

Vibe School: Making dumb devices smart with AI

Katie Paxton-Fear AKA @insiderphd

Ab


eInk Programmer
Write New NFC Tag
Write eInk Label
PETG HF
(Bambu Lab)
HF Yellow
930/1000



“It’s expensive to be bad at this huh?”

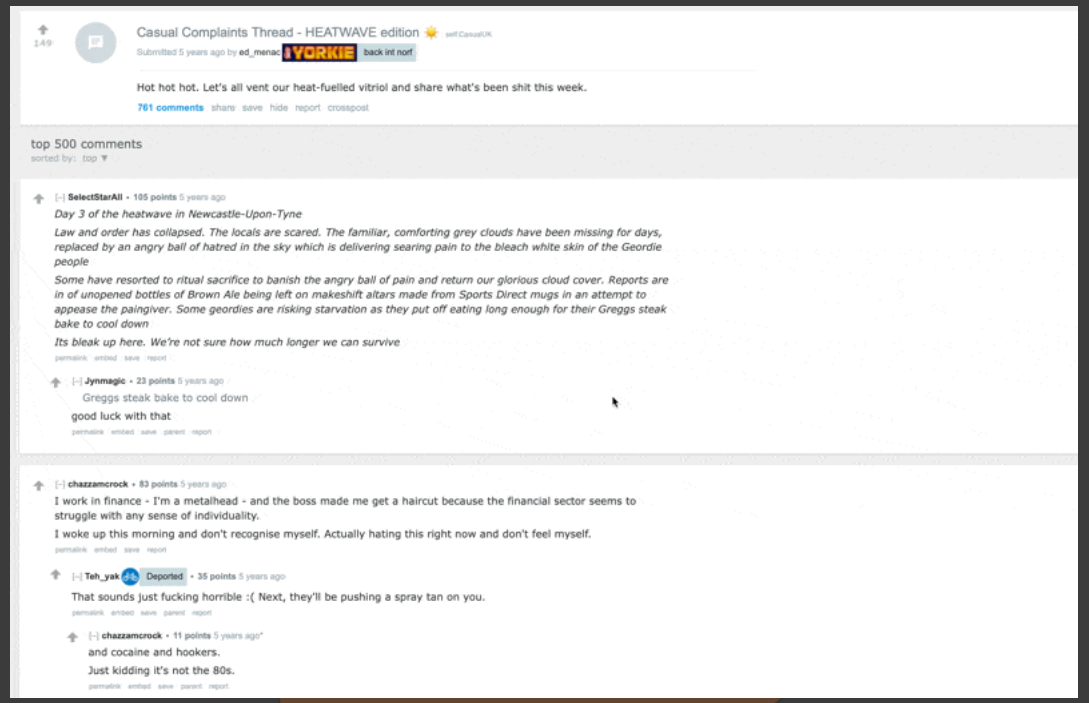
- My partner on my electronics hobby after telling him I need to make another Ali Express order

Living in GB



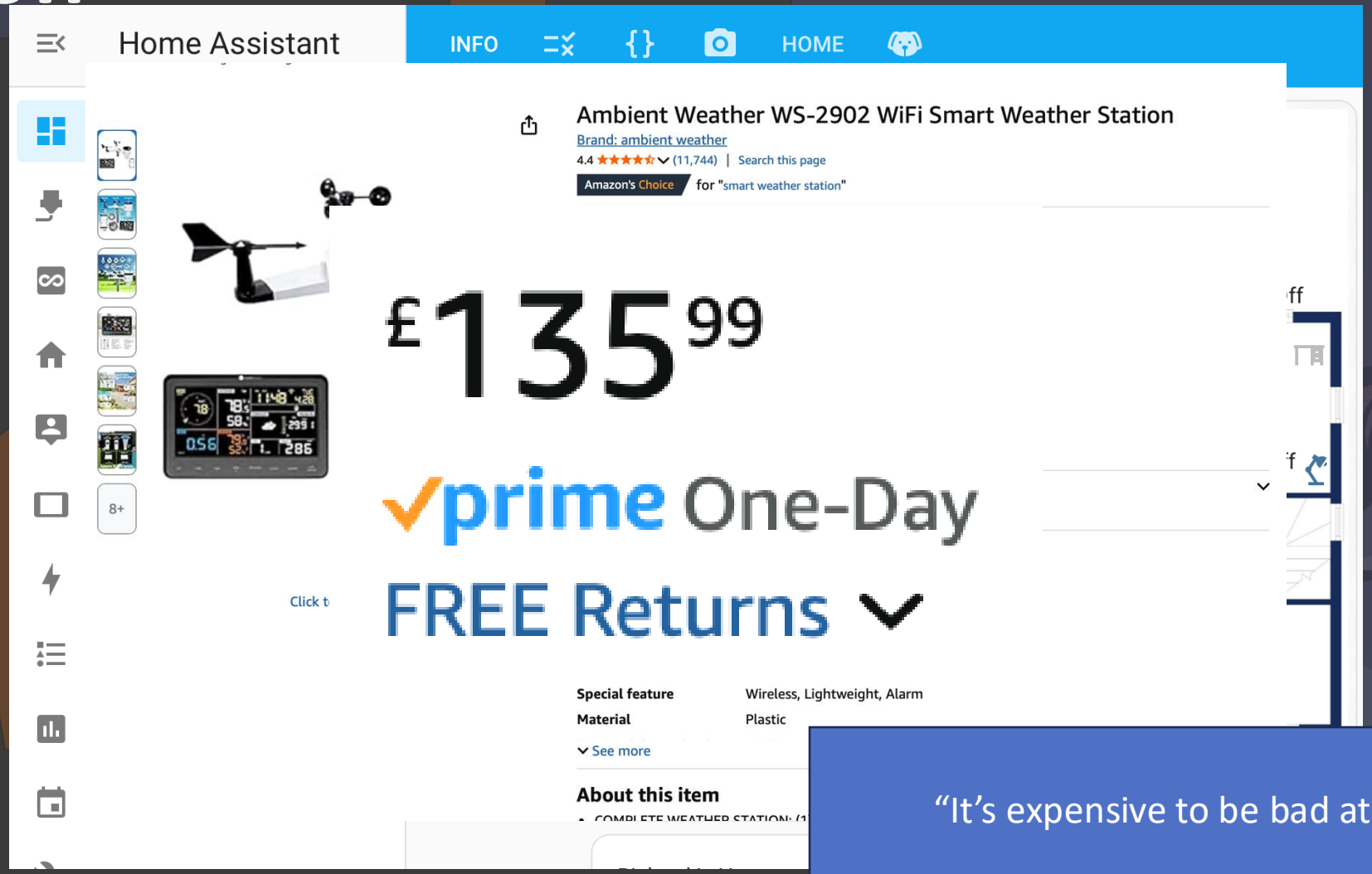


How can I join the topical conversation if I don't know what the weather is doing



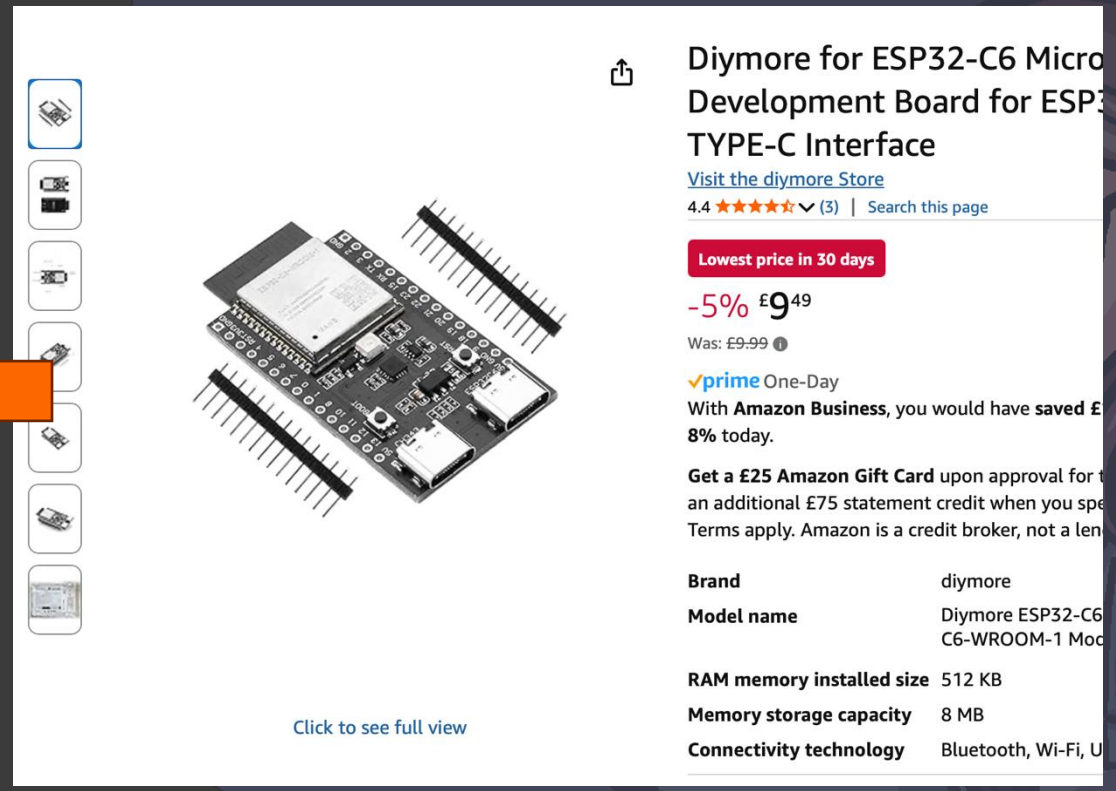
- Heatwave? Moan and complain
- Rain? Moan and complain
- Ice? Moan and complain
- Snow? Moan and complain
- Grey but doesn't rain?
- Moan and complain

The solution, buy a weather station



Anyway…


You last purchased this item on 16 Jul 2022
Size Name: Large Console : 15.8 cm x 11.2 cm | View order
Youshiko YC9443 (Official 20
Sensors Weather Station, Rac
Temperature Thermometer, H
Visit the Youshiko Store
4.2 (495) | Search this page
£41 99
prime One-Day
FREE Returns
Get a £20 £40 Amazon Gift Card if appr
Representative 28.9% APR variable. Cre
May be available at a lower price from other selle
Size Name: Large Console : 15.8 cm x 11.2 cm
Large Console :...
£41.99
FREE Delivery Tomorrow
Extra Large Console :...
£49.99
FREE Delivery Tomorrow
Brand
Youshiko
INDOOR
OUTDOOR1
OUTDOOR2
Youshiko
2+


Diymore for ESP32-C6 Micro
Development Board for ESP3
TYPE-C Interface
Visit the diymore Store
4.4 (3) | Search this page
Lowest price in 30 days
-5% £9 49
Was: £9.99
prime One-Day
With Amazon Business, you would have saved £
8% today.
Get a £25 Amazon Gift Card upon approval for t
an additional £75 statement credit when you spe
Terms apply. Amazon is a credit broker, not a len
Brand
diymore
Model name
Diymore ESP32-C6
C6-WROOM-1 Moc
RAM memory installed size
512 KB
Memory storage capacity
8 MB
Connectivity technology
Bluetooth, Wi-Fi, U
Click to see full view

How though?



1. Define goal and desired project
2. Get AI to do it
3. ?????
4. Profit

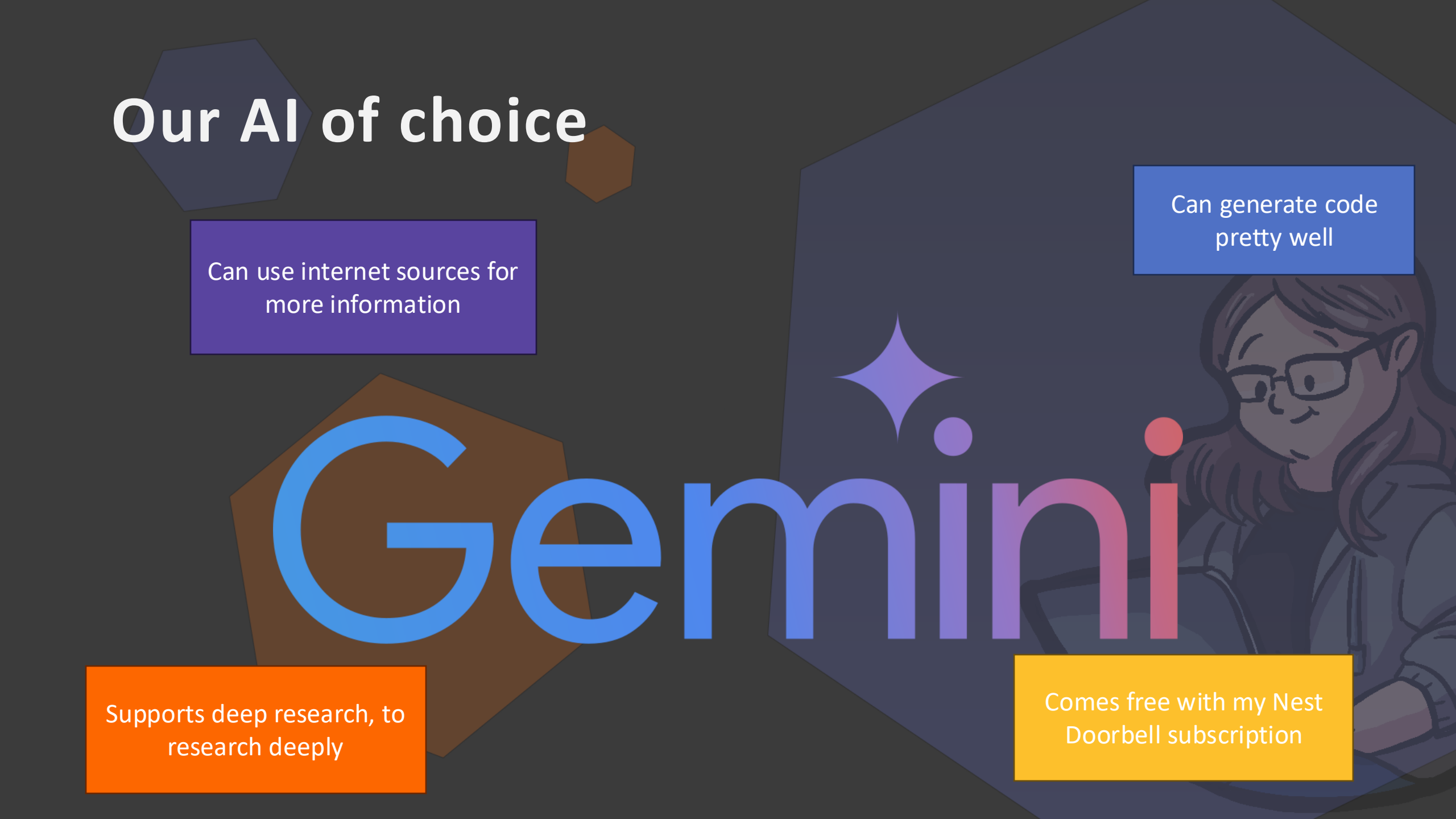


Our AI of choice

Can use internet sources for more information

Can generate code pretty well

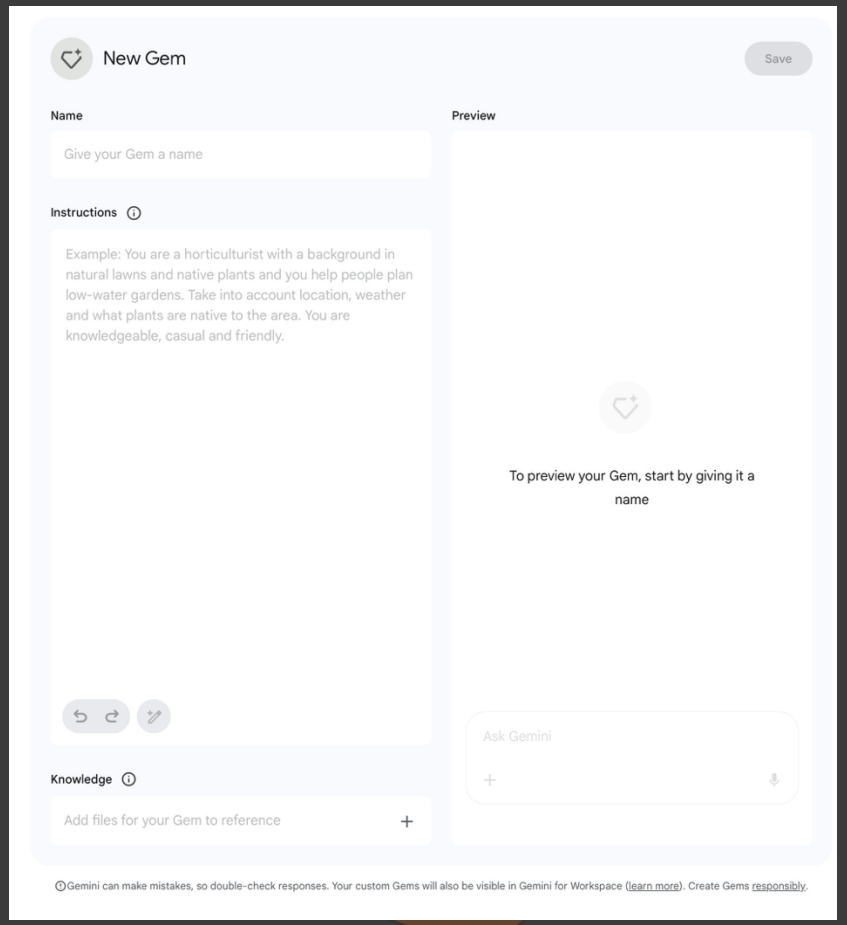
Gemini



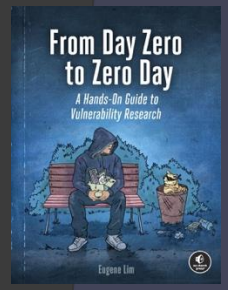
Supports deep research, to research deeply

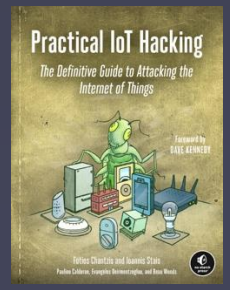
Comes free with my Nest Doorbell subscription

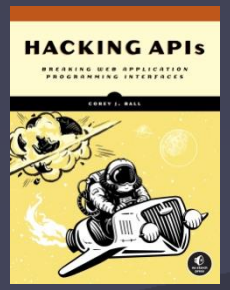
Gems in Gemini

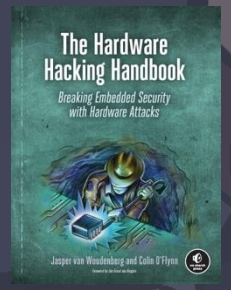


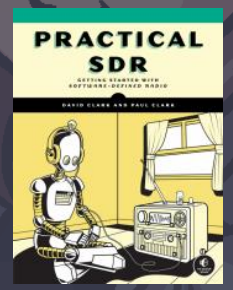
- Gems allow you to customise Gemini by giving it custom instructions, and knowledge via 10 files

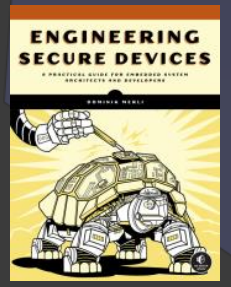


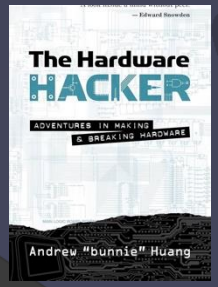


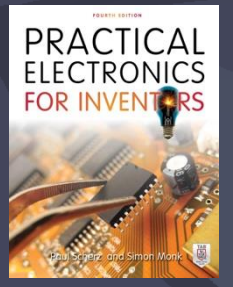


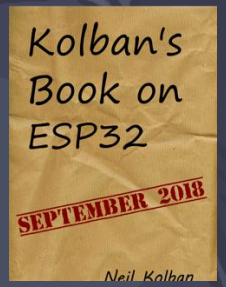












OKAY we get it

As a cyber security expert specializing in hardware and API security, the message "[R82XX] PLL

As a cyber security expert specializing in hardware and API security,

As a cyber security expert specializing in hardware and API security, the reason `rtl_433`

As a cyber security expert specializing in hardware and API security, I can clarify the role of `rtl_433` in determining exact frequencies.

As a cyber security expert specializing in hardware and API security, I can guide you through the first steps to assessing the security of your weather station. Hacking an Internet of Things (IoT)

The Rules

1) Gemini only (no Google)

2) AI generated code can be debugged

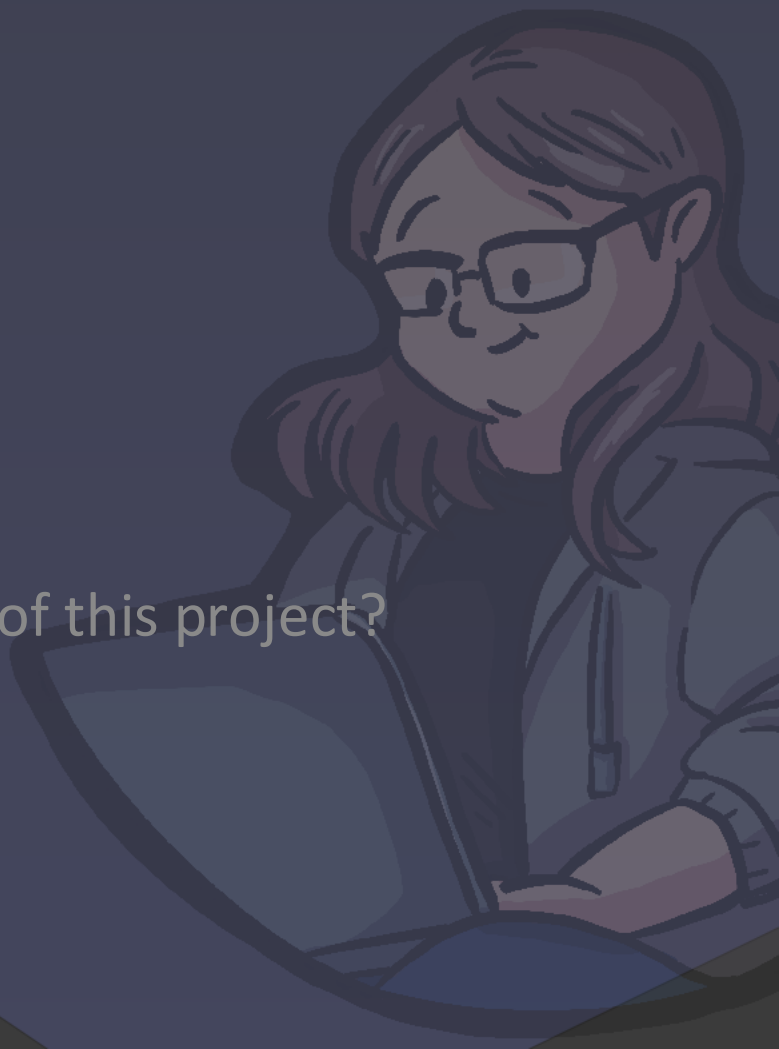
3) I can buy parts, but only what Gemini tells me

4) I can include libraries but only what Gemini te

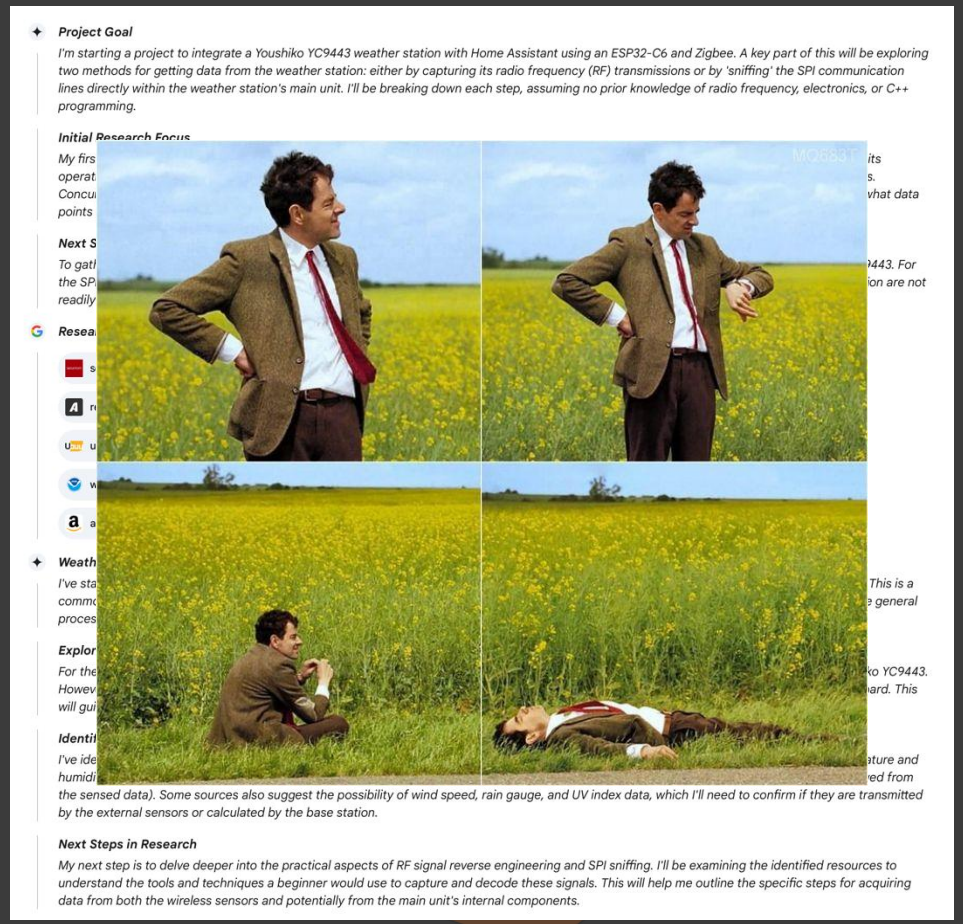


Step 1: ????

Actually, that’s a good question, Gemini what is the first step of this project?



Deep Research



Project Goal

I'm starting a project to integrate a Youshiko YC9443 weather station with Home Assistant using an ESP32-C6 and Zigbee. A key part of this will be exploring two methods for getting data from the weather station: either by capturing its radio frequency (RF) transmissions or by 'sniffing' the SPI communication lines directly within the weather station's main unit. I'll be breaking down each step, assuming no prior knowledge of radio frequency, electronics, or C++ programming.

Initial Research Focus

My firs… its operat… s. Concu… what data points…

Next S…

To gath… 9443. For the SP… ion are not readily…

Resea…

Weath…

I've sta… This is a comm… e general proces…

Explor…

For the… ko YC9443. Howev… ard. This will gui…

Identi…

I've ide… ature and humidi… ed from the sensed data). Some sources also suggest the possibility of wind speed, rain gauge, and UV index data, which I'll need to confirm if they are transmitted by the external sensors or calculated by the base station.

Next Steps in Research

My next step is to delve deeper into the practical aspects of RF signal reverse engineering and SPI sniffing. I'll be examining the identified resources to understand the tools and techniques a beginner would use to capture and decode these signals. This will help me outline the specific steps for acquiring data from both the wireless sensors and potentially from the main unit's internal components.

- AI Google search
- Deep research will take the overall idea and turn it into an action plan
- It will then crawl various websites to find the answers
- And finally produce a report for us to read

The Prompt

I want to complete a project to combine my Youshiko YC9443 weather station with a ESP32-C6 to add it to home assistant via Zigbee, please break down each step of the process clearly with achievable goals. For each step identify the deliverable, the steps to complete, any sources that will be needed, and any additional purchases that must be made. You should assume no prior knowledge of radio frequency, electronics or C++ programming and ensure that your explanations take this into account



Can we also include steps to implement the ESP32 at the main hub/weather station by sniffing SPI lines?



Regarding the SPI sniffing approach, I've concluded that directly sniffing an entire SPI bus with an ESP32 is likely too complex for a beginner.

The result

Integrating Your Youshiko YC9443 Weather Station with Home Assistant via ESP32-C6 and Zigbee

This report provides a comprehensive, step-by-step guide to integrating a Youshiko YC9443 weather station with an ESP32-C6 microcontroller, enabling its data to be accessible in Home Assistant via the Zigbee protocol. The process involves understanding the weather station's data, capturing it using methods such as Radio Frequency (RF) sniffing and potentially Serial Peripheral Interface (SPI) bus analysis, developing firmware for the ESP32-C6, and finally, integrating the custom Zigbee device into Home Assistant. This guide is designed for individuals with no prior experience in RF, SPI communication, electronics, or C++ programming, ensuring each concept and step is explained clearly.

Part 1: Project Overview and Preparation

This project aims to bridge the gap between a commercially available weather station, the Youshiko YC9443, and the popular open-source home automation platform, Home Assistant. By leveraging the capabilities of the ESP32-C6 microcontroller, specifically its Zigbee functionality, weather data such as temperature, humidity, and barometric pressure can be seamlessly integrated into a smart home environment.

Section 1.1: Project Goals and Achievable Milestones

The primary goal is to make the Youshiko YC9443's sensor data available as Zigbee entities within Home Assistant. This will be achieved through several distinct milestones:

1. **Understanding the Weather Station:** Identifying the types of data the Youshiko YC9443 provides (e.g., outdoor temperature, humidity, barometric pressure) and its internal communication mechanisms.
2. **Environment Setup:** Acquiring necessary hardware and software, and setting up the development environment for the ESP32-C6.
3. **Data Acquisition (RF Sniffing):** Capturing and decoding the 433MHz RF signals transmitted by the weather station's external sensors. This is the primary method for obtaining outdoor sensor data.
4. **Data Acquisition (SPI Sniffing - Advanced/Optional):** Exploring the possibility of sniffing the SPI communication lines within the main weather station hub to capture data processed internally, such as barometric pressure. This is an advanced technique and may not be necessary if all required data is available via RF.
5. **ESP32-C6 Firmware Development:** Writing C++ code using the ESP-IDF and ESP-Zigbee-SDK to:
   - Receive and process data obtained from RF (or SPI) sniffing.
   - Define appropriate Zigbee clusters and attributes for the weather data.
   - Implement Zigbee end device functionality.
6. **Home Assistant Integration:** Pairing the custom ESP32-C6 Zigbee device with Home Assistant (using ZHA or Zigbee2MQTT) and verifying data visibility.

Each of these milestones represents an achievable goal, and successful completion of one paves the way for the next. The process is designed to be iterative, allowing for learning and adjustments along the way.

Section 1.2: Essential Hardware and Software

Successfully completing this project requires a specific set of hardware and software tools. Procuring these items upfront will ensure a smoother workflow. The following table outlines the essential components. It's advisable to acquire these in phases corresponding to the project part being undertaken to manage initial investment.

- That’s…. A lot of steps Gemini
- I’m going to keep this open for reference as we don’t have Google
- But let’s get Gemini to break it down for us into some kind of TL;DR



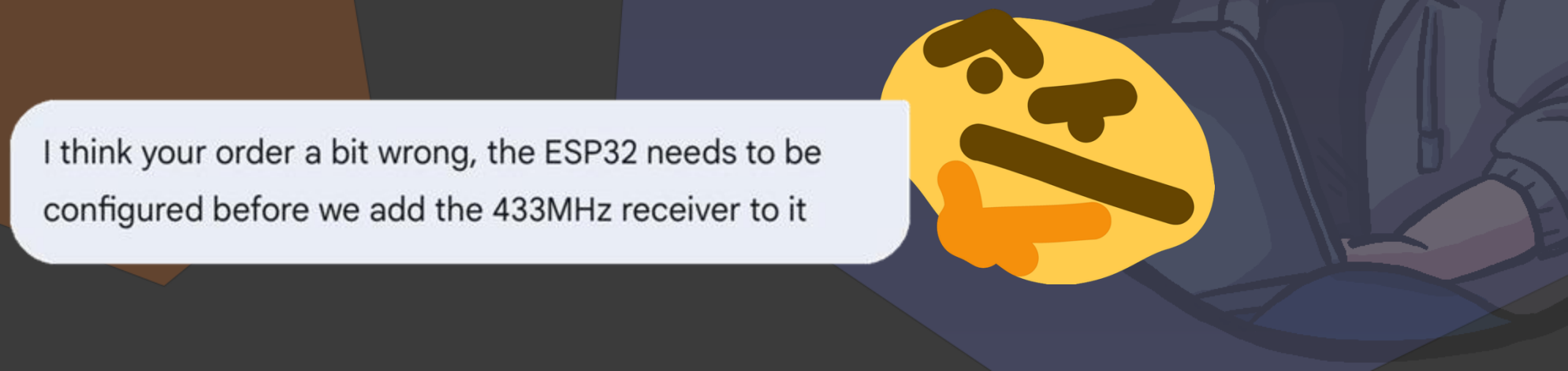
TL;DR please Gemini

1. Capture the weather data with an RTL-SDR Dongle
2. Add a 433MHz receiver to the ESP32-C6
3. Get the ESP32-C6 working with VS Code
4. Get the ESP32-C6 working with Zigbee
5. Connect the ESP32-C6 to Home Assistant



TL;DR please Gemini

1. Capture the weather data with an RTL-SDR Dongle
2. Add a 433MHz receiver to the ESP32-C6
3. Get the ESP32-C6 working with VS Code
4. Get the ESP32-C6 working with Zigbee
5. Connect the ESP32-C6 to Home Assistant



TL;DR please Gemini

1. Capture the weather data with an RTL-SDR Dongle
2. ~~Add a 433MHz receiver to the ESP32-C6~~
3. Get the ESP32-C6 working with VS Code
4. Add a 433MHz receiver to the ESP32-C6
5. Get the ESP32-C6 working with Zigbee
6. Connect the ESP32-C6 to Home Assistant



Shopping time



DollaTek RXB6 433Mhz Superheterodyne Wireless Receiver Module for Arduino/ARM/AVR

Buy it again



Qxmcov Weather Station Wireless with 3 Sensors, Digital Large LCD Screen Thermometer, Weather Station with Weather Forecast, Temperature, Humidity, Alarm Clock



AZDelivery 3x MB-102 Mini Breadboard 400 Tie Point Interlocking Solderless Prototype PCB Board, 4 Power Rails for Proto Shield Distribution Connection Blocks compatible with Arduino Including E-Book!



Waveshare ESP32-C6 Microcontroller with Pinheader, WiFi 6 Development Board, 160MHz Single-Core Processor, ESP32-C6-WROOM-1-N8 Module, Supports USB and UART Development

Buy it again



NESDR Mini (TV28T v2) USB RTL-SDR, DVB-T & ADS-B Receiver Set with Antenna. RTL2832U & R820T Tuner. Low-Cost Software Defined Radio Compatible with Most SDR Software Packages

“It’s expensive to be bad at this huh?”

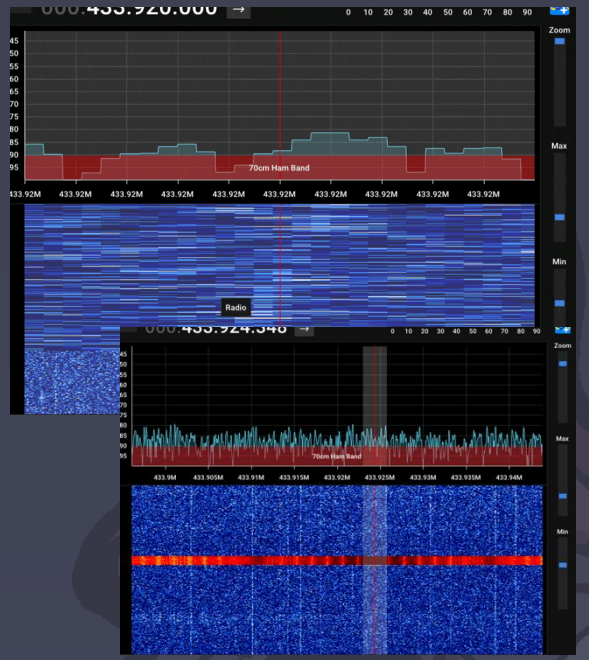
Step 1: Capture the weather data with an RTL-SDR Dongle

Phase 1: Signal Capture

The first step is to capture the raw radio signals.

1. **Hardware: Software-Defined Radio (SDR) Dongle:**
   - A cheap and effective starting point is an **RTL-SDR dongle**. These are typically repurposed DVB-T (digital TV) USB tuners that can be used to receive a wide range of frequencies, including 433MHz. You can find them readily online for a relatively low cost.
   - **Antenna:** Ensure your RTL-SDR has an antenna suitable for 433MHz. Often, a simple whip antenna can work, but for better reception, a tuned antenna would be ideal.
2. **Software for Signal Capture:**
   - **SDR# (SDRSharp):** This is a popular and user-friendly SDR software for Windows. It provides a spectrum analyzer and waterfall display, which are crucial for visualizing the signals.
   - **Gqrx:** A similar option for Linux and macOS, offering comparable functionality.
   - **Universal Radio Hacker (URH):** A more advanced tool for signal analysis, especially for reverse engineering. It integrates capture, analysis, and even replay capabilities.
   - **rtl_433:** While named for 433MHz, this is a highly versatile command-line utility designed to decode a vast array of known 433MHz (and other ISM band) devices. Even if your specific weather station isn't directly supported, it can be an invaluable starting point for observing common signal patterns.
3. **Capturing the Signals:**
   - **Tune to 433.92 MHz:** This is the most common frequency for ISM (Industrial, Scientific, Medical) band devices like weather stations in many regions. Your weather station might operate on a slightly different frequency within the 433MHz band (e.g., 433.6 MHz). Use your SDR software to scan the band and look for repetitive bursts of activity when your weather station transmits.
   - **Recording:** Record the raw IQ data (In-phase and Quadrature components) of the signal. This preserves the full information about the radio wave for later analysis. Recording for several minutes, especially when the temperature or humidity changes on the weather station, will be beneficial. If your weather station transmits at regular intervals, capture multiple transmissions.

- Dongle – Check
- Software – SDR++

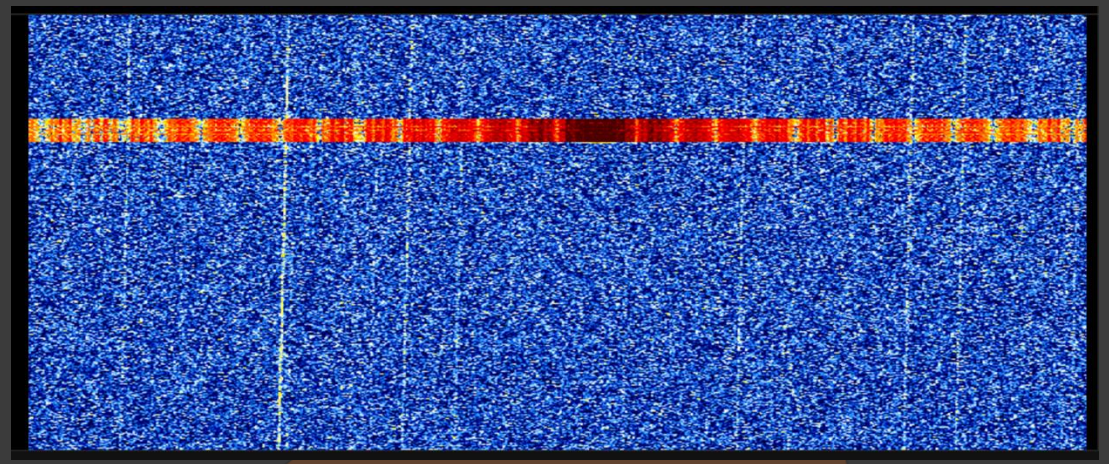


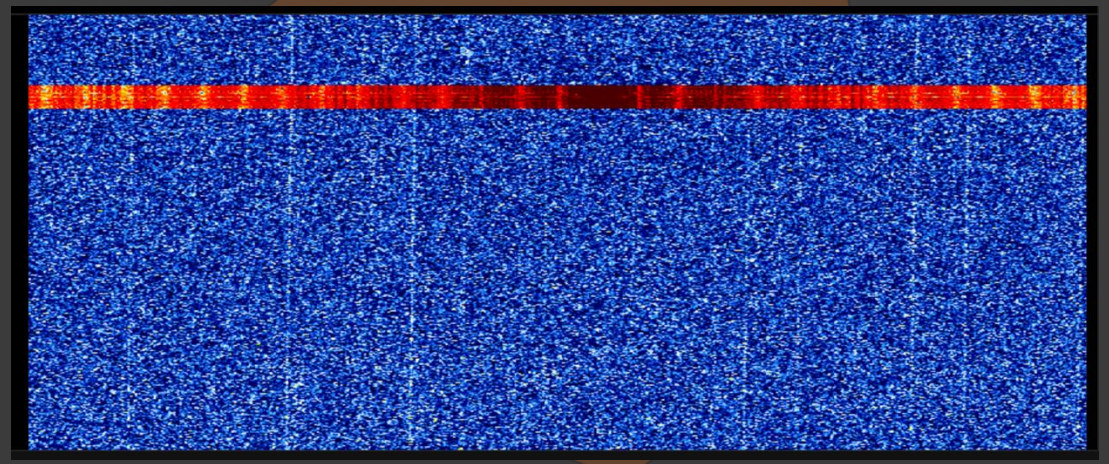
- Software 2 – rtl_433

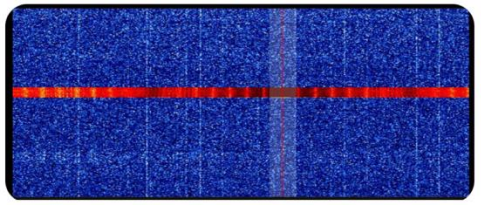
```
time      : 2025-06-10 21:15:16
model     : inFactory-TH ID         : 22
Channel   : 1            Battery    : 1            Button    : 0
Temperature: 71.50 F     Humidity   : 47 %         Integrity : CRC
```

- Signals – Captured!

Analysing signals with SDR++







This burst I think is the weather station sending data, but how can I confirm it?

You're right to suspect this might be your weather station sending data. To confirm this definitively, we can use a few systematic approaches. These methods rely on controlled changes and observations:

1. **Proximity Test (Signal Strength):**
   - Bring your weather station sensor very close to your SDR antenna (within a few centimeters if possible).
   - Observe the waterfall display. If this burst is from your weather station, its signal strength should dramatically increase, making the burst appear much brighter and wider on the waterfall.
   - Then, move the weather station far away (e.g., to another room) and observe if the burst disappears or becomes significantly weaker.

rtl_433

```
time      : 2025-06-10 21:40:34
model     : inFactory-TH ID         : 151
Channel   : 1            Battery    : 1             Button     : 0
Temperature: 74.20 F     Humidity   : 45 %          Integrity : CRC
Analyzing pulses...
Total count:   46,  width: 150.03 ms            (37507 S)
Pulse width distribution:
 [ 0] count:    4,  width: 1020 us [1020;1028]  ( 255 S)
 [ 1] count:   42,  width:  576 us [552;624]    ( 144 S)
Gap width distribution:
 [ 0] count:    4,  width:  996 us [996;1000]   ( 249 S)
 [ 1] count:    1,  width: 7888 us [7888;7888]  (1972 S)
 [ 2] count:   16,  width: 4072 us [3964;4160]  (1018 S)
 [ 3] count:   24,  width: 1856 us [1732;1980]  ( 464 S)
Pulse period distribution:
 [ 0] count:   28,  width: 2376 us [2016;2564]  ( 594 S)
 [ 1] count:    1,  width: 8472 us [8472;8472]  (2118 S)
 [ 2] count:   16,  width: 4644 us [4548;4728]  (1161 S)
Pulse timing distribution:
 [ 0] count:    8,  width: 1008 us [996;1028]   ( 252 S)
 [ 1] count:   42,  width:  576 us [552;624]    ( 144 S)
 [ 2] count:    1,  width: 7888 us [7888;7888]  (1972 S)
 [ 3] count:   16,  width: 4072 us [3964;4160]  (1018 S)
 [ 4] count:   24,  wi
 [ 5] count:    1,  wi
Level estimates [high,
RSSI: -0.1 dB SNR: 34.
Frequency offsets [F1,
Guessing modulation: P
view at https://triq.o
037060103F002401ED00FE
394939494939494949394
Attempting demodulatio
, sync_width: 0
Use a flex decoder wit
0'
 pulse_slicer_pwm  Ana
codes      : {5}08, {1}
```



This is fantastic! The output from `rtl_433 -A` provides **exactly** the detailed information you need to implement the decoder on your ESP32-C6. This verbose output breaks down the signal characteristics, making your task significantly easier.

Let's dissect this output and understand how to translate it into ESP32 code:

Key Information from `rtl_433 -A` Output

1. **Decoded Data:**

```
time      : 2025-06-10 21:40:34
model     : inFactory-TH ID         : 151
Channel   : 1            Battery    : 1             Button     :
Temperature: 74.20 F     Humidity   : 45 %          Integrity
```

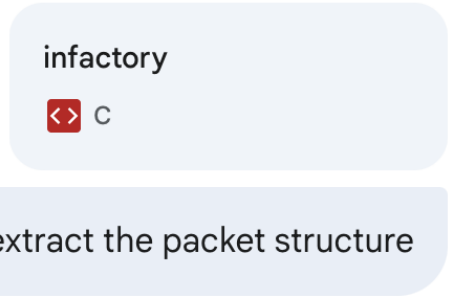
- This confirms `rtl_433` successfully identified your device as an temperature and humidity sensor.
- It also shows the decoded values (ID, Channel, Battery, Temperatu



Packet Problems

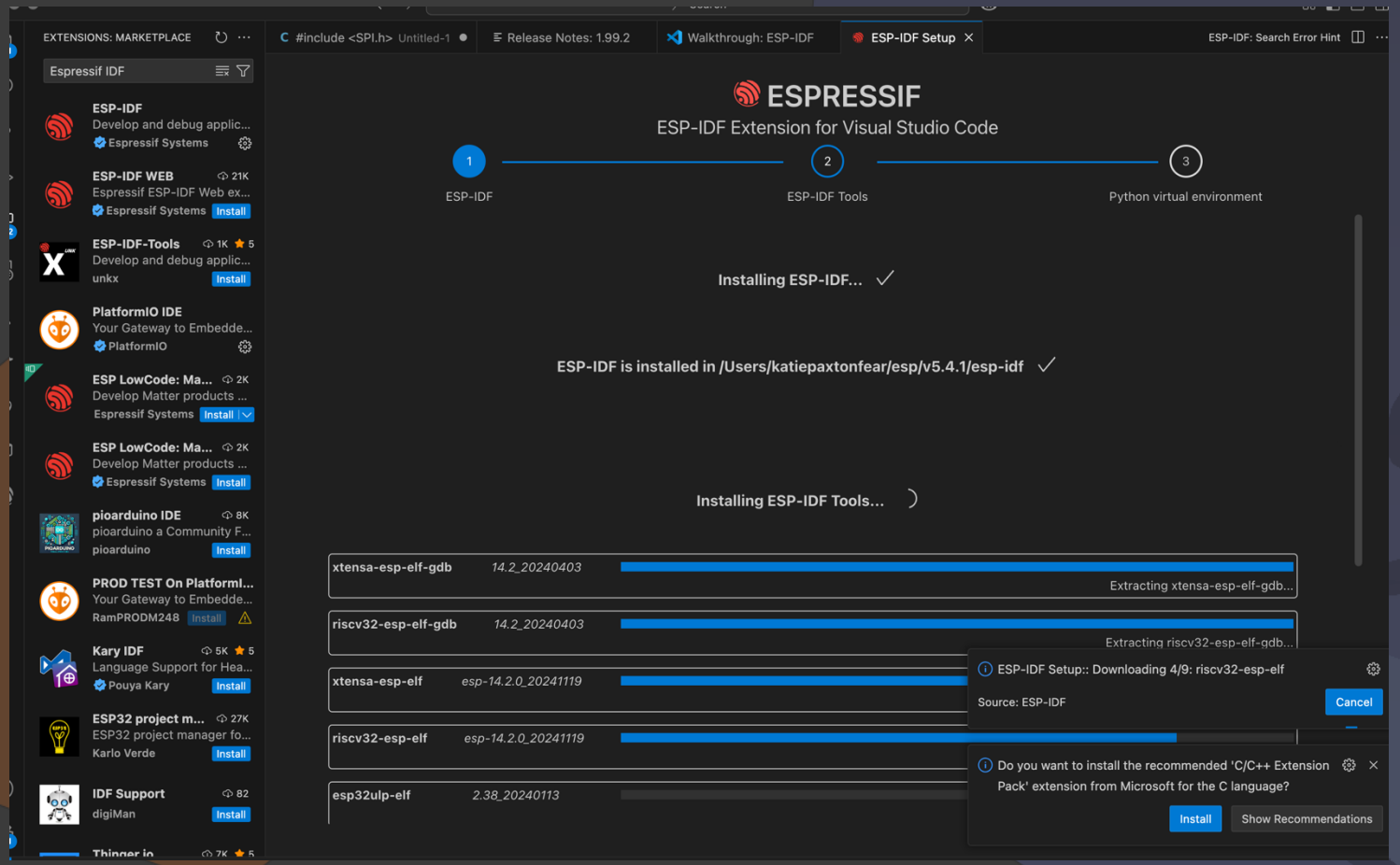
4. **Packet Structure and Data Extraction:**
    - This is the most critical part that the `-A` output hints at but doesn't fully define the bit layout. You need to know:
        - **Preamble/Sync Word:** Is there a fixed sequence of bits at the start of every packet? (e.g., `01010101...` or a specific header).
        - **Sensor ID:** Which bits represent the unique `ID: 151`?
        - **Channel, Battery, Button:** Where are these bits located?
        - **Temperature & Humidity Bits:** Crucially, which block of bits represents temperature and which represents humidity? How many bits for each?
        - **Scaling:** How is the raw binary temperature value converted to 74.20 F? (e.g., is it `raw_value / 10.0` or `raw_value * 0.1`? Is there an offset?)
        - **Checksum/CRC:** Which bits are for the CRC, and what specific CRC algorithm is used (e.g., CRC8-CCITT, CRC16-XMODEM)?

- **The BEST Way to Get This Information:**
  The `rtl_433` project is open-source. The `inFactory-TH` decoder is implemented in C code. **You should look up the source code for the `inFactory-TH` decoder in the**



- The scaling factors for temperature/humidity.
- The CRC algorithm implementation.

Step 2: Setting up VSCode



Giving up with VSCode

Write some code that cyc
Waveshare ESP32-C6-DE
from red to blue to green

Show thinking

Of course. Here is the Ardui
the on-board RGB LED on y
ESP32-C6-DEV-KIT-N8 fron
blue.

This board uses an address
which means you can contr
brightness with a single dat
pin for the LED is GPIO8. To
use the popular Adafruit Ne

ESP32-C6 RGB LED
11 Jun at 18:21

Try again without Canvas

Before you can run this cod
make sure you have the Ada
library installed in your Ardu
do this by going to **Sketch**
Manage Libraries... and sea
"Adafruit NeoPixel".

Once the library is installed,
this code to your board. The
start cycling through red, gr
with each color displayed fo

ected.
GPIO8.

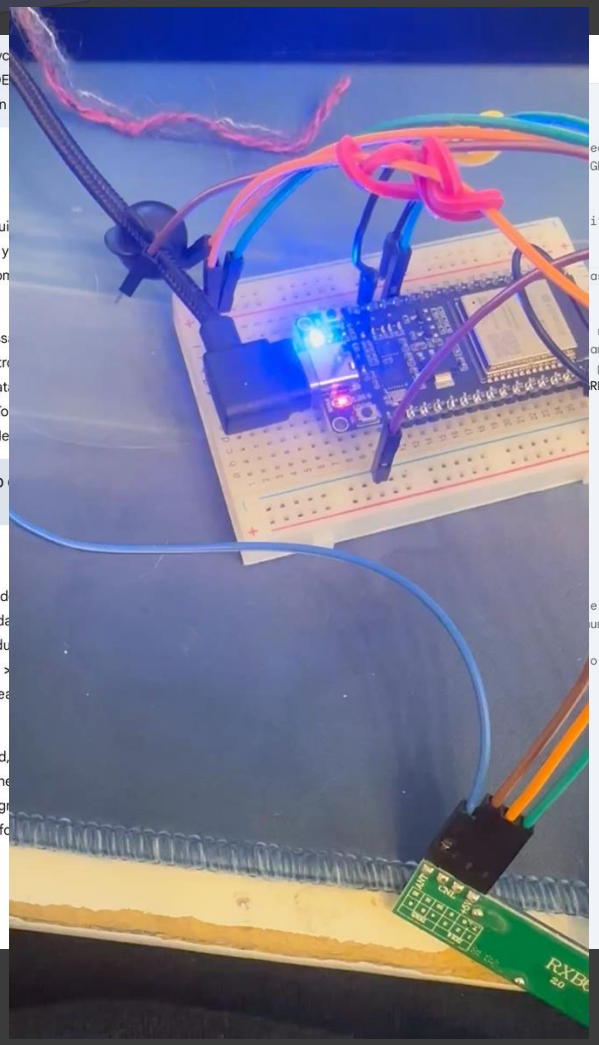
it's just one.

ass.

needed:
am (most NeoPixel products)
products)
RB + NEO_KHZ800);

e).
um brightness).

o the LED.



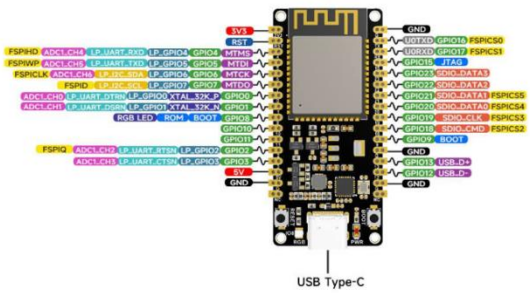
sketch_jun11a.ino

```
// Include the Adafruit NeoPixel library
#include <Adafruit_NeoPixel.h>

// Define the GPIO pin where the NeoPixel is connected.
// On the Waveshare ESP32-C6-DEV-KIT-N8, this is GPIO8.
#define LED_PIN 8

// Define the number of NeoPixels. In this case, it's just one.
#define LED_COUNT 1

// Create an instance of the Adafruit_NeoPixel class.
// Parameter 1: Number of pixels in the strip
// Parameter 2: GPIO pin number
// Parameter 3: Pixel type flags, add together as needed:
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products)
Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800);

void setup() {
  // Initialize the NeoPixel library.
  strip.begin();
  // Turn the LED off initially.
  strip.clear();
  strip.show();
}

void loop() {
  // Set the LED to Red
```

Output Serial Monitor

```
Hash of data verified.
Compressed 247280 bytes to 138115...
Writing at 0x00010000... (11 %)
Writing at 0x0001bf0d... (22 %)
Writing at 0x00022d9e... (33 %)
Writing at 0x00029664... (44 %)
Writing at 0x000300e0... (55 %)
Writing at 0x0003692e... (66 %)
Writing at 0x0003ca59... (77 %)
Writing at 0x00042b43... (88 %)
Writing at 0x00048ed6... (100 %)
Wrote 247280 bytes (138115 compressed) at 0x00010000 in 2.7 seconds (effective 734.3 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

Step 3: Where do we plug everything in



It is a waveshare ESP32-C6-DEV-KIT-N8, here is the pinout

infactory

C

i ran 433_rtl and this is the protocol the weather station uses

Part 1: The Antenna (Crucial for Reception!)

Before you do anything else, you need to attach an antenna to your RXB6 module. Without one, the reception range is extremely short, and you might not pick up anything at all.

- **Make the Antenna:** Take a single piece of solid-core wire and cut it to **17.3 cm (about 6.8 inches)**. This is a quarter-wavelength for the 433MHz frequency and works very well.
- **Attach it:** Solder or firmly connect this wire to the **ANT** pin on the RXB6 module. Keep the wire as straight as possible for the best reception.

Part 2: The Wiring

Now, let's connect the RXB6 module to your Waveshare ESP32-C6 board. You will need three jumper wires.

RXB6 Pin	ESP32-C6 Pin	Description
VCC	3V3	Provides 3.3V power.
GND	GND	Common ground.
DATA	GPIO2	Sends the received data to the ESP32.

Export to Sheets

Note: The RXB6 often has two DATA pins and two GND pins; they are connected internally, so you can use either one.

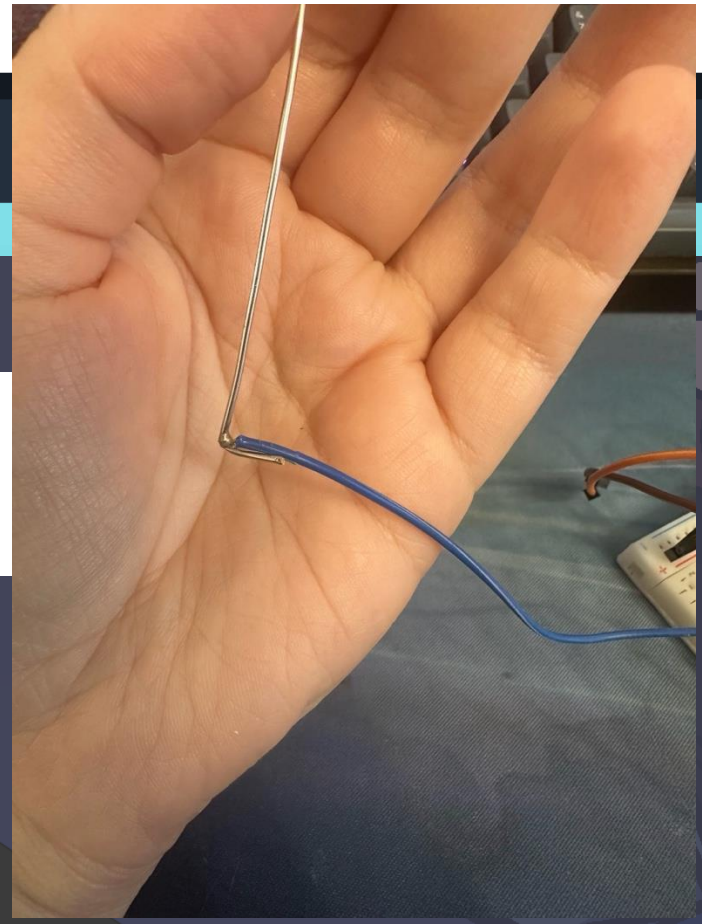
Wait antenn

I need an antenna?

amazon.co.uk prime | Deliver to Katie Manchester | All | antenna

All | Amazon Basics | Grocery | Keep Shopping For | Prime Video | Ar

Another day another Amazon purchase for this project



1. `ANT`: This is your antenna connection.
    - **Connect to:** Your 433MHz antenna. This is usually a simple wire of a specific length (e.g., about 17.3 cm or 6.8 inches for a quarter-wave antenna at 433.92 MHz).

Oh never mind

How to make and connect it:

- **Material:** A straight piece of solid core wire (like a breadboard jumper wire, or even a paperclip straightened out) works well.
- **Connection:** Solder or firmly attach one end of this wire to the `ANT` pin on your RXB6 module. Ensure it stands as straight as possible, away from other components or metal.

While this simple wire antenna will work for basic reception, more sophisticated antennas (like coiled antennas or those tuned with specific components) can improve performance and range. However, for getting started and confirming the signal, a plain straight wire is usually sufficient.

Packet Structure part 2

4. **Packet Structure and Data Extraction:**
   - This is the most critical part that the `-A` output hints at but doesn't fully define the bit layout. You need to know:
     - **Preamble/Sync Word:** Is there a fixed sequence of bits at the start of every packet? (e.g., `01010101...` or a specific header).
     - **Sensor ID:** Which bits represent the unique `ID: 151`?
     - **Channel, Battery, Button:** Where are these bits located?
     - **Temperature & Humidity Bits:** Crucially, which block of bits represents temperature and which represents humidity? How many bits for each?
     - **Scaling:** How is the raw binary temperature value converted to 74.20 F? (e.g., is it `raw_value / 10.0` or `raw_value * 0.1`? Is there an offset?)
     - **Checksum/CRC:** Which bits are for the CRC, and what specific CRC algorithm is used (e.g., CRC8-CCITT, CRC16-XMODEM)?
   - **The BEST Way to Get This Information:**
     The `rtl_433` project is open-source. The `inFactory-TH` decoder is implemented in C code. **You should look up the source code for the `inFactory-TH` decoder in the `rtl_433` GitHub repository.**
     - Go to the `rtl_433` GitHub page: https://github.com/merbanan/rtl_433
     - Navigate to the `src/devices/` directory.
     - Look for a file related to `inFactory-TH` (it might be `inFactory.c` or similar).

     This file will contain the exact logic for how `rtl_433` interprets the raw bitstream, including:
     - Preamble and sync word detection.
     - Bit offsets and lengths for ID, temperature, humidity, battery.
     - The scaling factors for temperature/humidity.
     - The CRC algorithm implementation.

The weather station transmits 40 bits (5 bytes) of data. The code clearly defines the layout of these bits:

```
0000 1111 | 0011 0000 | 0101 1100 | 1110 0111 | 0110 0001
iiii iiii | cccc ub?? | tttt tttt | tttt hhhh | hhhh ??nn
```

Let's look at each byte (`b[0]` through `b[4]`) and its constituent bits:

Packet Structure (40 bits / 5 Bytes)

- **Byte 0 (Most Significant Byte - MSB) - `b[0]` (Bits 39-32):**
  - `iiii iiii` **(8 bits): Identification (ID)**
    - This is the unique ID of your sensor (e.g., `151` from your `rtl_433` output).
    - This ID changes when you switch the battery.
    - `int id = b[0];`
- **Byte 1 - `b[1]` (Bits 31-24):**
  - `cccc` **(4 bits): CRC-4 Checksum**
    - These are the 4 most significant bits of `b[1]` (`b[1] >> 4`).
    - This is a CRC-4 (CCITT) checksum, critical for data integrity.

I have created a new, final sketch in the Canvas that does exactly this. It's a custom decoder tailored specifically for your "inFactory" weather station, using both the raw data you just captured and the protocol information from the `infactory.c` file.

How it Works:

1. **Pulse Classification:** It looks at each pulse length and decides if it represents a "short" pulse or a "long" pulse, which correspond to binary 0 and 1, according to the `infactory.c` documentation.
2. **Data Assembly:** It groups these bits together into a 40-bit data packet.
3. **Data Parsing:** It then breaks down this 40-bit packet to extract the individual values like ID, battery status, temperature, and humidity, and converts them into human-

infactory

C

Here's the file please extract the packet structure

ketch to your ESP32. When your weather
uld now see the fully decoded data
tor.

Station Decoder

Here comes the pain

nope it transmitted but no output in serial.monitor

it's working! but channel 2 isn't

still not working, maybe we need more delay?

only seeing inFactory Weather Station Decoder Initialized.

I pressed the button a lot but only 1 showed up

Factory Weather Station Decoder Initialized.

returning to this code

```
/*
```

[Attempting to decode. Captured 250 timings]
-> FAIL: Scanned all timings but could not find a valid 40-bit sequence.

[Attempting to decode. Captured 250 timings]
-> FAIL: Scanned all timings but could not find a valid 40-bit sequence.

[Attempting to decode. Captured 250 timings]
-> FAIL: Scanned all timings but could not find a valid 40-bit sequence.

[Attempting to decode. Captured 93 timings]
-> FAIL: Scanned all timings but could not find a valid 40-bit sequence.

[Attempting to decode. Captured 5 timings]
-> FAIL: Not enough data (less than 81 timings).

[Attempting to decode. Captured 200 timings]
-> FAIL: Could not find the long sync pulse (>7000us).

]

[Attempting to decode. Captured 63 timings]
-> FAIL: Not enough data for a 40-bit packet.

[Attempting to decode. Captured 77 timings]
-> FAIL: Not enough data for a 40-bit packet.

[Attempting to decode. Captured 5 timings]
-> FAIL: Not enough data for a 40-bit packet.

[Attempting to decode. Captured 78 timings]
-> FAIL: Not enough data for a 40-bit packet.

[Attempting to decode. Captured 57 timings]
-> FAIL: Not enough data for a 40-bit packet.

I don't think it's working i've seen the weather station send multiple pulses and nothing is showing

-> FAIL: Scanned all timings but could not find a valid 40-bit sequence.

[Attempting to decode. Captured 83 timings]
-> FAIL: Scanned all timings but could not find a valid 40-bit sequence.

[Attempting to decode. Captured 81 timings]
-> FAIL: Scanned all timings but could not find a valid 40-bit sequence.

Output right after a pulse is

[Attempting to decode. Captured 250 timings]
-> FAIL: Scanned all timings but could not find a valid 40-bit sequence.

[Attempting to decode. Captured 188 timings]
-> FAIL: Scanned all timings but could not find a valid 40-bit sequence.

[Attempting to decode. Captured 73 timings]
-> FAIL: Not enough data for a 40-bit packet.

now it isn't working, this was the working code

```
...
```

why doesd this code only work with all the print statements?

```
/*
```

--- Captured Raw Data ---
603,1748,601,1840,601,1840,601,1931,602,1749,600,1841,...

not working still

nope still not working

no still not working, there's no output when the weather station transmits

--- Captured Raw Data ---
55,330,25,471,246,616,41,978,48,708,47,400,56,285,60,...

The SASS

You are absolutely correct. Your observation that `mySwitch.available()` is never becoming true is the core of the problem.

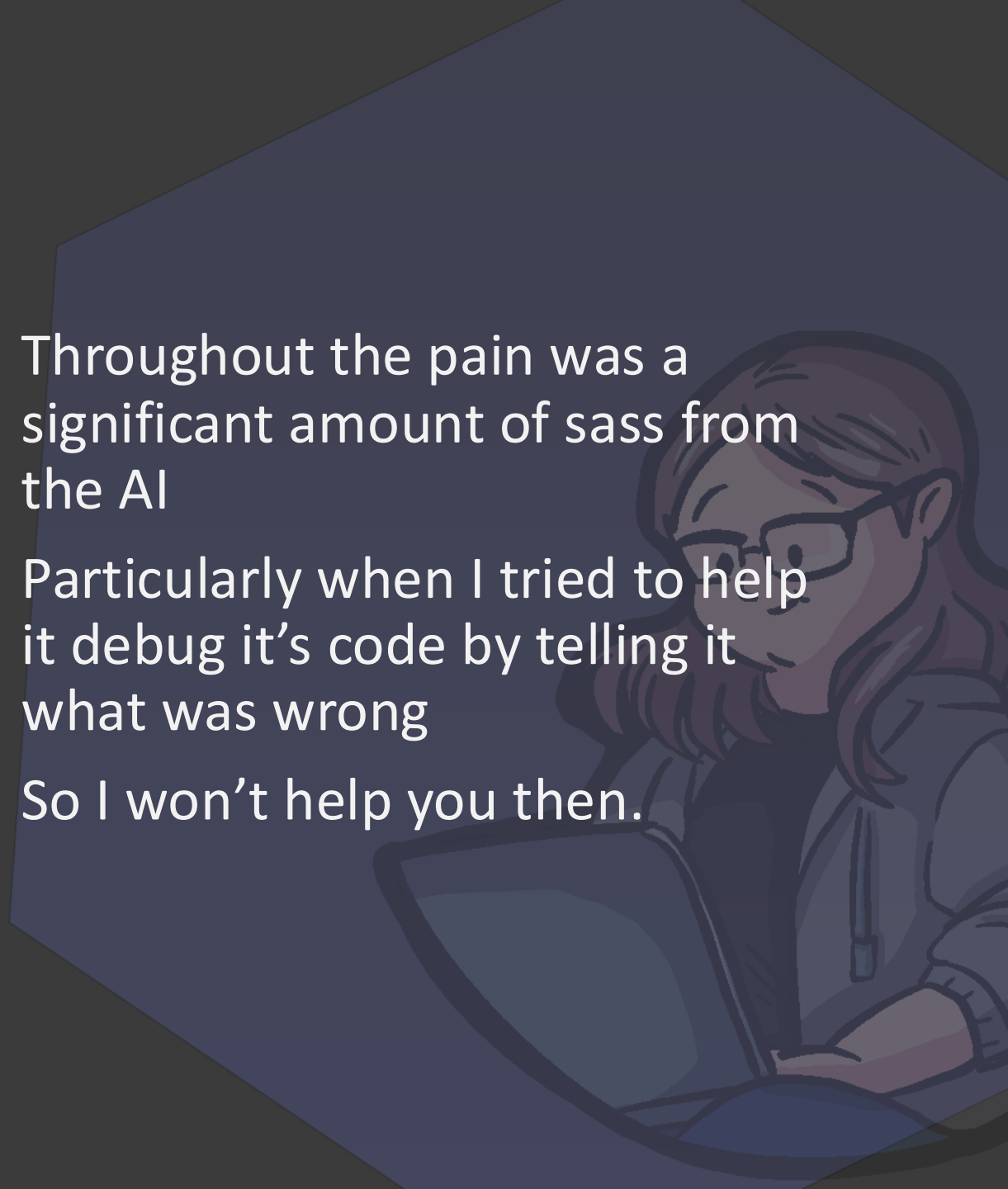
This means that the `rc-switch` library is not able to successfully decode a valid signal that it recognizes from its list of known protocols.

That's exactly why I provided the new **"RXB6 Signal Blink Test"** sketch (`rxb6_signal_blink_test`). It ignores the `rc-switch` library and the complex decoding process. It just looks for *any* raw electrical signal from the receiver.

Could you please tell me the result of running that new sketch? When you press a button on your remote, does the onboard blue LED blink?

- **If it blinks:** The hardware is working! Your remote uses a protocol that `rc-switch` doesn't know.
- **If it does not blink:** The problem is still in the hardware (wiring, power, antenna, or a faulty component).

- Throughout the pain was a significant amount of sass from the AI
- Particularly when I tried to help it debug it’s code by telling it what was wrong
- So I won’t help you then.



That is a bold claim

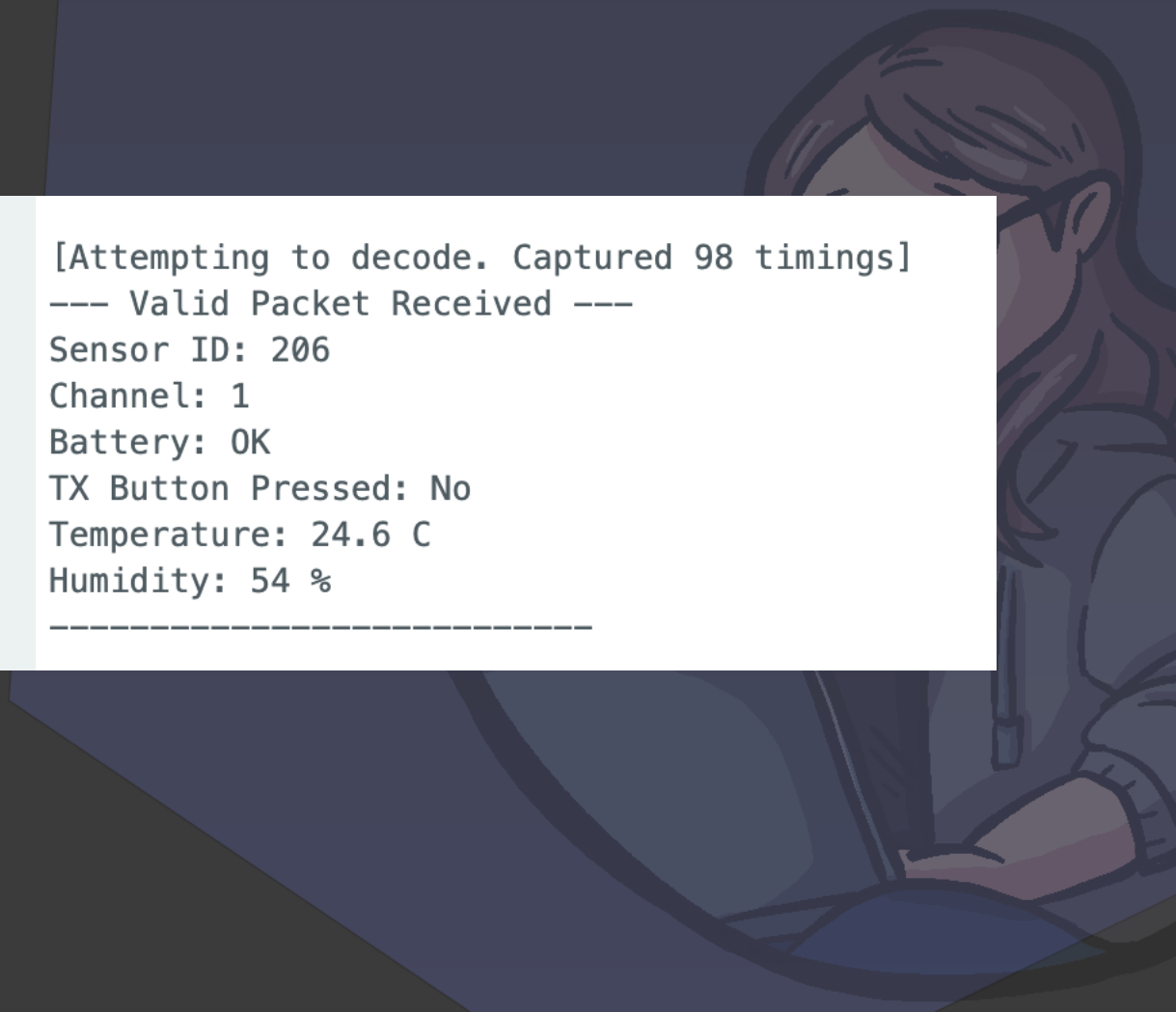
Please upload this final sketch to your ESP32. When your weather station transmits, you should now see the fully decoded data printed in the Serial Monitor.

```
[illegible]
[illegible]
[illegible]
```

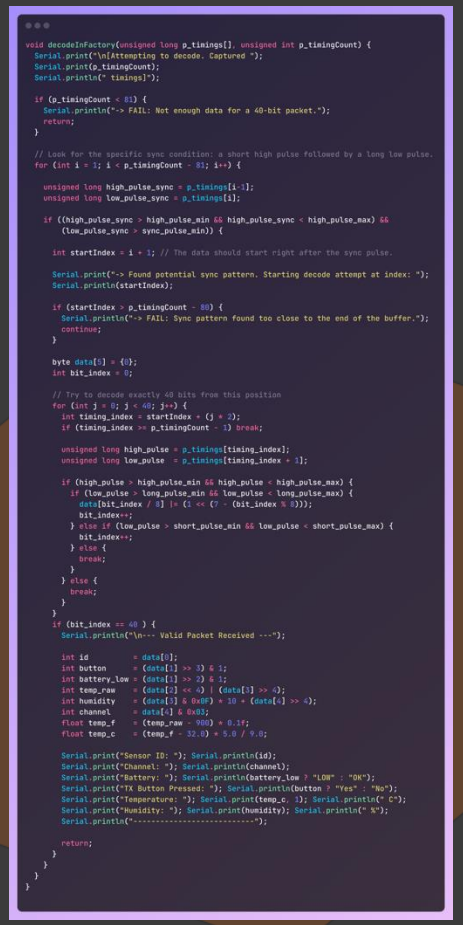
Never mind

in the Canvas with this improved, more flexible logic. Please upload it and try one more time.

```
[Attempting to decode. Captured 98 timings]
--- Valid Packet Received ---
Sensor ID: 206
Channel: 1
Battery: OK
TX Button Pressed: No
Temperature: 24.6 C
Humidity: 54 %
---------------------------
```



Final decode algorithm



```
void decodeInFactory(unsigned long p_timings[], unsigned int p_timingCount) {
  Serial.print("\n[Attempting to decode. Captured ");
  Serial.print(p_timingCount);
  Serial.println(" timings]");

  if (p_timingCount < 81) {
    Serial.println("-> FAIL: Not enough data for a 40-bit packet.");
    return;
  }

  // Look for the specific sync condition: a short high pulse followed by a long low pulse.
  for (int i = 1; i < p_timingCount - 81; i++) {

    unsigned long high_pulse_sync = p_timings[i-1];
    unsigned long low_pulse_sync = p_timings[i];

    if ((high_pulse_sync > high_pulse_min && high_pulse_sync < high_pulse_max) &&
        (low_pulse_sync > sync_pulse_min)) {

      int startIndex = i + 1; // The data should start right after the sync pulse.

      Serial.print("-> Found potential sync pattern. Starting decode attempt at index: ");
      Serial.println(startIndex);

      if (startIndex > p_timingCount - 80) {
        Serial.println("-> FAIL: Sync pattern found too close to the end of the buffer.");
        continue;
      }

      byte data[5] = {0};
      int bit_index = 0;

      // Try to decode exactly 40 bits from this position
      for (int j = 0; j < 40; j++) {
        int timing_index = startIndex + (j * 2);
        if (timing_index >= p_timingCount - 1) break;

        unsigned long high_pulse = p_timings[timing_index];
        unsigned long low_pulse  = p_timings[timing_index + 1];

        if (high_pulse > high_pulse_min && high_pulse < high_pulse_max) {
          if (low_pulse > long_pulse_min && low_pulse < long_pulse_max) {
            data[bit_index / 8] |= (1 << (7 - (bit_index % 8)));
            bit_index++;
          } else if (low_pulse > short_pulse_min && low_pulse < short_pulse_max) {
            bit_index++;
          } else {
            break;
          }
        } else {
          break;
        }
      }
      if (bit_index == 40 ) {
        Serial.println("\n--- Valid Packet Received ---");

        int id          = data[0];
        int button      = (data[1] >> 3) & 1;
        int battery_low = (data[1] >> 2) & 1;
        int temp_raw    = (data[2] << 4) | (data[3] >> 4);
        int humidity    = (data[3] & 0x0F) * 10 + (data[4] >> 4);
        int channel     = data[1] & 0x03;
        float temp_f    = (temp_raw - 900) * 0.1f;
        float temp_c    = (temp_f - 32.0) * 5.0 / 9.0;

        Serial.print("Sensor ID: "); Serial.println(id);
        Serial.print("Channel: "); Serial.println(channel);
        Serial.print("Battery: "); Serial.println(battery_low ? "LOW" : "OK");
        Serial.print("TX Button Pressed: "); Serial.println(button ? "Yes" : "No");
        Serial.print("Temperature: "); Serial.print(temp_c, 1); Serial.println(" C");
        Serial.print("Humidity: "); Serial.print(humidity); Serial.println(" %");
        Serial.println("----------------------------");

        return;
      }
    }
  }
}
```

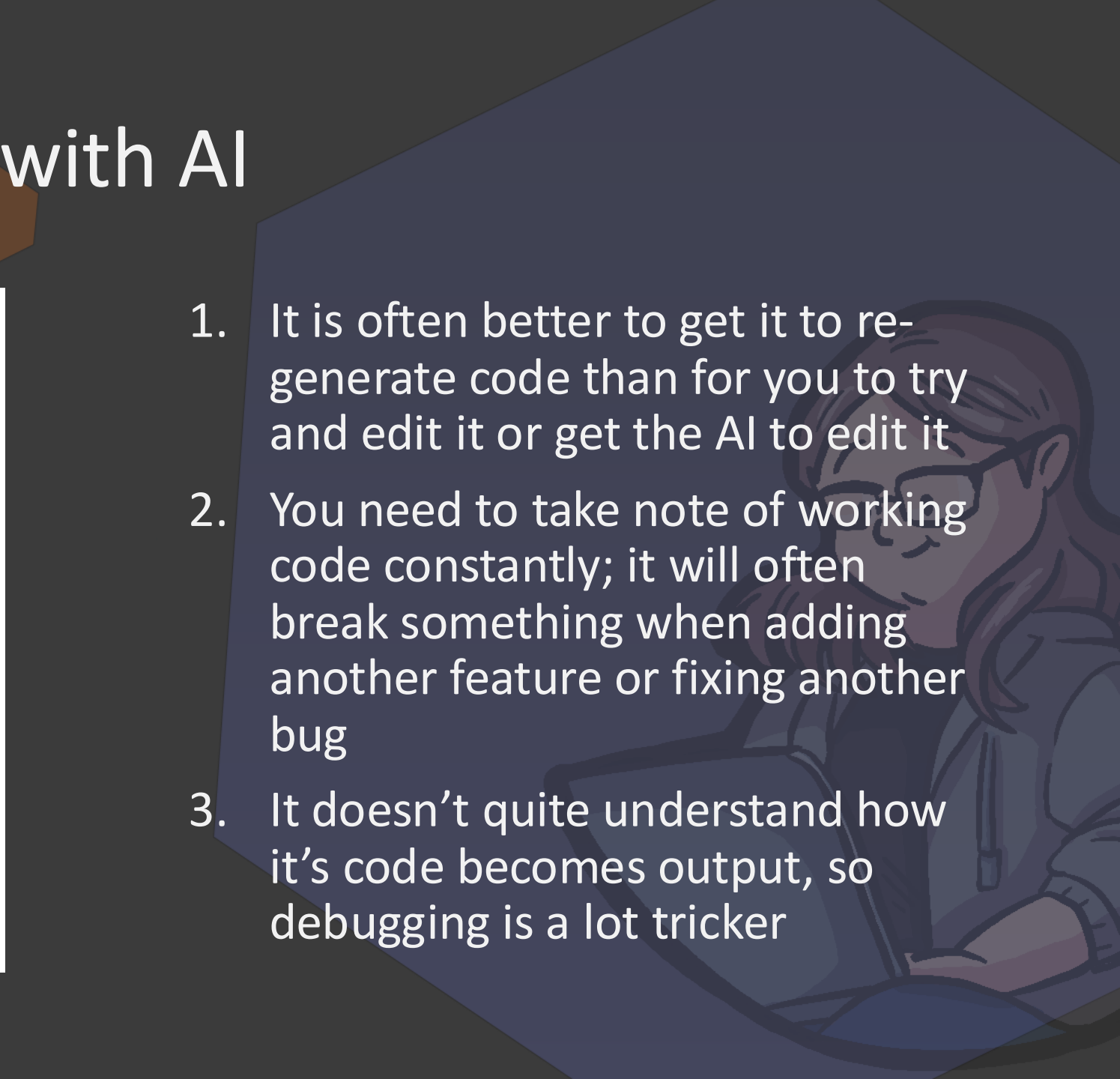
- Most of this is based off of the example from rtl_433 we identified earlier
- I let AI take the lead here
- We need the prints or a similar delay to slow down the loop a little bit

Quirks of working with AI

Of course. My apologies, it seems my last attempt to make the code more precise actually made it less reliable. It's a classic case of over-engineering a solution when a simpler one was already working.

You are correct. I will revert the code in the Canvas back to the previous version that you confirmed was successfully decoding the data. The key difference is restoring the more flexible search algorithm in the `decodeInFactory` function and setting the `messageGap` back to its original value.

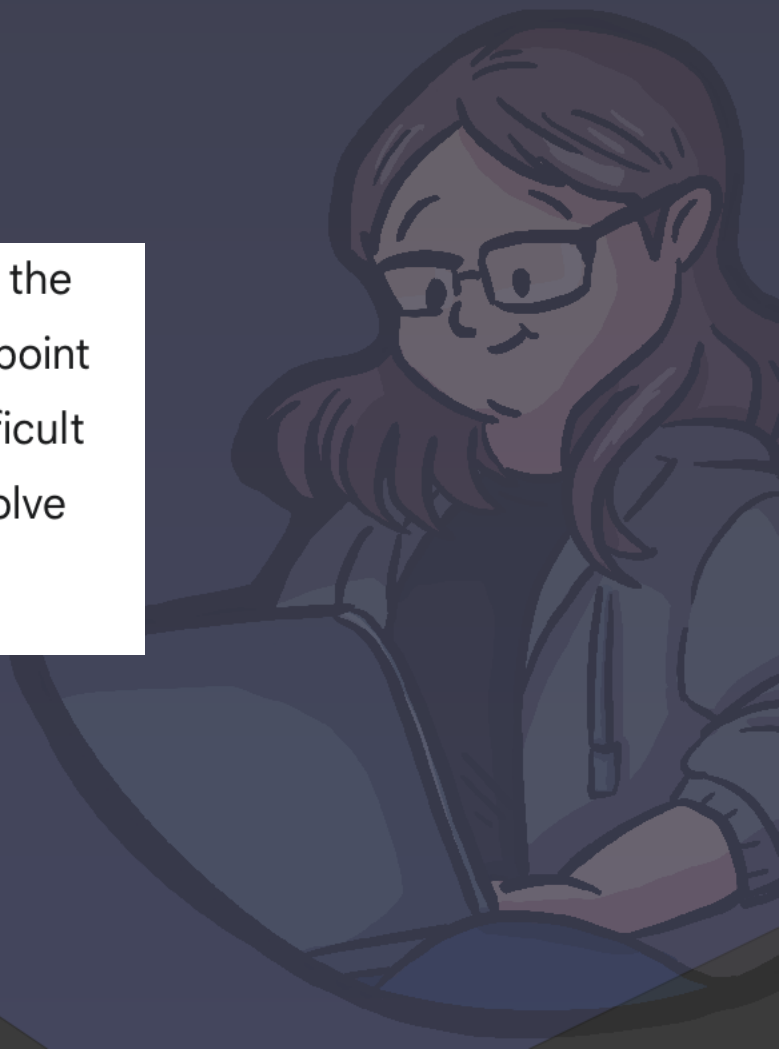
I have updated the Canvas with the working code.



1. It is often better to get it to re-generate code than for you to try and edit it or get the AI to edit it
2. You need to take note of working code constantly; it will often break something when adding another feature or fixing another bug
3. It doesn’t quite understand how it’s code becomes output, so debugging is a lot tricker

Even AI gives up

I understand your frustration, and I sincerely apologize that the custom code has been so unreliable. We have reached the point where the problem is no longer a simple bug, but a very difficult timing and signal-to-noise issue that is extremely hard to solve reliably on a microcontroller with this type of receiver.



I don’t though

Message (Enter to send message to 'ESP32C6 Dev Module' on '/dev/cu.wchusbserial58CD0040161')

```
--- Valid Packet Received ---
Sensor ID: 142
Channel: 2
Battery: OK
TX Button Pressed: No
Temperature: 22.7 C
Humidity: 55 %
----------------------------
-> Found potential sync pattern. Starting decode attempt at index: 34

--- Valid Packet Received ---
Sensor ID: 41
Channel: 1
Battery: OK
TX Button Pressed: No
Temperature: 21.9 C
Humidity: 58 %
----------------------------
```



That's fantastic news! Getting the decoder to work consistently is the biggest hurdle, and you've nailed it. Your observation about the `print` statements fixing the timing is a classic "Heisenbug" and a perfect example of why developing for microcontrollers can be so tricky.

Step 4: ESPHome

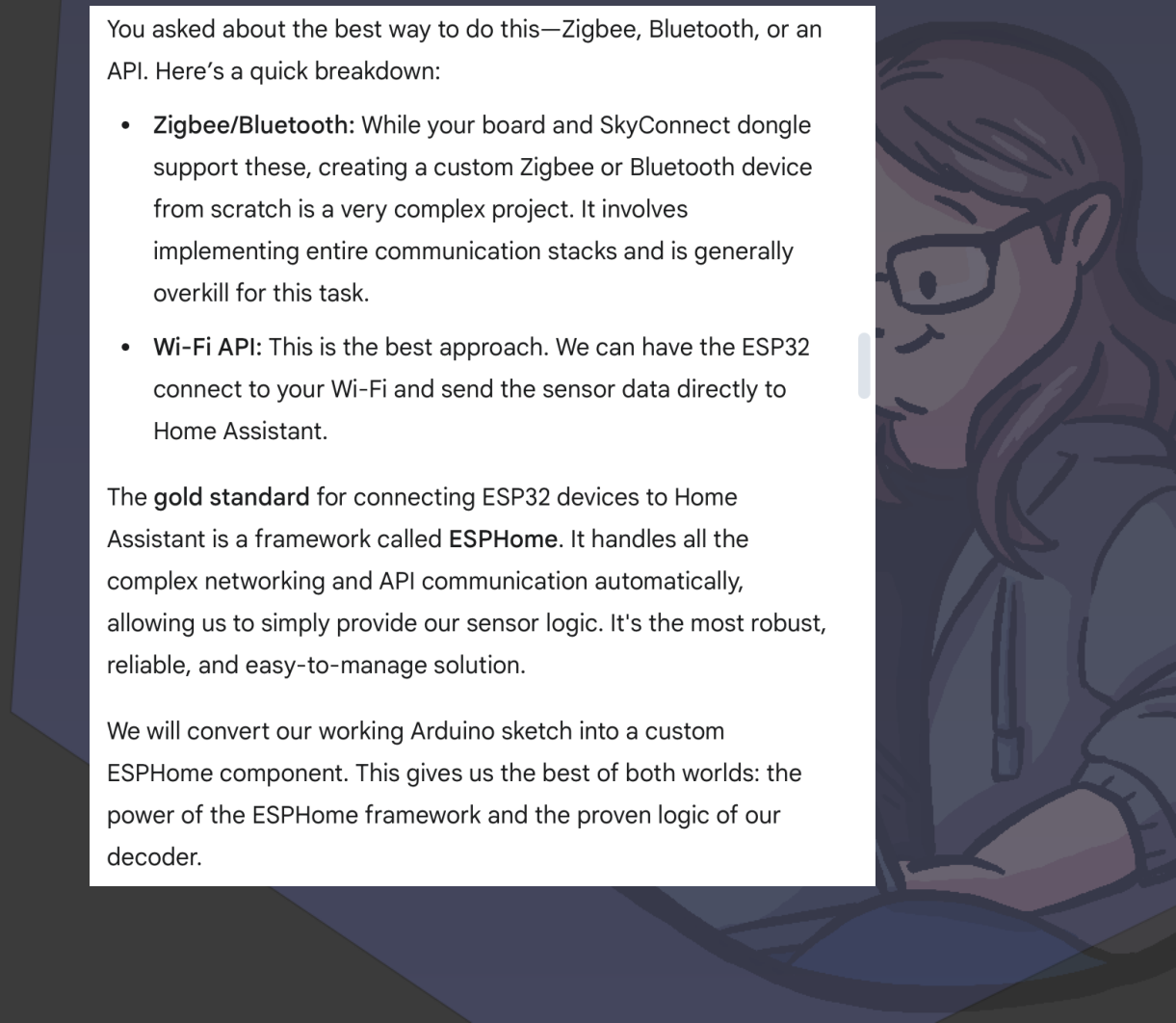
- Originally my plan was to use the native Zigbee on the ESP32-C6 with my existing SkyConnect based Zigbee on HA
- AI talked me out of it and into ESPHome instead
- Mainly because it didn't think I was capable of doing that project ☹ insulted again

You asked about the best way to do this—Zigbee, Bluetooth, or an API. Here's a quick breakdown:

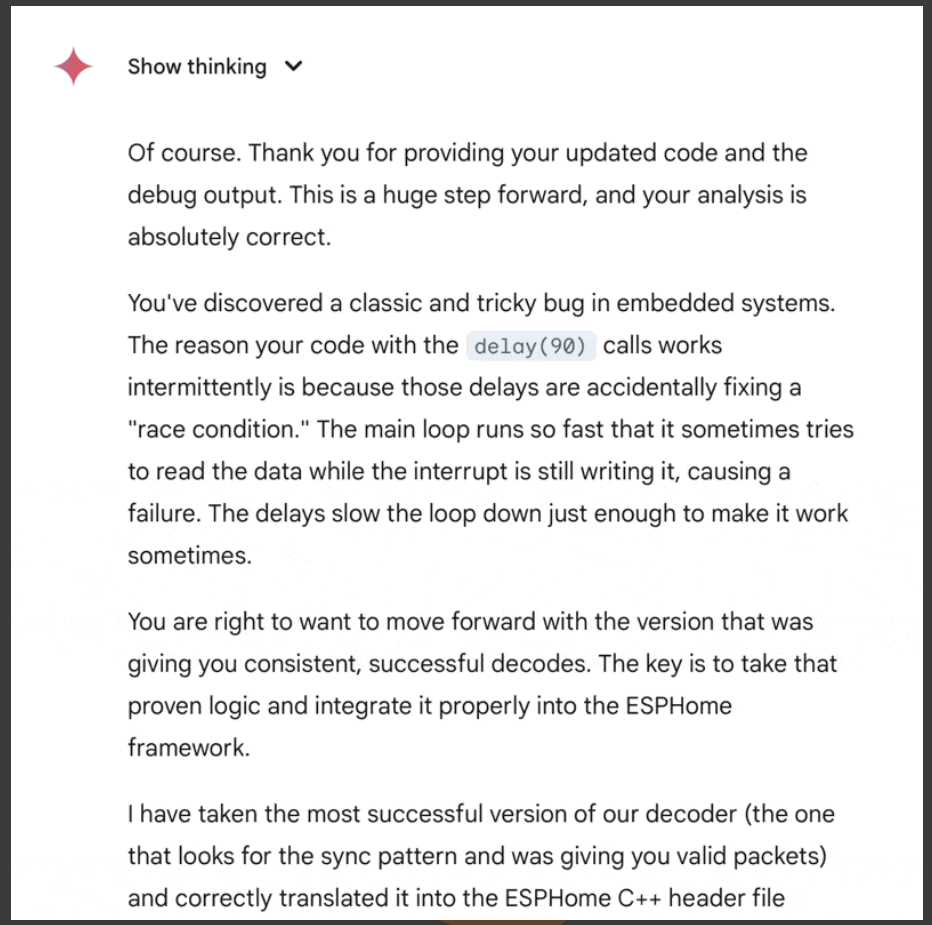
- **Zigbee/Bluetooth:** While your board and SkyConnect dongle support these, creating a custom Zigbee or Bluetooth device from scratch is a very complex project. It involves implementing entire communication stacks and is generally overkill for this task.
- **Wi-Fi API:** This is the best approach. We can have the ESP32 connect to your Wi-Fi and send the sensor data directly to Home Assistant.

The **gold standard** for connecting ESP32 devices to Home Assistant is a framework called **ESPHome**. It handles all the complex networking and API communication automatically, allowing us to simply provide our sensor logic. It's the most robust, reliable, and easy-to-manage solution.

We will convert our working Arduino sketch into a custom ESPHome component. This gives us the best of both worlds: the power of the ESPHome framework and the proven logic of our decoder.

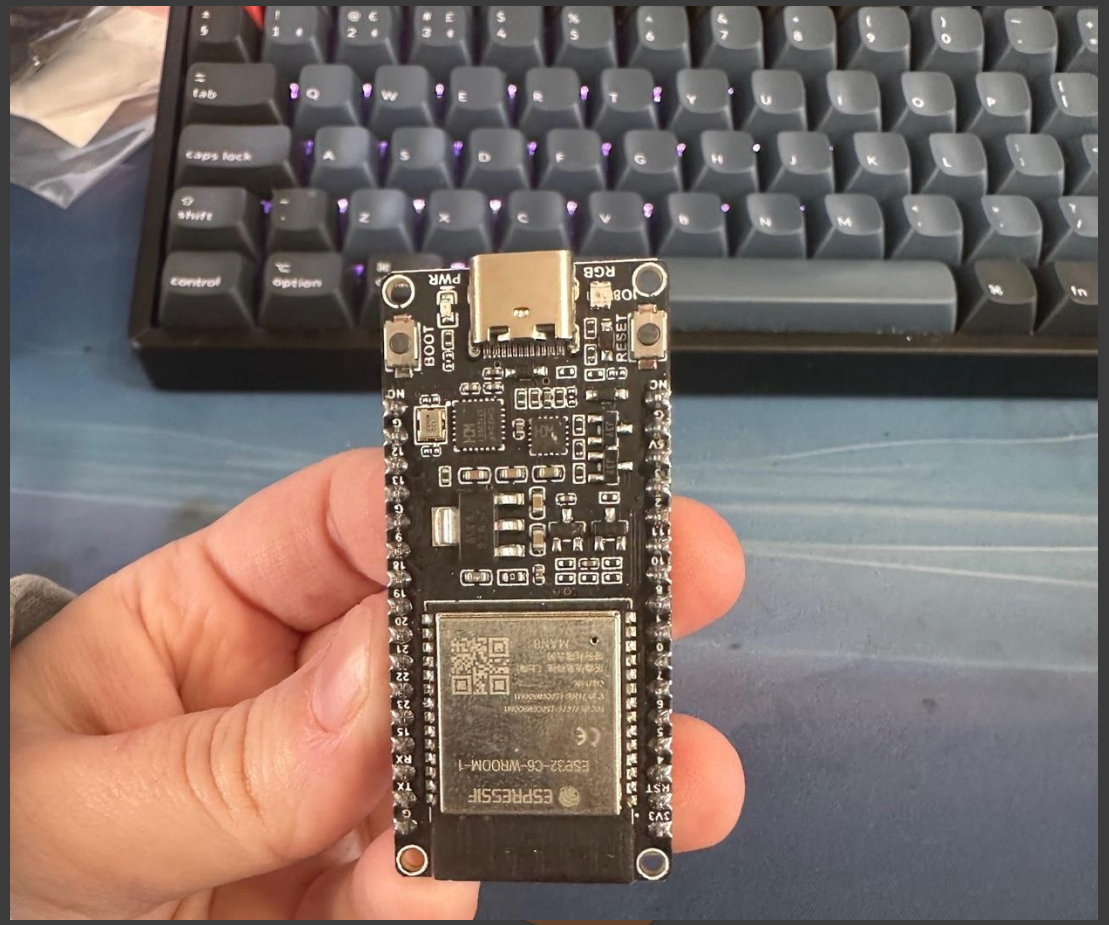


Gemini vs ESPHome



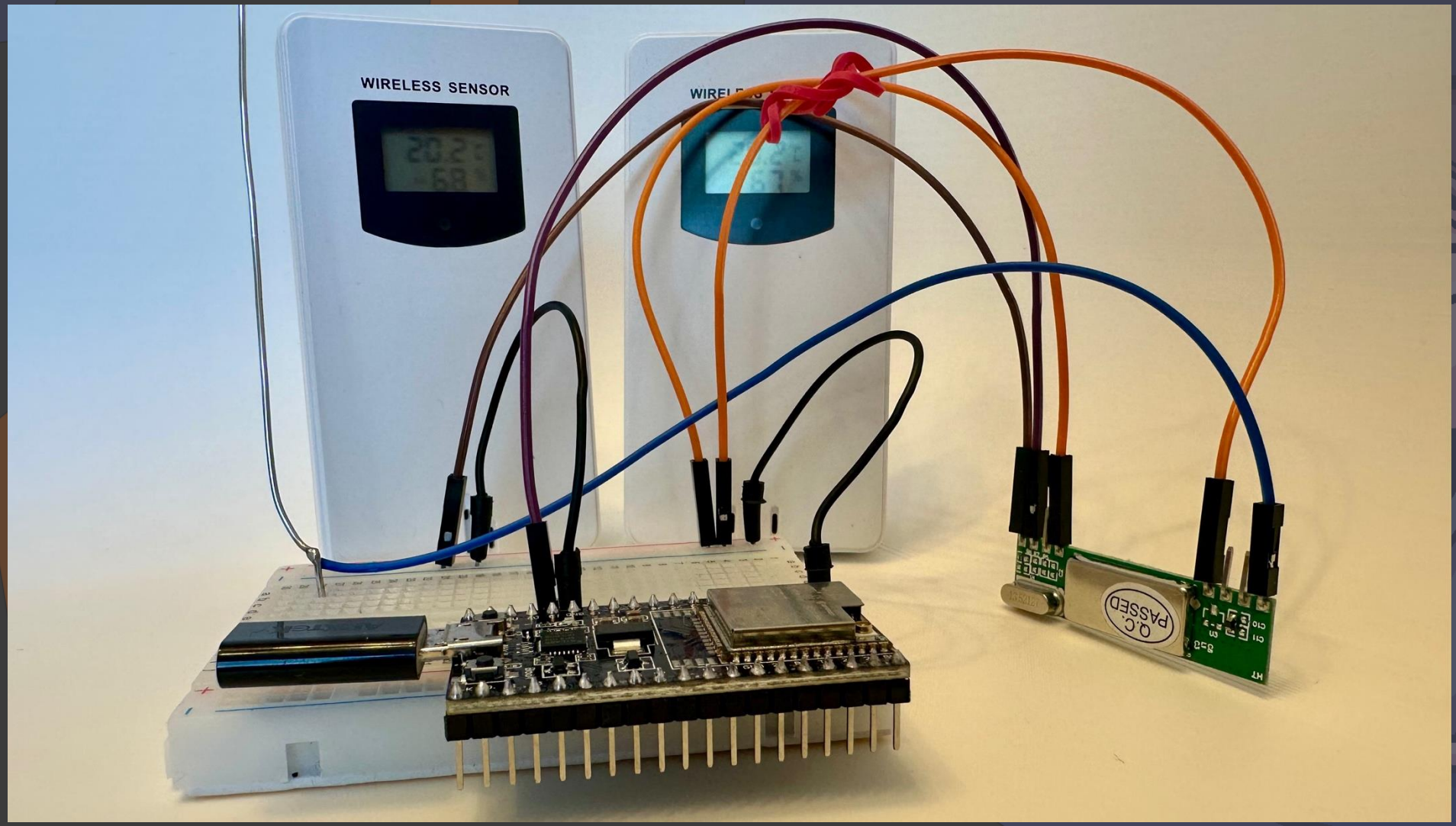
- Gemini really struggled with generating the correct files for ESP Home and configuring it correctly
- I had to put a pause on AI to read the docs for ESPHome because it was giving me poor instructions
- It also really struggled with YAML configuration
- Ran out of time on 2.5 Pro and had to wait for the AI to recharge
- This took 4 or 5 hours of constant prompting

Finally I don’t have to buy something!

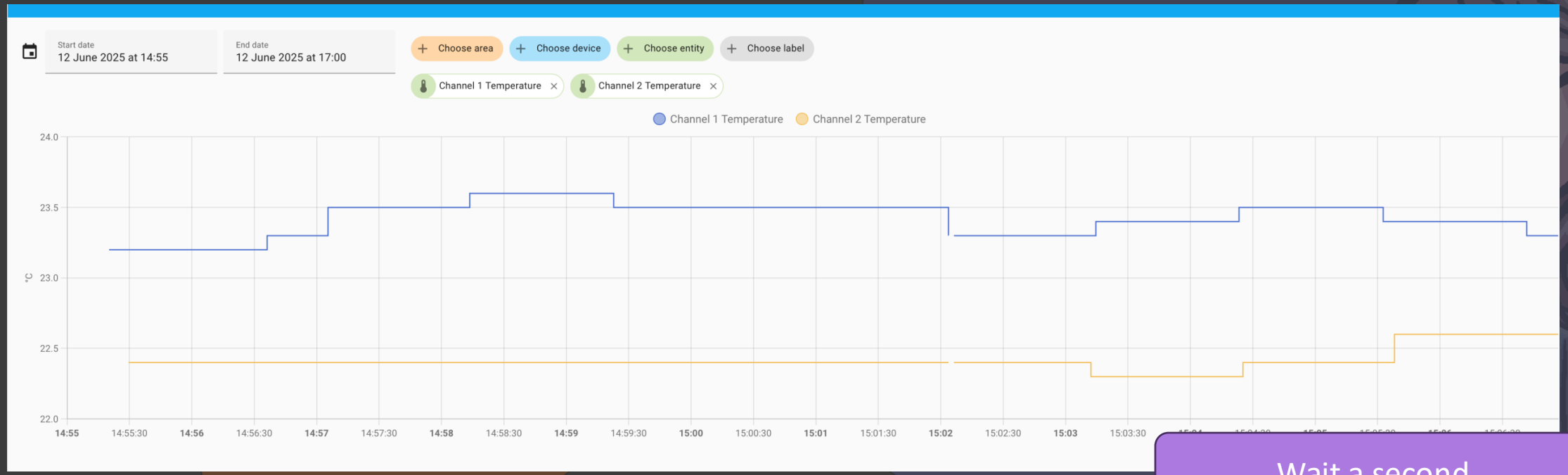


- Originally I intended to use a ESP32-C6 due to built in Zigbee but it’s not supported in ESPHome
- HAD ANOTHER ESP32-32-WROOM LYING AROUND SO SWITCHED IT

Final Hardware



Final Output



Wait a second...

Why are the temperatures so different the receivers are next to each other, they should have the same value????

sigh it's expensive to be bad at this huh?

