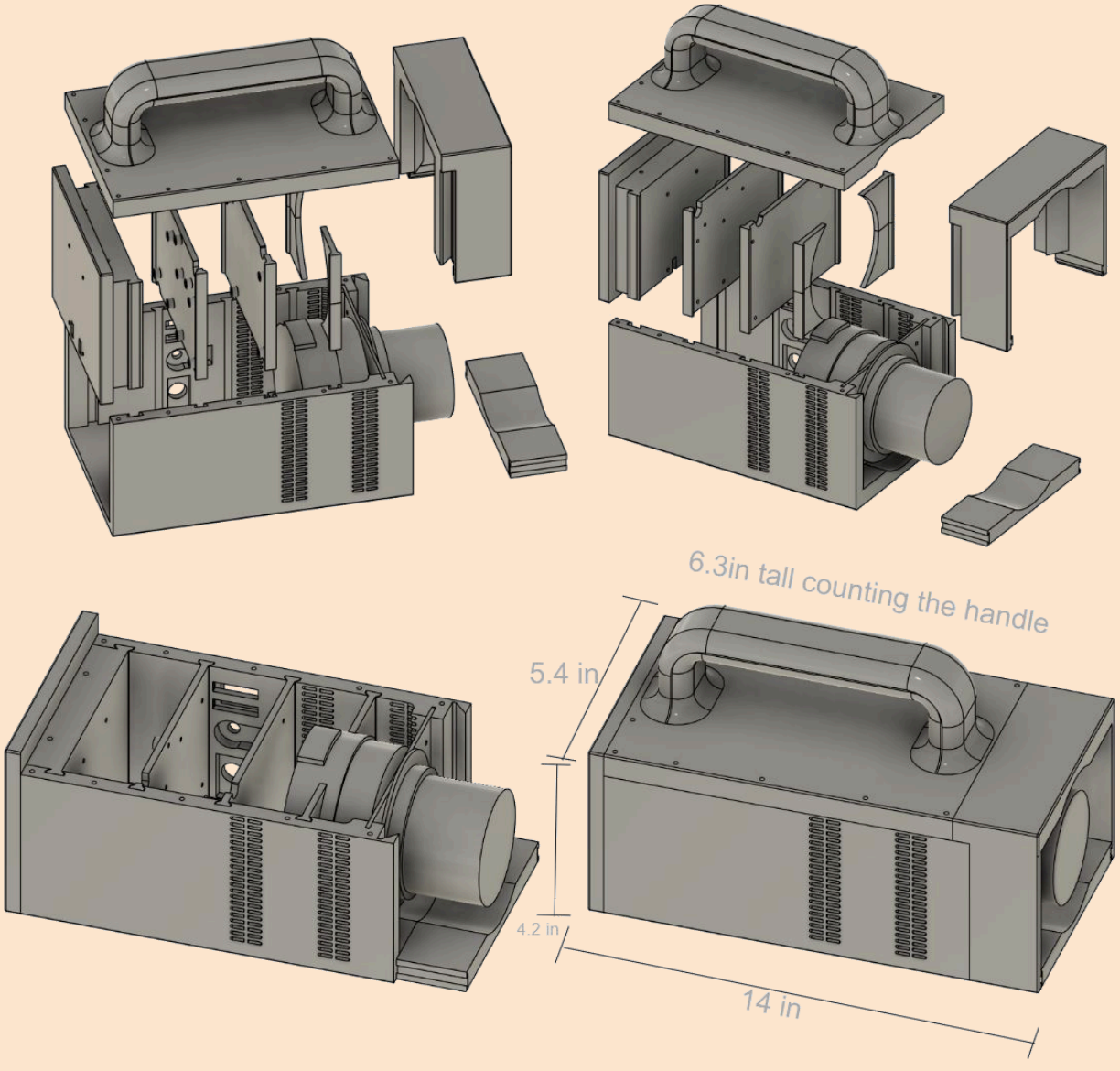
- Started by selecting our **100 W LED** and basing our components around that.
- **Optics:** selected a lens that delivered a controlled, long-reach beam with useful spillage to prevent harsh hot spotting
- **Cooling:** targeted quiet operation in the tight vents a compact design would require; selected **AMD Wraith Stealth** for footprint, fin density, and low noise
- **Power (M18): recreated the tool-side battery protection** for M18 packs so the light can draw power safely



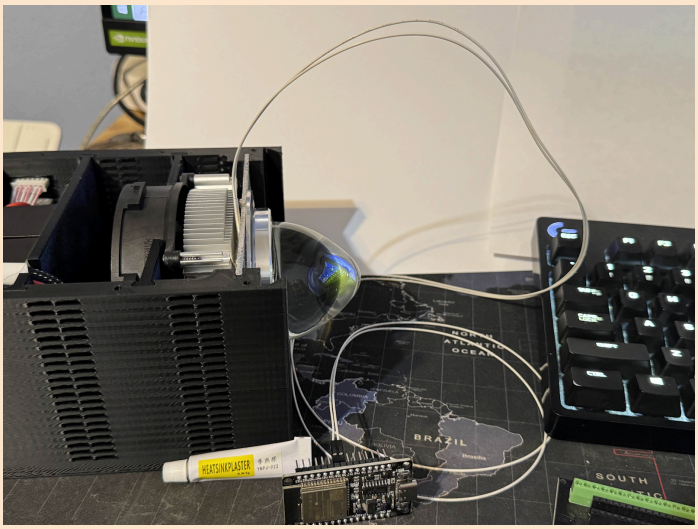
**2) CAD Modeling**



- The CAD phase was the most time-consuming, focusing on a compact, tight-fitting enclosure that preserved airflow and service access
- Modular, slot-in sections with pre-mounted electronics—fast to assemble/disassemble and easy to service
- **Designed in Autodesk Fusion 360:** parametric model with modular assemblies and tolerance-driven fits
- **Tolerance-first guides: Dual-dovetail rail slots** leverage the plastic's natural flex to reduce tight-tolerance requirements

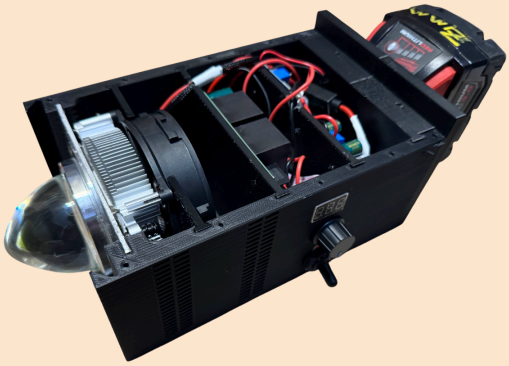


**3) Thermal Tuning**

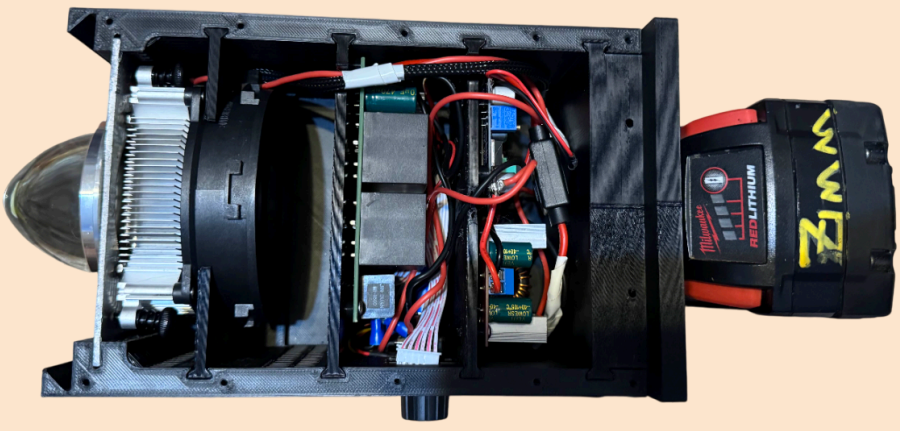



- **Optimized approach:** Initial plan used MCU fan curves; testing showed the fan is effectively inaudible except in a truly silent room, so **active control offered no practical benefit**
- **Method:** Set fan speed with an **adjustable buck converter** while logging LED temperature (probe seated in the LED's **integrated probe slot**); an **ESP32** streamed that data to a laptop during tuning
- **Result:** At **12.8 V** (≈**25 °C** ambient), the LED stabilized near **85 °C** at continuous full power—**ample headroom** with **near-silent** acoustics
- **Decision:** Fix the buck at **~12.8 V** in the final build—**no MCU in the cooling path**. This **reduced complexity and cost** with **no practical downside** for noise or thermal performance

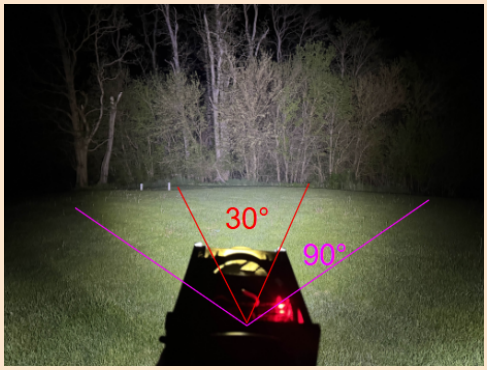
**4) Electrical and control**



- **LED brightness: high-frequency PWM** — the LED board/driver was chosen for its **very high PWM carrier**, so even **professional cinema/photography cameras** don't show banding or flicker in normal use.
- **LED powering buck/boost converter:** Chosen to boost and stabilize the batteries' voltage to ensure peak brightness no matter the battery level
- **Fan power: buck converter** set to lower and stabilize the batteries voltage to the validated voltage from thermal tuning
- **Harnessing: quick-disconnect** properties of the **M18 interface** to gain a hot swappable robust battery solution.



**5) Optics and lighting**




- **Optics:** The selected lens **concentrates most output into a 30° cone**, then **softly tapers** into a **90° spill**, giving long-reach visibility without a harsh hotspot or tunnel vision
- **Photography suitable:** No visible banding/flicker on **iPhone 15 slow-motion (1080p/240 fps)** and **Canon EOS R50 slow-motion (FHD/120 fps)** test clips
- **Brightness: Meets the design target and feels excellent in use**—subjectively very bright

## Conclusion & Reflection

I'm genuinely thrilled with how this light turned out. It's compact, modular, and easy to service; the slot-in sections and screw terminals make assembly and maintenance painless. The beam is exactly what I wanted—most of the output in a clean 30° core with a soft 90° spill—and it's **absurdly bright** in real use. My photos don't do it justice; the perceived brightness and throw are far better to the eye than a camera can capture.

A big win is power: I'm not buying into another swappable-battery ecosystem. I'm using the Milwaukee M18 packs I already own, which keeps costs down and uptime high. The cooling solution hits the sweet spot too—a fixed fan set point that keeps temps in check while staying essentially inaudible—and the high-frequency PWM plays nicely with cameras.

Along the way, this project built real, **transferable skills.** It was my **first big build** where I **reverse-engineered a lot of dimensions** from off-the-shelf parts and turned them into **tolerance-driven, parametric fits** in Fusion 360. I designed a **modular enclosure** that assembles/disassembles quickly and applied **DFMA** thinking—fastener access, service clearances, and **dual-dovetail guides** that use plastic compliance. I instrumented the light with a **temperature probe** and **ESP32 logging** and, using the data that I collected in order to calibrate a circuit: **set a single validated fan voltage**. I also validated camera usability (iPhone 15 @ 240 fps and Canon EOS R50 @ 120 fps) and recreated the tool-side M18 interface so hot-swap is reliable.

Overall, it meets (and honestly exceeds) my goals: simple to use, quiet, camera-friendly, and powerful—at a cost far lower than comparable lights. I'm proud of the result and excited to keep iterating, but as it stands, I'm very happy with it.

# Linux Home Server

## Overview

- **OS:** Ubuntu Server **24.04 LTS** (started on Debian for stability; moved to Ubuntu for newer features)
- **Approach:** Everything is **containerized** (Docker/compose), with per-service volumes, ports, and resource caps
- **Clients:** Fire TV Stick 4K devices at home (IPTV apps), plus a paired **phone app** so we can watch the same channels/recordings away from home via VPN

## Hardware (current)

- **CPU:** AMD **Ryzen 5 3600** *(used)*
- **RAM: 32 GB DDR4** *(used)*
- **GPU:** AMD **FirePro W5100** *(used)*
- **System SSD: 512 GB Fanxiang**
- **Motherboard:** MSI **X370 Gaming Pro Carbon** *(used)*
- **Storage:** 2× **Seagate BarraCuda 8 TB** *(refurbished server HDDs)* for DVR/NAS
- **Chassis: 4U rack mount** case (spacious, quiet fans, easy maintenance)


## Role & Timeline

**Role:** Solo build & operations
**Scope:** hardware → OS → services → backups/monitoring → ongoing maintenance
**Timeline:** Initial build **2019**, iterative upgrades to today's configuration

## Services — What, Why, and How

### 1) NAS (TrueNAS CE in Docker)

- **What:** A **TrueNAS Community Edition** instance (Docker) serves as the **central file server** for IPTV recordings, family files, and my **project archives**.
- **Why:** We cut cable, moved every TV to **Fire TV Stick 4K** with IPTV apps, and needed a **single place** to store recordings and keep them accessible on all TVs and phones.
- **How:**
    - **Storage layout: SSD** (512 GB) handles **OS boot** and **all containers/apps** for responsiveness. Two **8 TB HDDs** form a **mirrored pool** (RAID1-style) for **data**—recordings, archives, backups, and projects.
    - **Shares: SMB** exports for TVs/phones/PCs with separate shares: **Recordings**, **Archive**, **Backups**, **Projects**.
    - **Health & integrity:** Scheduled **SMART** tests and periodic **scrubs**; snapshots on key datasets (Projects, Backups).
    - **Retention:** Weekly cleanup on *Recordings*; keepers moved to *Archive*.
    - **Off-site safety (Projects):** Scheduled **one-way sync to Google Drive** as a **secondary copy**. The **NAS is source-of-truth**; I also keep local snapshots and periodic **offline exports** so I'm protected even if a Google account is compromised or a provider loses data.

### 2) Network-wide Ad Blocking (AdGuard Home)

- **What: AdGuard Home** as the home DNS, blocking ads/trackers for every device.
- **Why:** Clean browsing on all TVs/phones without per-device setup; simple allow-lists for edge cases.
- **How:**
    - Runs in Docker with persistent volumes (config + logs).
    - Upstream DNS with fallback, **per-device rules**, query logs for "top talkers."
    - **Tailscale** routes my phone/laptop DNS through AdGuard so **ad block follows me** off-site (huge quality-of-life win).

### 3) Nextcloud Office

- **What:** Private, lightweight cloud editing/notes for me.
- **Why:** Central place for docs without relying on third-party accounts and subscriptions.
- **How:** Dockerized with reverse proxy; data on the NAS volume; nightly backup of the data dir + DB dump.

### 4) VPN / Remote Access (Tailscale)

- **What: Tailscale** on the server + my devices.
- **Why:** Simple, secure remote access; lets the **AdGuard DNS** work everywhere I am.
- **How:** Device routes are advertised from the server; no port-forward juggling; works on LTE/guest Wi-Fi.

### 5) Home Assistant

- **What: Home Assistant** (Docker) to control the **smart outlets** that power my 3D printers.
- **Why:** Convenience and safety—centralized on/off control, prevent overnight idling, and cut power when prints are done so the printers aren't left energized.
- **How:**
    - Integrates the printer **smart plugs** into a simple dashboard (phone + web).
    - **Automations:** timed **auto-off** after long idle, one-tap **preheat power-on**, and a **manual kill-switch** tile.
    - **Extras:** basic **energy stats** per outlet to see print job consumption.

### 6) Pterodactyl Panel (Game Server Hosting)

- **What: Pterodactyl** panel in Docker to spin up/down game servers—Minecraft almost always; ARK Survival Evolved, Astroneer, etc. whatever me and my friends are playing at the time.
- **Why:** Click-to-launch servers with clean isolation and easy updates, without breaking the system. Secure **dual-layer sandboxing** (Docker host ➜ Pterodactyl game containers) protects the system, and the panel provides **clear dashboards/logs/metrics** for quick admin.
- **How:**
    - Compose templates with **CPU/RAM limits**, so storage services stay responsive.
    - World saves and configs live on **mounted volumes,** with periodic backups.

### 7) “Lab” — Containers & VMs for Try-outs

- **What:** A separate compose stack / VM pool for experiments.
- **Why:** I like to test services without risking the family’s TV night.
- **How:** Dedicated network/subnet, capped resources, documented teardown scripts.

*(Previously: self-hosted website and **Jellyfin** for DVD rips. The IPTV UI ended up so clean that I migrated that media into the IPTV workflow and retired Jellyfin.)*

## Design Choices

- **Separation of concerns:** Each service in its **own container** with explicit **volumes/ports** → easy updates/rollbacks.
- **Simple recovery:** docker compose + .env → one-command rebuilds on new hardware.
- **Storage hygiene:** Organized shares, predictable paths, **retention policy** for recordings, and tested **restore scripts**.
- **Network sanity:** AdGuard as primary DNS; Tailscale for off-site continuity; no ad-hoc port chaos.
- **Performance isolation:** Resource ceilings on game servers; NAS always gets the I/O it needs.
- **Observability:** SMART checks, container health checks, and simple logs/dashboards so I catch issues early.

## Validation & Results

- **DVR/NAS:** Recordings land reliably; **playback** is smooth on TVs and phones (local + over VPN).
- **Ad blocking:** House-wide reduction of ads/trackers; quick per-device bypass when troubleshooting.
- **Game nights:** Servers spin up in **minutes**; stable latency on LAN and solid for friends online.
- **Ops:** Clean reboots and updates; compose files make redeploys **predictable**.

## Conclusion & Reflection

This server started as a budget build in 2019 and has grown into a dependable platform I use every day. It balances reliability (SSD for OS/containers, mirrored HDD pool for data), privacy (AdGuard + Tailscale—even on public Wi-Fi), and flexibility (spinning up game servers or experiments without touching the NAS). Moving from Debian to Ubuntu 24.04 brought newer stacks, while Docker + Compose keeps services reproducible and easy to recover. It was also my **first build in a server chassis**, which taught me how rack gear differs from desktops—front-to-back airflow, non consumer power supply standards, cable management in tight spaces, drive sleds and mounting, and the importance of planning for serviceability in a 4U rack.

What I gained goes beyond a working setup. On the hardware side, I learned **server standards**—rack mounting, PSU form factors, fan/pinout conventions, and cable/airflow planning—that don't show up in typical desktop builds. I developed real **server home-lab operations** skills—owning the full stack from hardware through OS, monitoring, and routine maintenance—and a strong **storage and backup discipline**, with a mirrored pool, dataset snapshots, and off-site/offline copies that I actually test to restore. I leveled up **service management** by standardizing deployments with Compose, versioning configs, setting resource ceilings, and making rollbacks predictable. I also built a practical **security** mindset: key-only SSH with Fail2ban, firewall allow-lists and segmentation, non-root containers with minimal mounts/capability drops, and Tailscale ACLs for private, encrypted access. Underpinning all of it is **containerization**—per-service isolation, health checks, persistent volumes, and clean rebuilds on new hardware. It's a home server, but it doubled as my lab for Linux, networking, storage, security, and service orchestration—and it keeps paying off every time I spin up something new and it just works.