Advance-P4_ESPHome-Tutorial

1.course introduction

In this class, we will teach you how to use and operate the CrowPanel Advanced 5inch ESP32-P4 HMI AI Display through ESPHome.

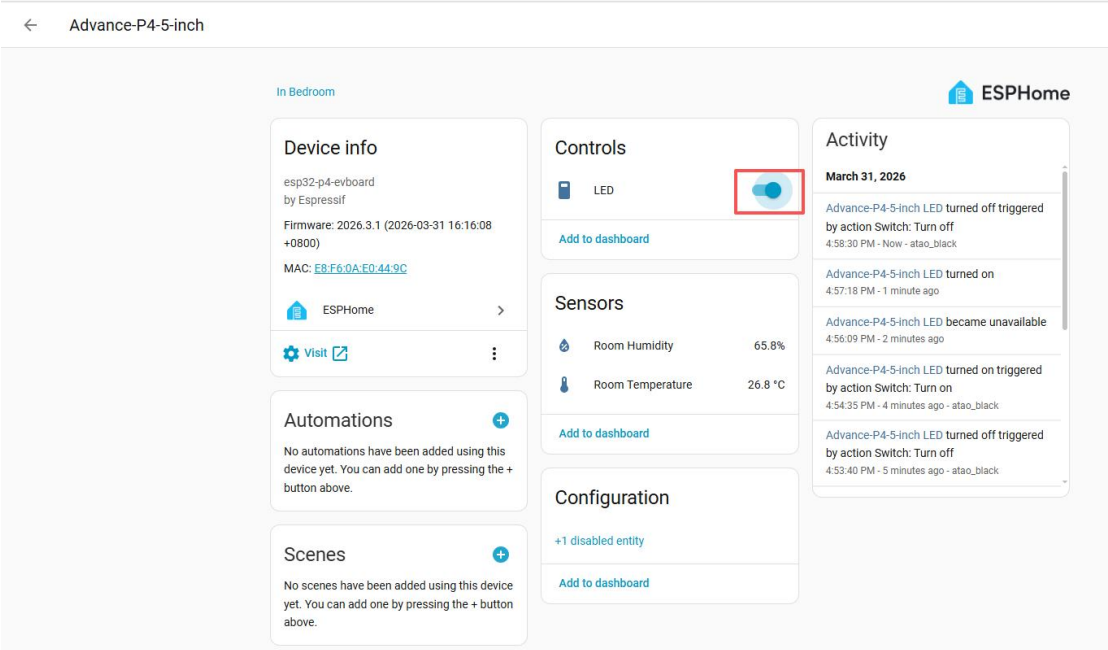
So let's move on to learn how to install the ESPHome environment, allowing you to edit code on it to achieve reading temperature and humidity data and remotely control the on and off of LEDs.

2.learning objectives

1. Get a basic understanding of the software required for writing ESPHome code.
2. Learn how to install the necessary software.
3. Learn how to create a new project.
4. Learn to write relevant code to achieve the function of collecting temperature and humidity data, and be able to view the temperature and humidity data remotely on the ESPHome platform.
5. Learn to write relevant code to achieve the function of turning on and off an LED, and be able to control the LED remotely on the ESPHome platform.

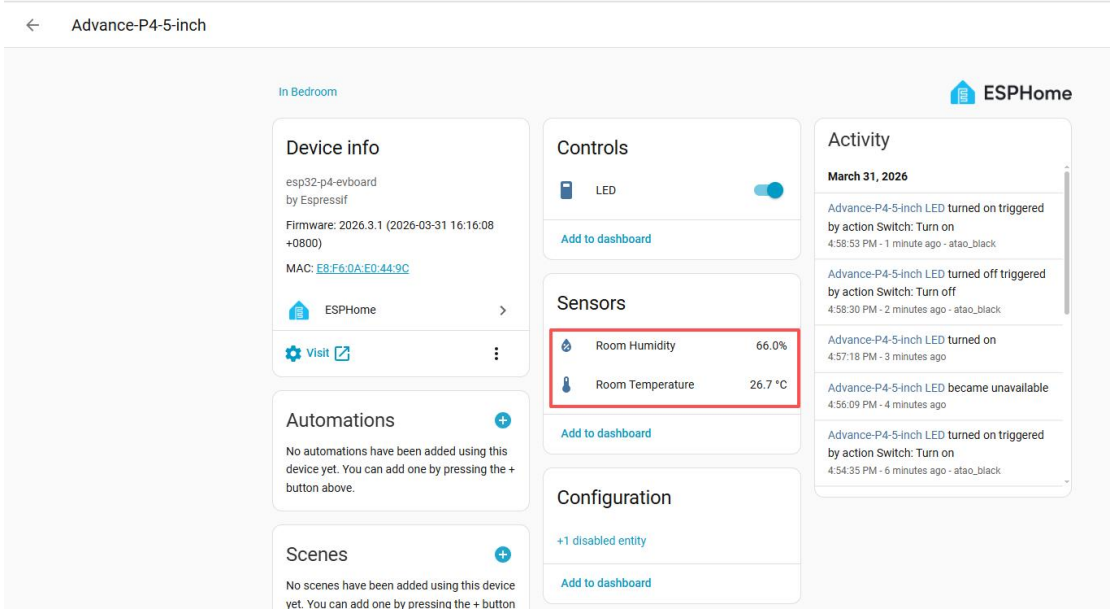
3.Project operation effect illustration

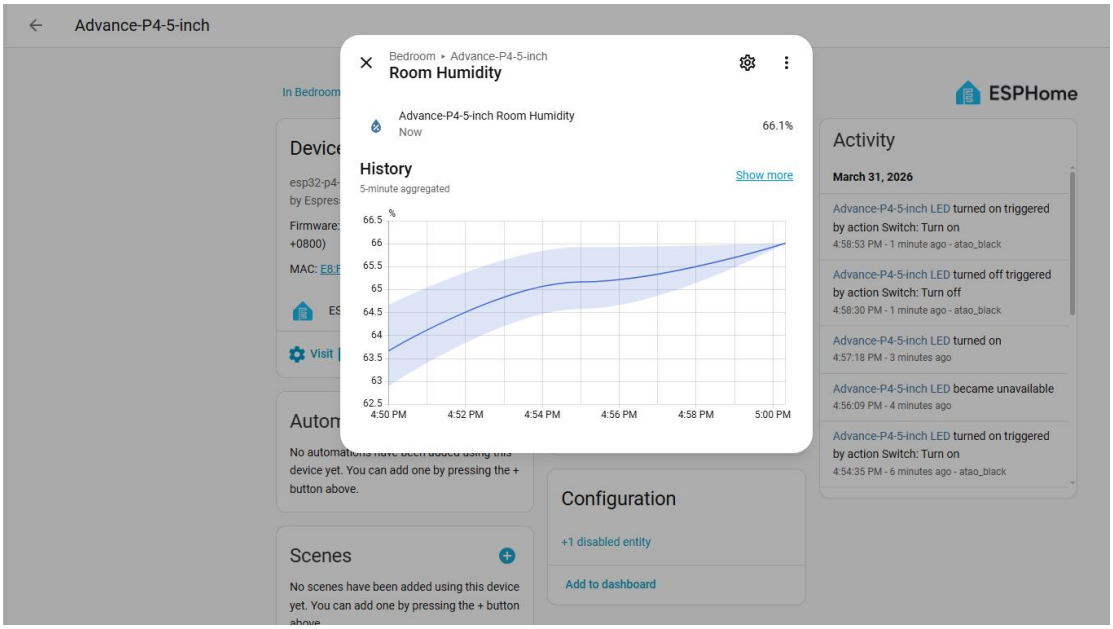
When you click the switch of the LED, you will be able to see that the indicator light on the screen is on, and the feedback from the LED will also be obvious.

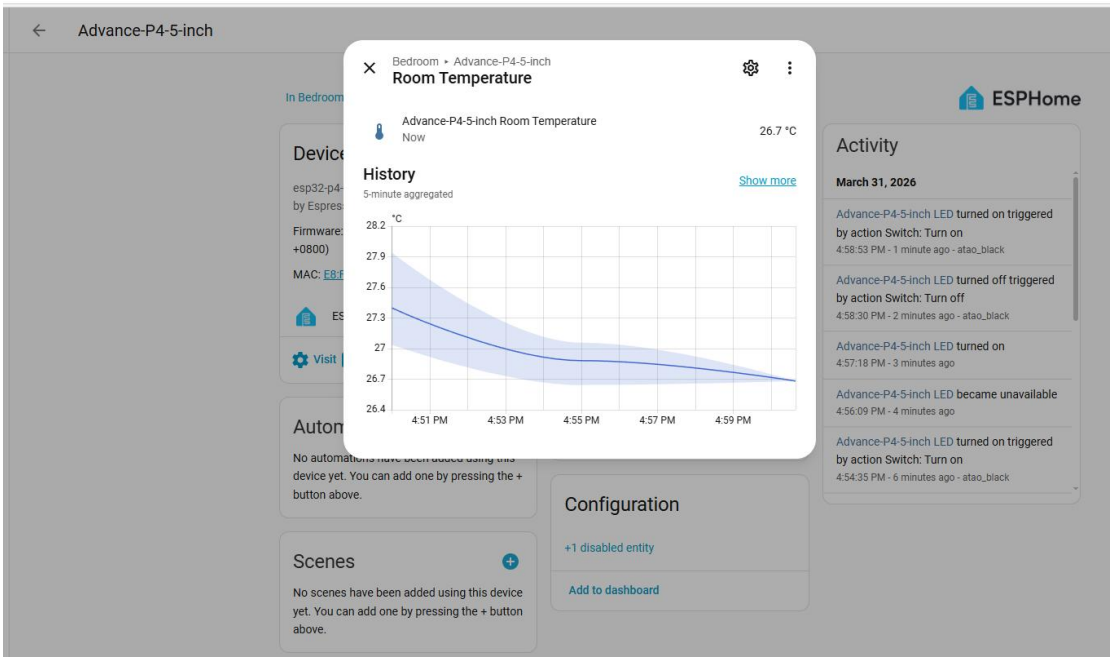


Next, when you click the switch of the LED, you will be able to see that the display light on the screen turns off, and the feedback from the LED is also quite obvious.

And you can also see the historical data of temperature and humidity collection from here.







4.Introduce the software

Here's a clear, accessible overview for international customers (no overly technical jargon):

Home Assistant (often called "HA" for short) is a free, open-source smart home management platform—think of it as the central "control brain" for all your smart devices. It works on computers, small servers, or even dedicated smart home hubs, and lets you manage every smart product you own (lights, thermostats, sensors, your rotary screen, etc.) in one unified interface—regardless of the brand or technology the device uses. For your rotary screen, HA acts as the "command center": it can receive inputs from the screen (like knob rotations or touch

taps), control other devices based on those inputs (e.g., turn up the lights when you twist the knob), and send data (like room temperature or music volume) to the screen for display.

ESPHome is a free, open-source tool built to configure smart hardware (specifically devices powered by ESP32/ESP8266 chips—your rotary screen uses an ESP32). The key benefit? You don't need to write complex code to make the screen work. Instead, you use simple, plain-text configurations (like filling in a template) to define what the rotary screen does: e.g., "show 'Hello World' on the display," "adjust the thermostat when the knob turns," or "wake the screen when touched." ESPHome takes these configurations, turns them into instructions the ESP32 can understand, and "loads" them onto your rotary screen—while also automatically syncing the screen's status (what it's displaying, how the knob is being used) with Home Assistant.

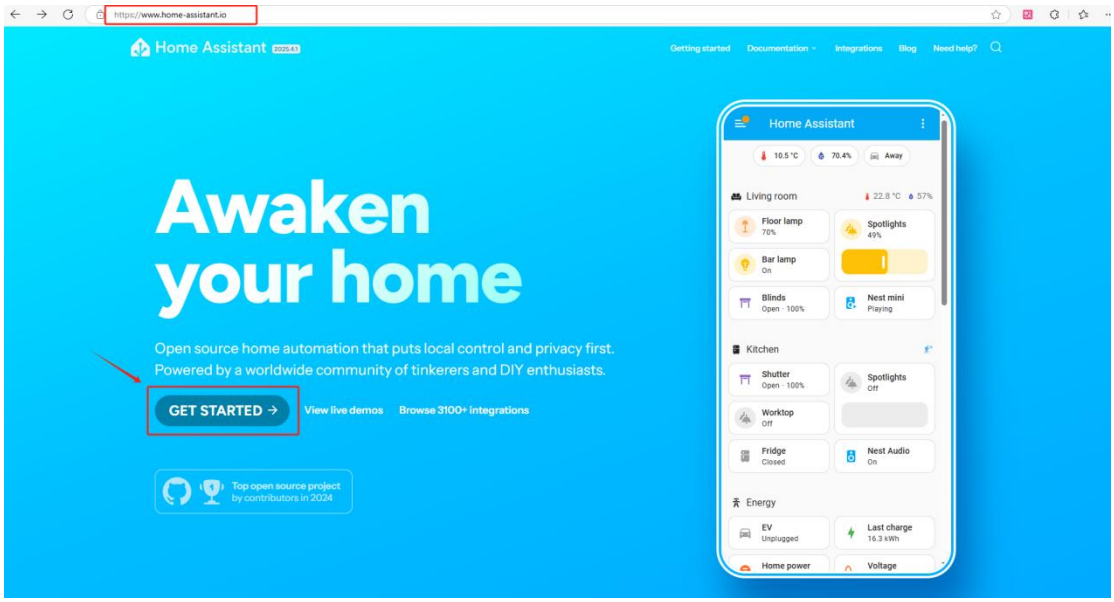
Together, Home Assistant and ESPHome turn your rotary screen into a flexible, customizable smart device: ESPHome gives the screen its "basic skills" (display, knob/touch response), and HA connects those skills to your entire smart home—so the screen can both control your devices and show you what's happening with them, all without requiring technical expertise.

5.How to install software

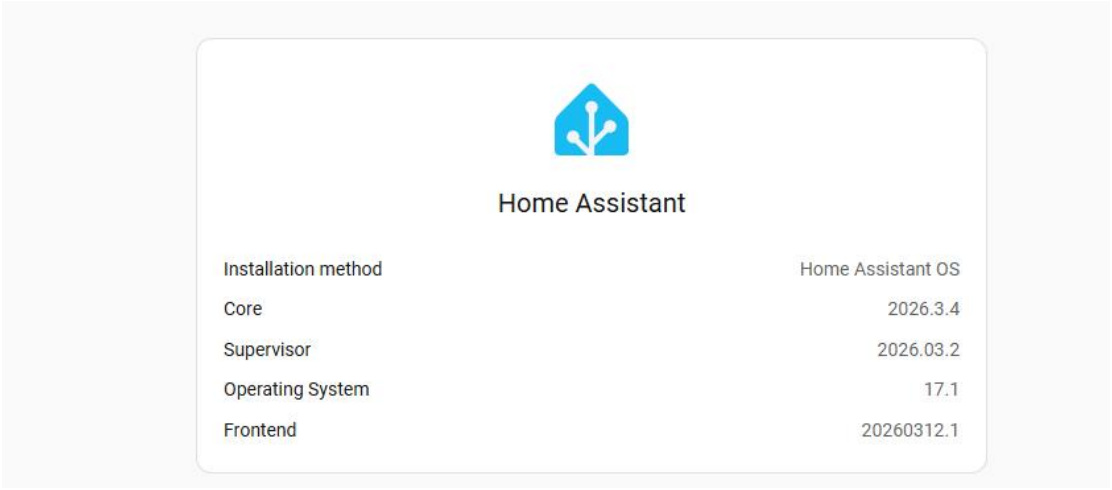
First, we need to prepare:
a Raspberry Pi 5;
a 64GB SD card;
a CrowPanel Advanced 5inch ESP32-P4 HMI AI Display;
A humidity and temperature sensor;
An LED light;

Download HomeAssitant

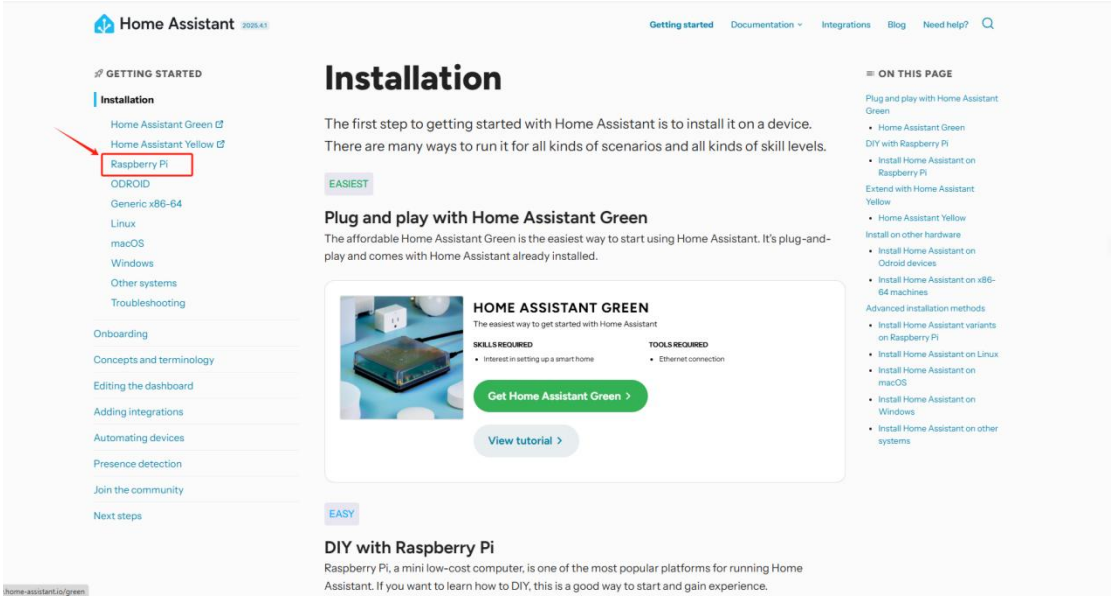
Open the official website of HomeAssitant：https://www.home-assistant.io/



The version we are using here is 2026/3/4. This is because of the version requirements of the ESP32-P4-related libraries needed for our subsequent code.



Select a Raspberry Pi and install the Home Assistant system according to this installation guide.(Install according to the official installation guide.)



After the installation is complete, insert the SD card with the Home Assistant system into the Raspberry Pi and connect the Ethernet cable.

Note: Make sure that the Wi-Fi network your CrowPanel Advanced 5inch ESP32-P4 HMI AI Display will connect to is on the same local area network (LAN) as the Raspberry Pi.

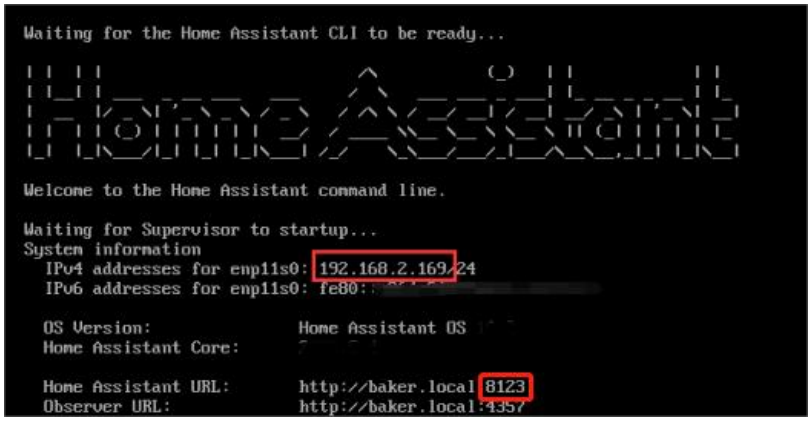
The following devices need to be on the same LAN:

① Your computer

② CrowPanel Advanced 5inch ESP32-P4 HMI AI Display

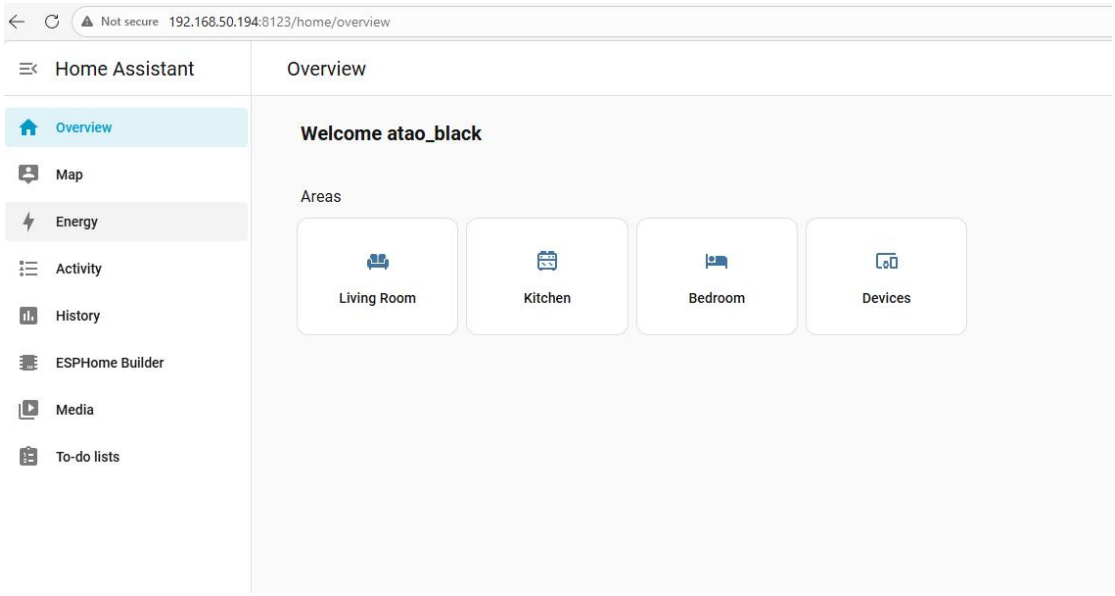
③ Raspberry Pi with the Home Assistant system

Note: The Raspberry Pi must be connected to a screen to view information. Once everything is ready, power on the Raspberry Pi and wait for it to load. The screen will display the following image.
(You need to connect a display screen to the Raspberry Pi in order to see these information.)

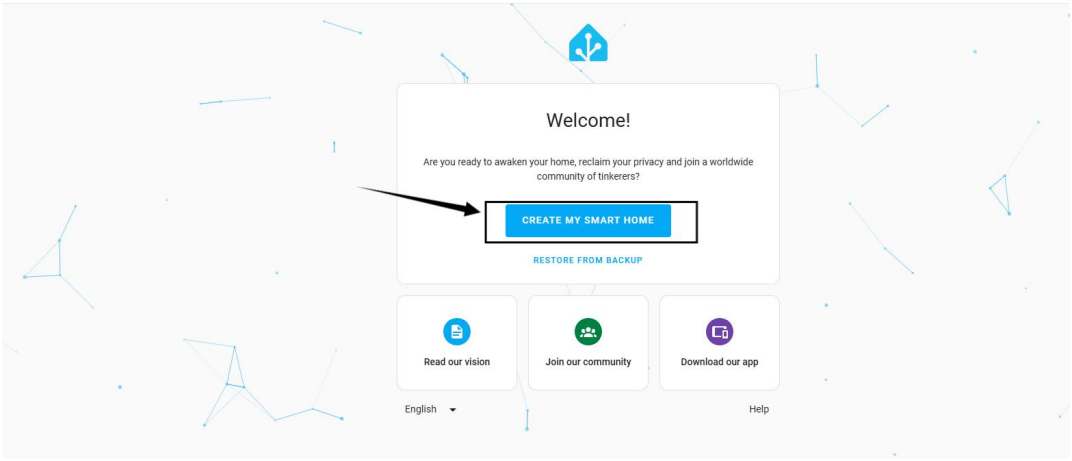


Remember the IP address and the Home Assistant URL.

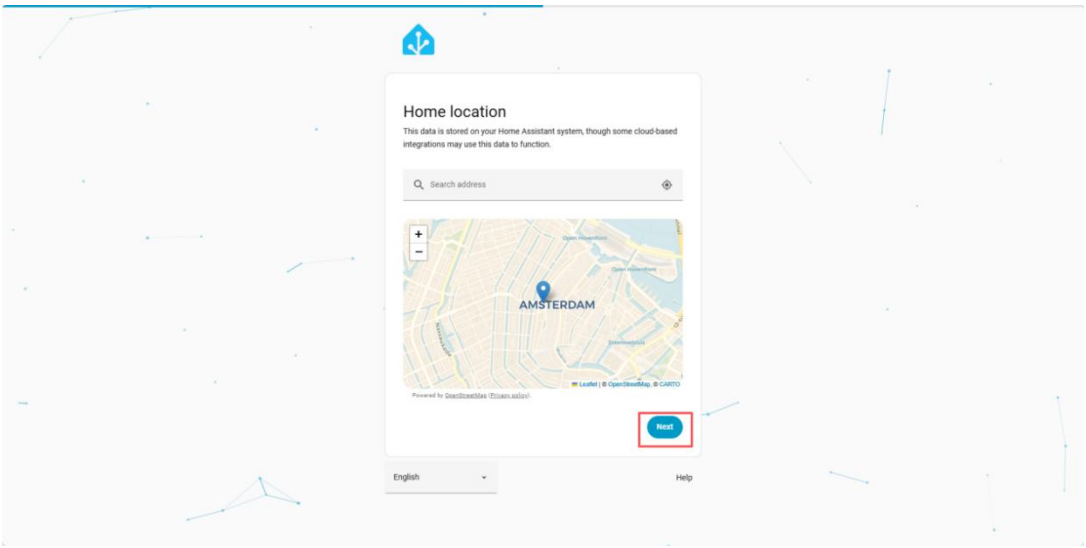
In my case, the IP address is: 192.168.50.194
So I enter **192.168.50.194:8123** in the browser.



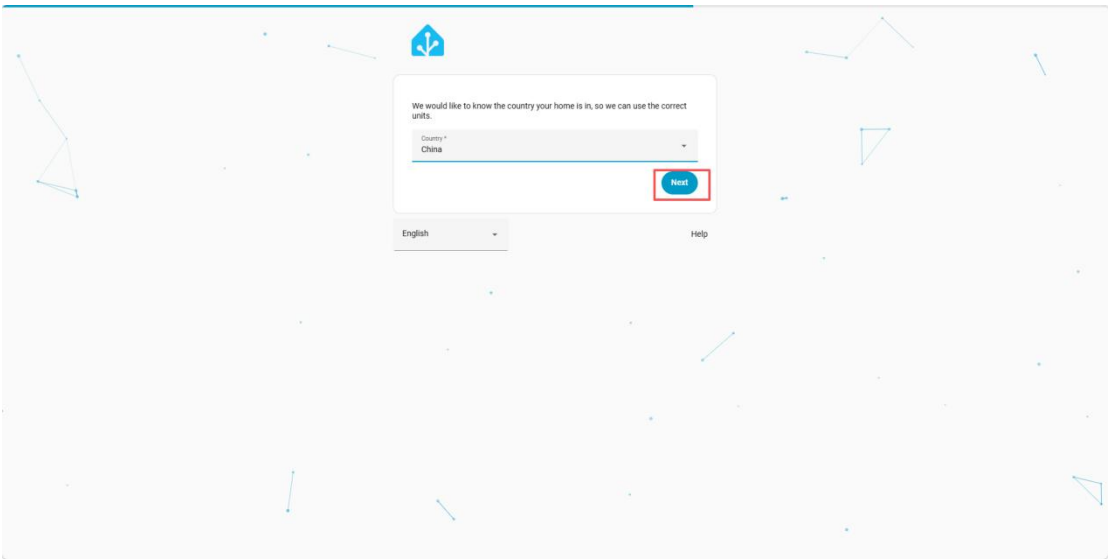
Next, start configuring the Home Assistant system.



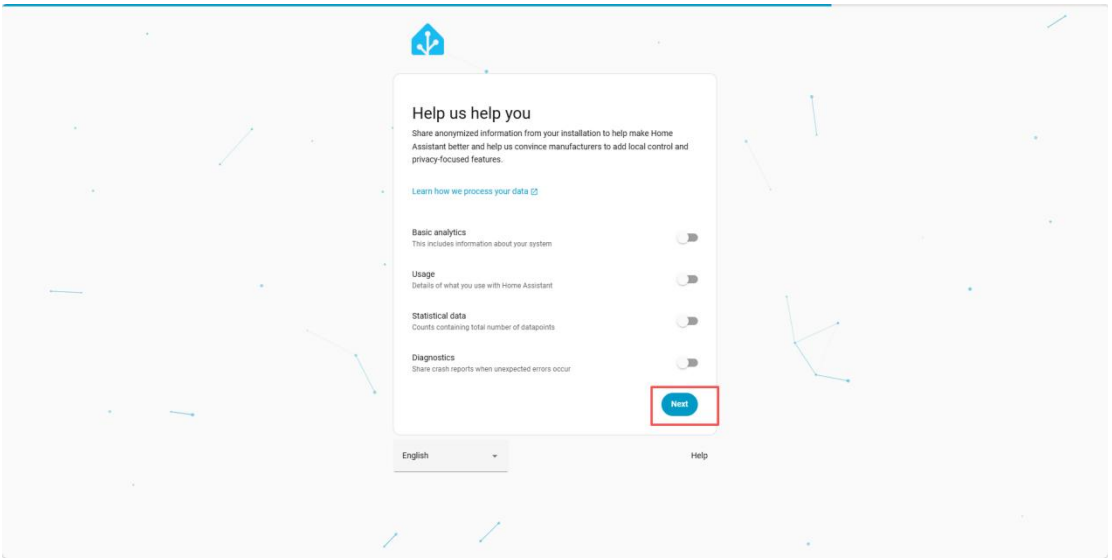
You can either manually set your location or allow it to be detected automatically.



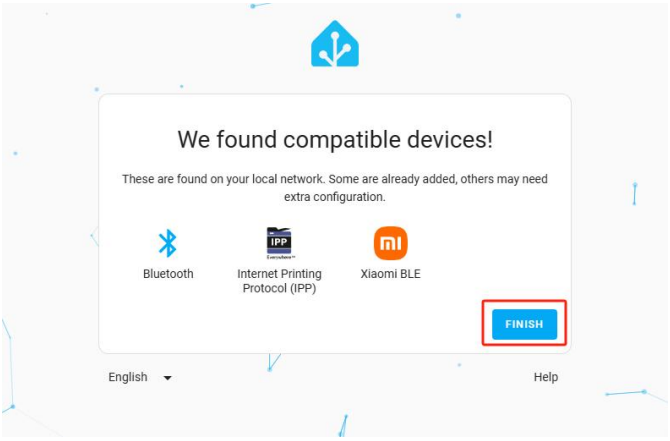
Set the country where you are located.



Keep it as default here.

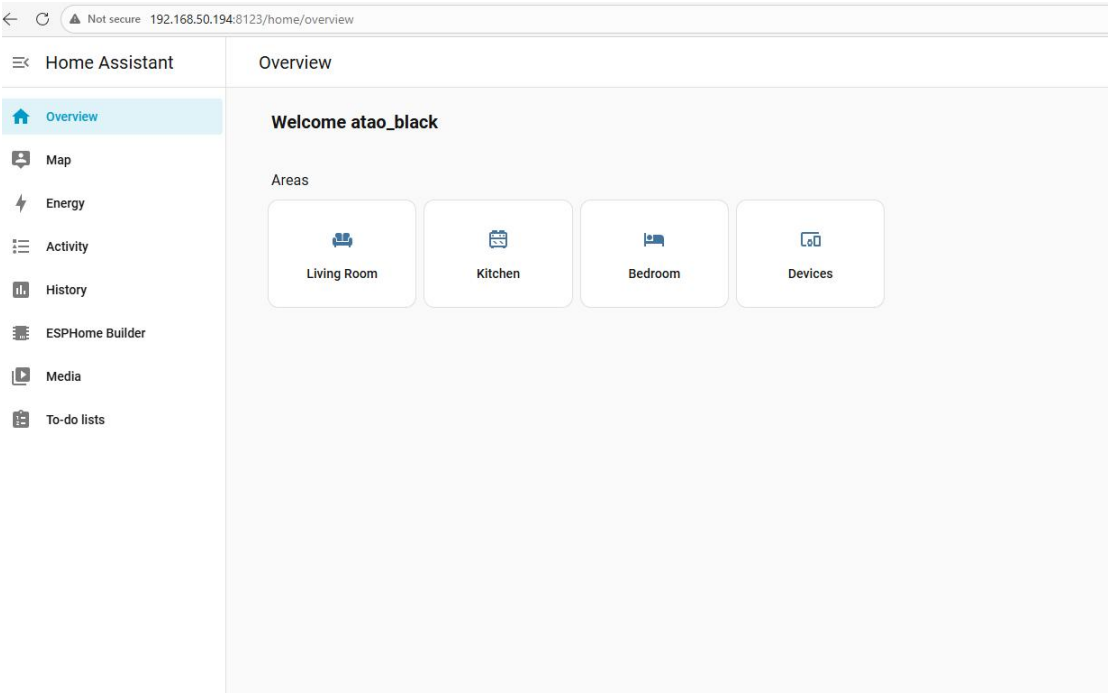


You can either add smart devices now or click **Finish** to add them later.
Here, we will add the devices later.

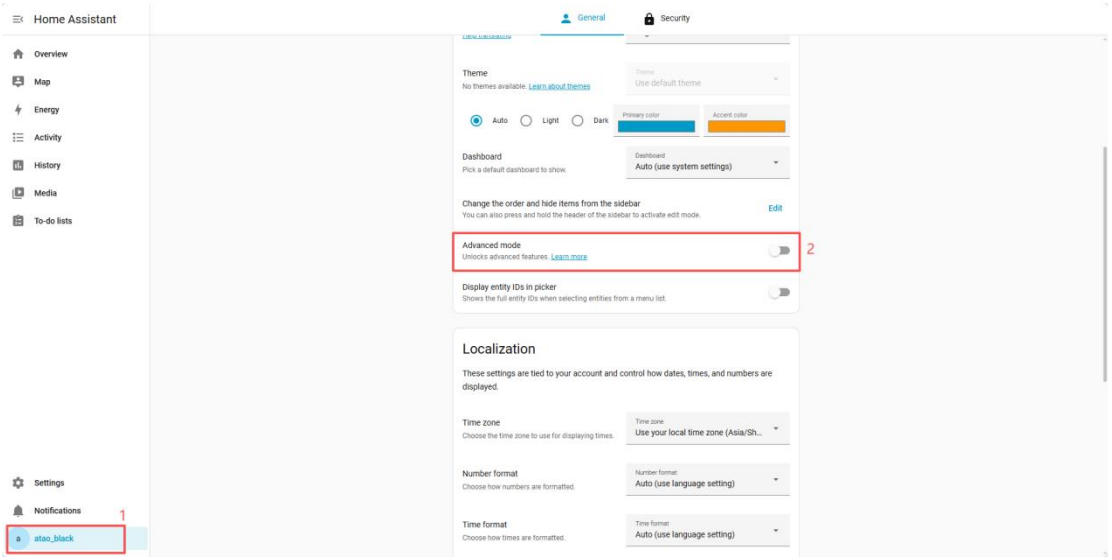


Add ESPHome

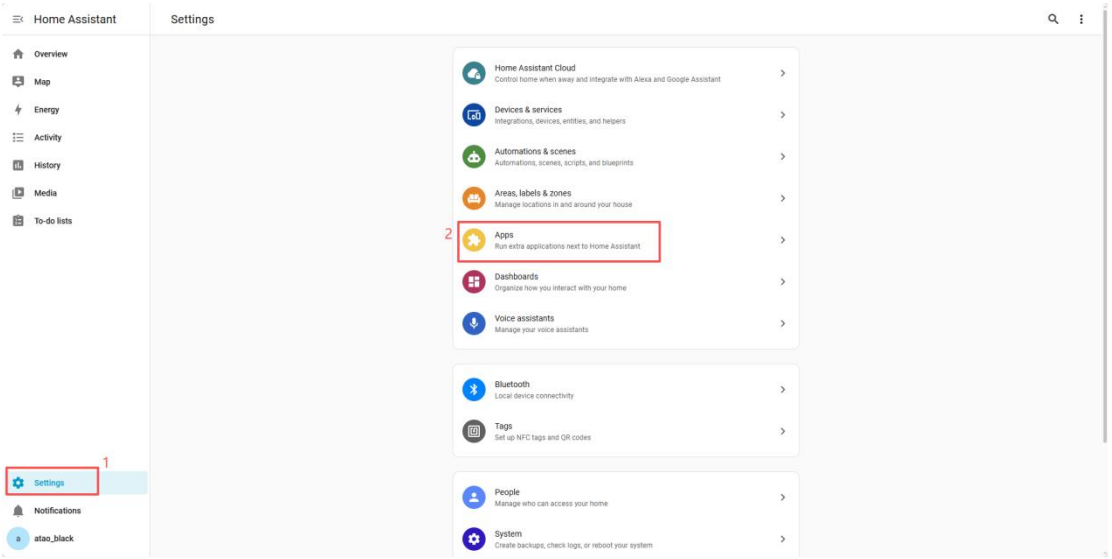
This will bring you to the main interface of Home Assistant.



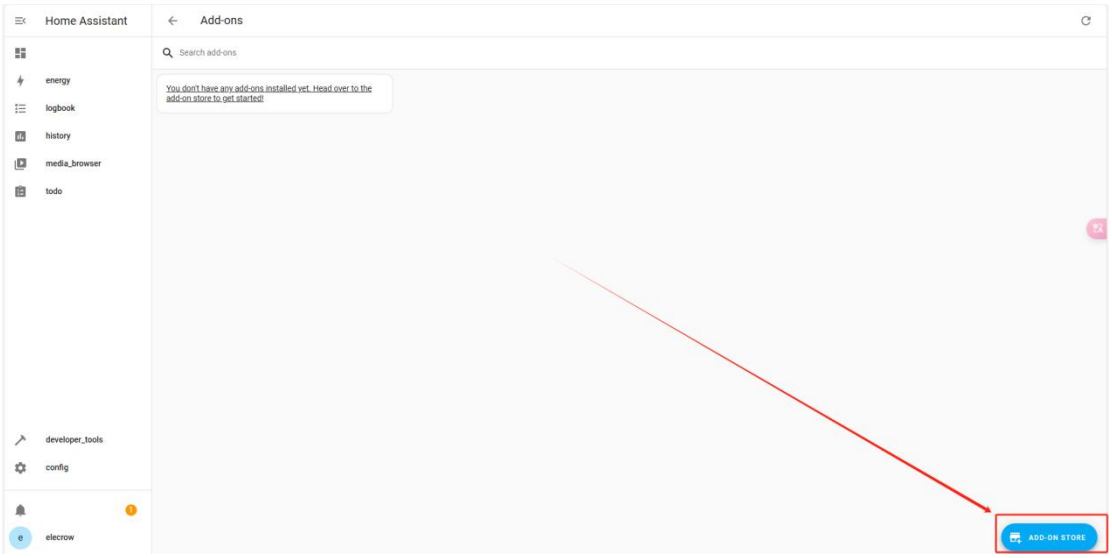
Click on the username and enable "Advanced Mode."

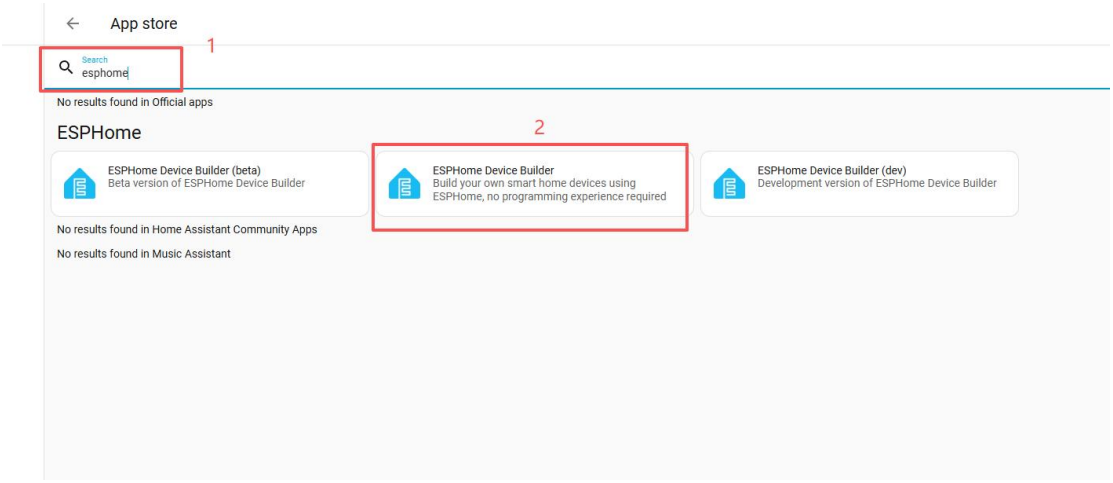


Click on "Settings" and then "Apps."

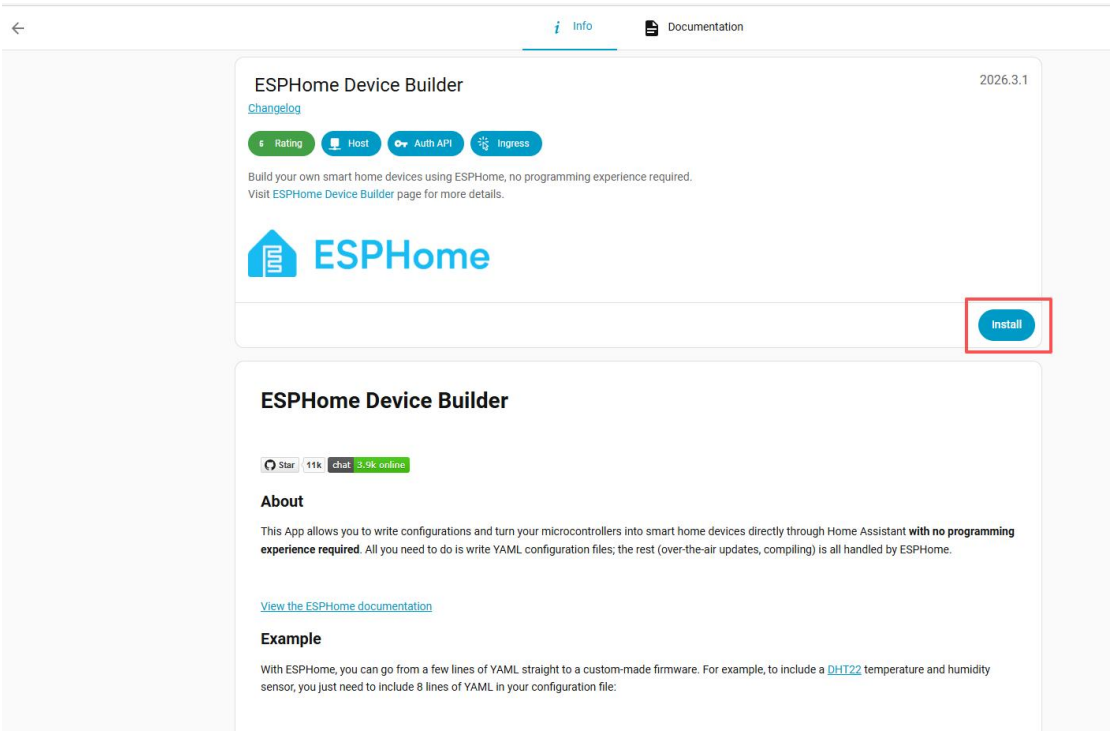


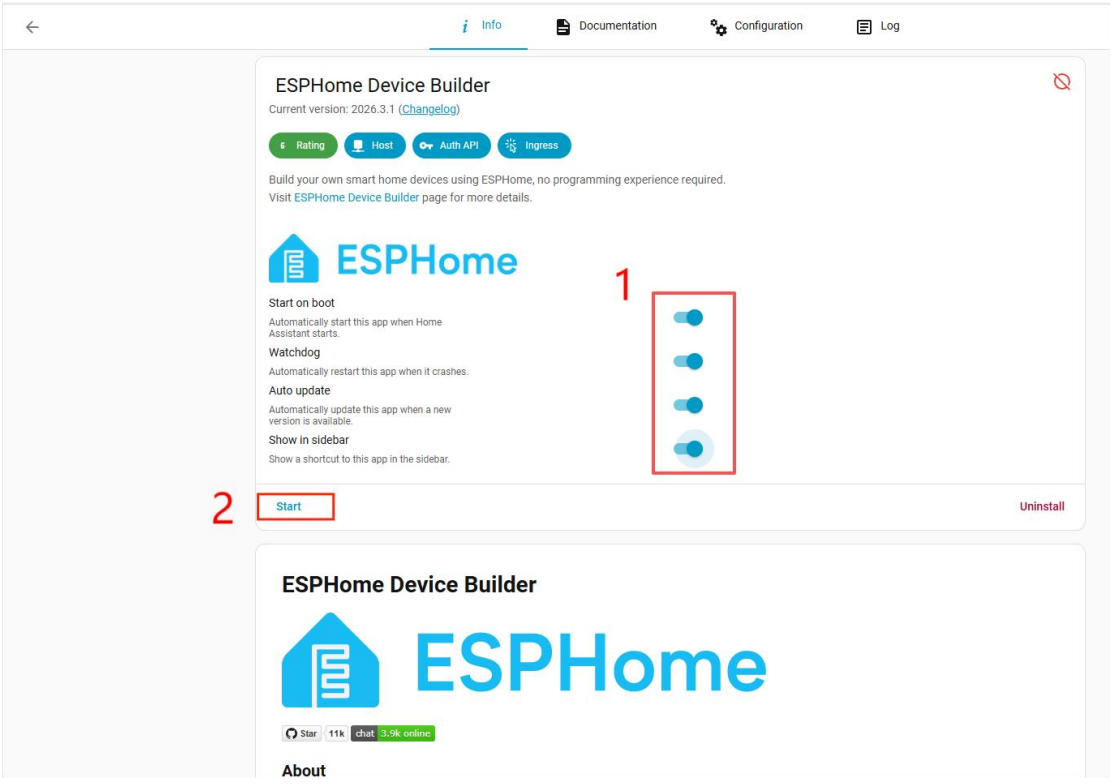
Click on "Apps" and type **ESPHome** to install it.





The installation is in progress, as shown in the image.



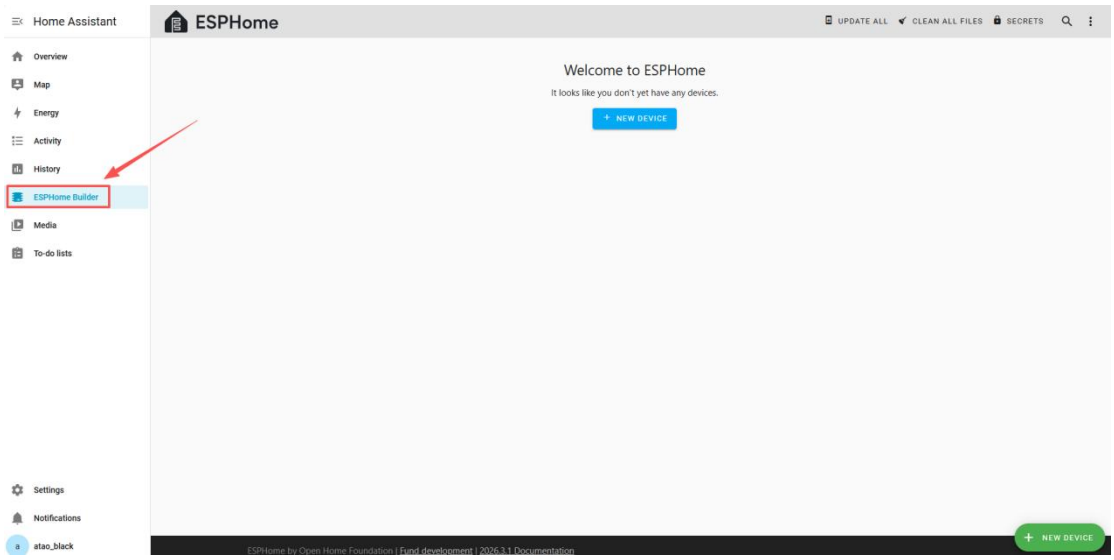


So we have successfully installed ESPHome. From now on, we can use ESPHome to edit the code and achieve the functions we want.

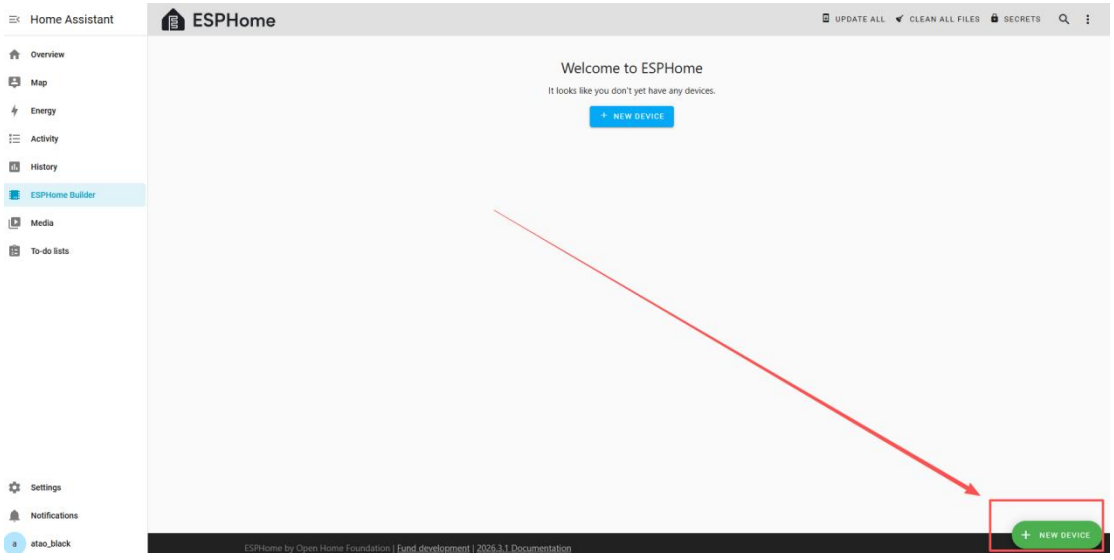
6. New Project

Once the installation is complete, we can start adding devices.

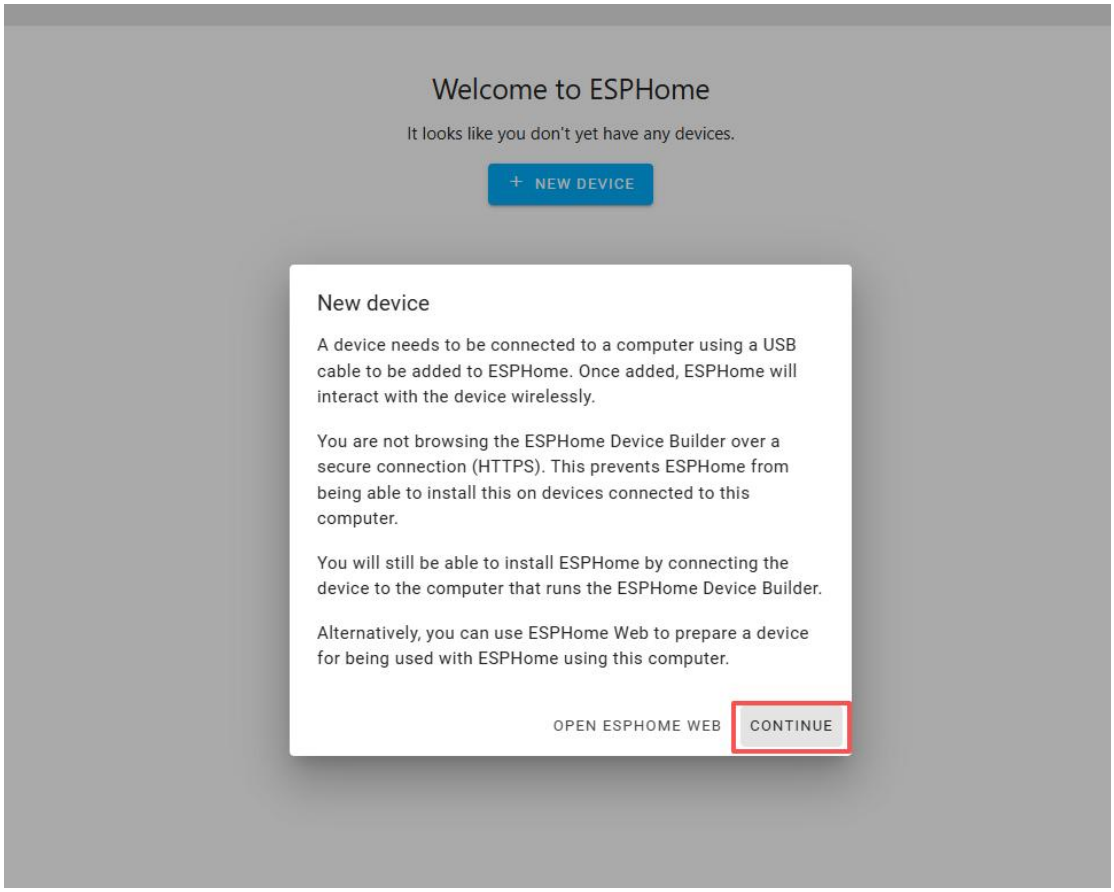
Come to ESPHome Builder



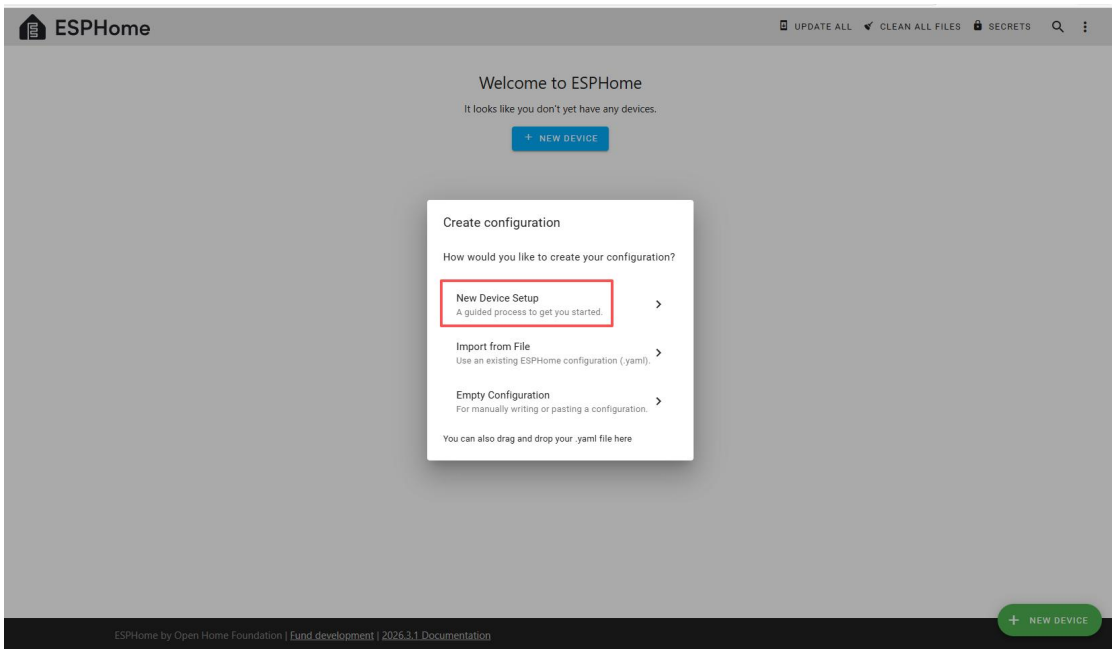
Click to add a new device



Click "Continue"

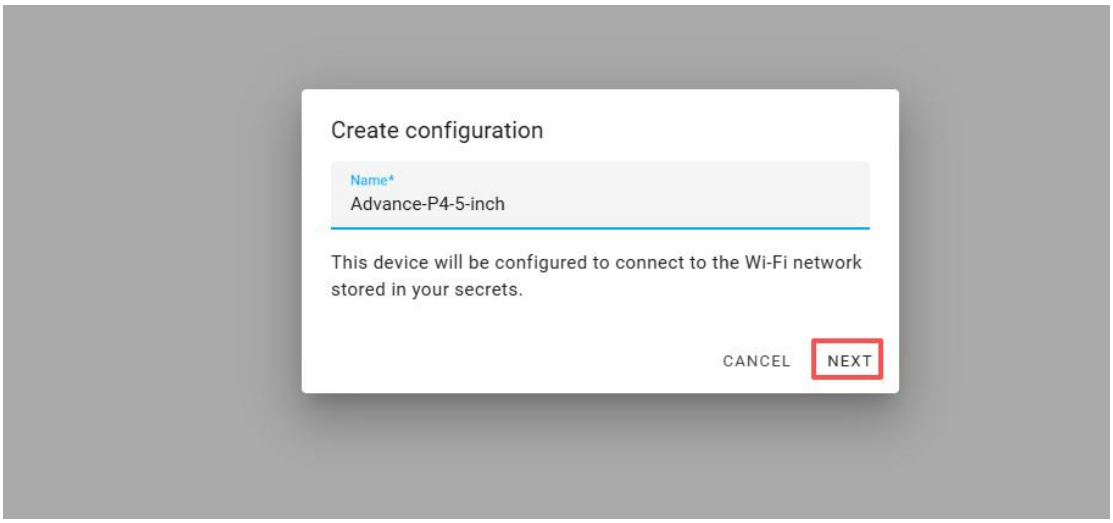


Add new equipment



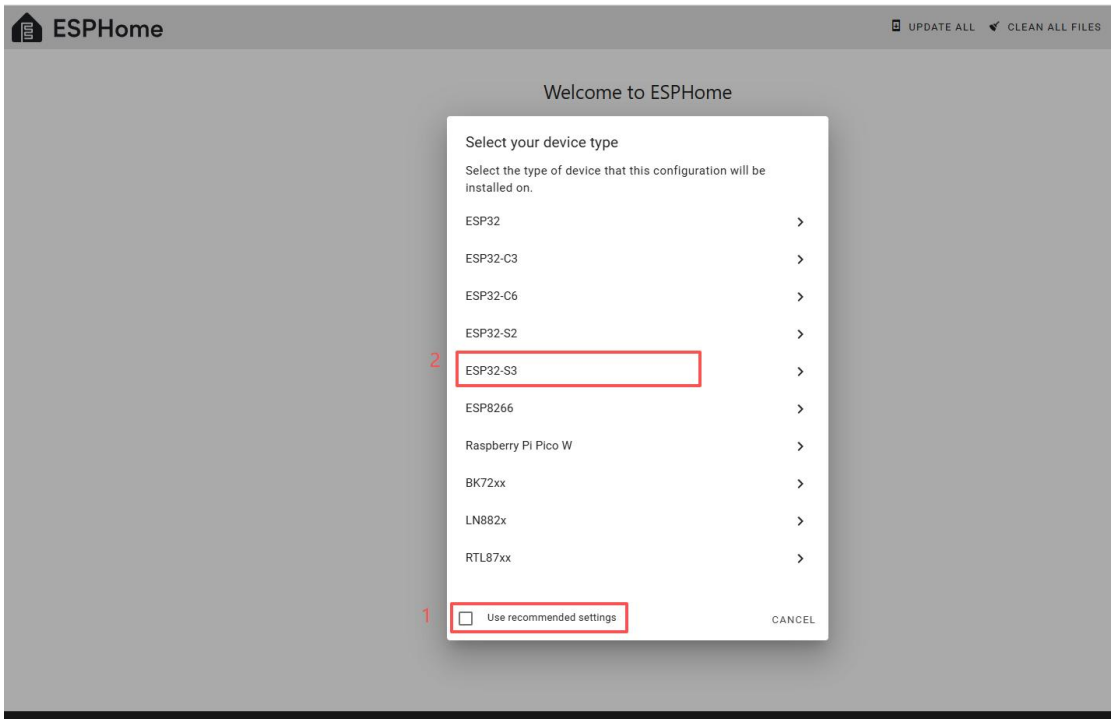
Set a name for this project, as well as the name and password of the Wi-Fi that you will have the device connect to.

(This Wi-Fi must be in the same local network as your computer host, and make sure your Wi-Fi is on the 2.4GHz frequency band, as the ESP32-C6 Wi-Fi module used by ESP32-P4 operates on 2.4GHz.)

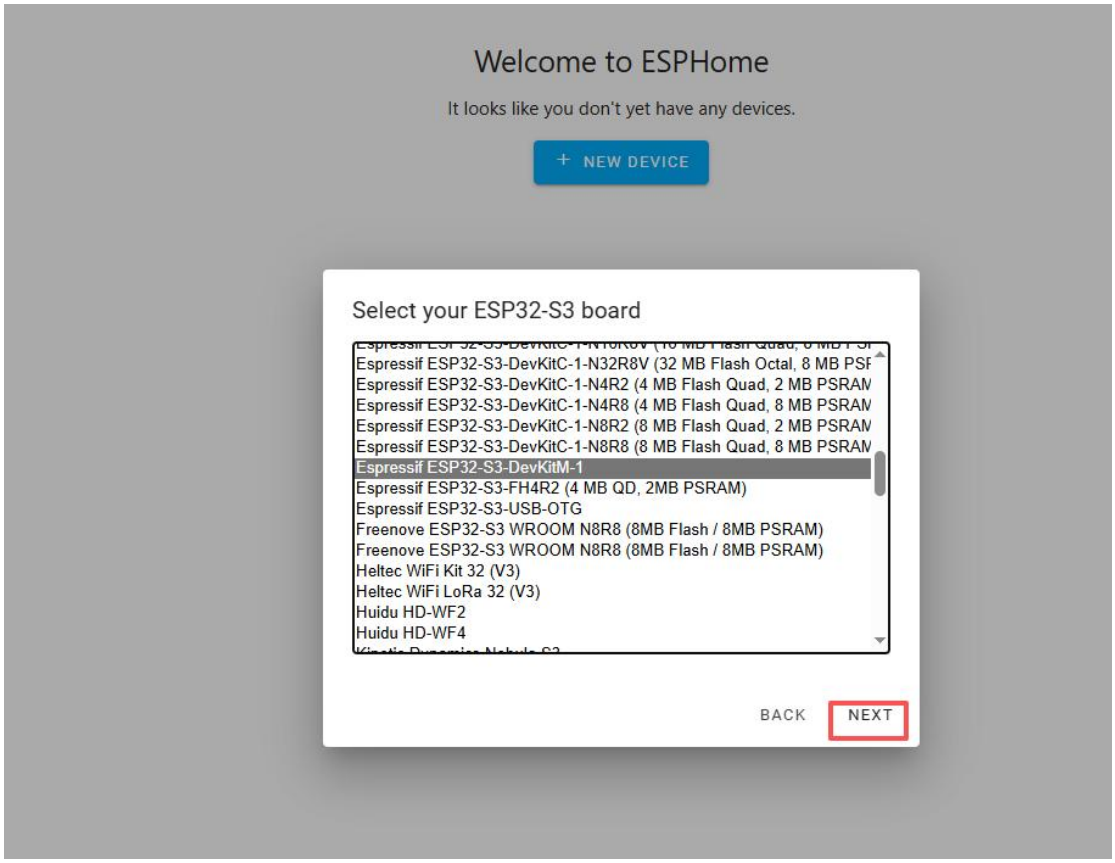


Here, do not check "Use recommended settings."

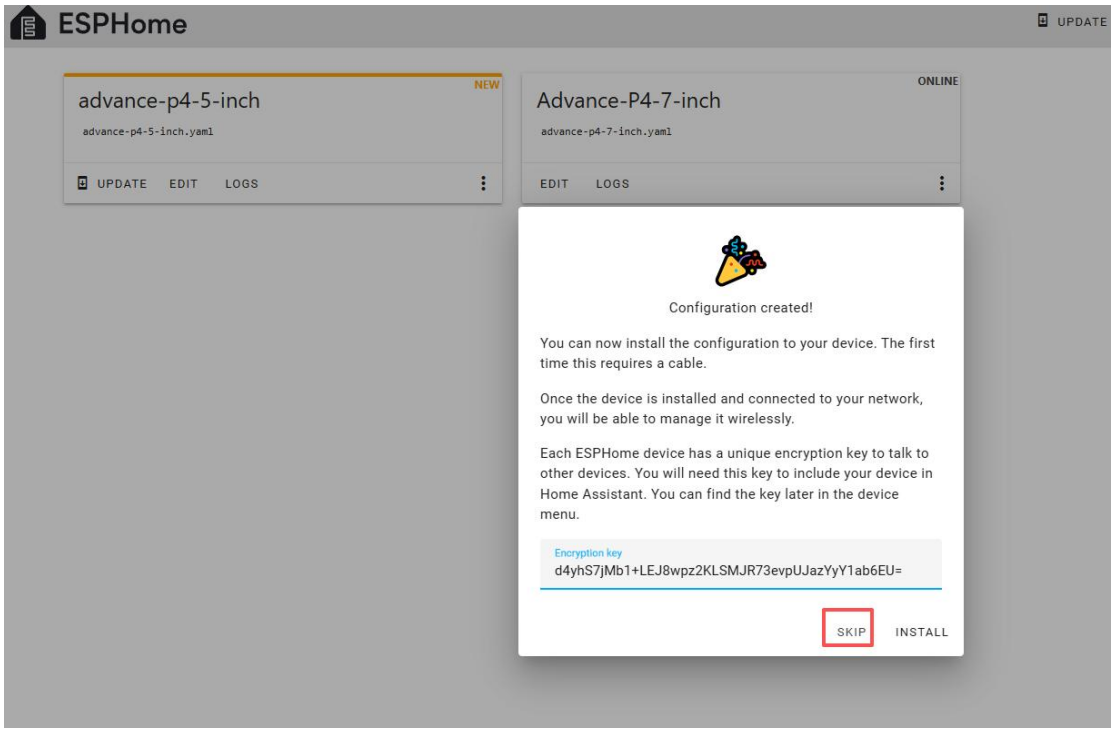
Here, we should have chosen the main control chip ESP32-P4 for the CrowPanel Advanced 5inch ESP32-P4 HMI AI Display , but currently the official has not provided the interface yet. So for now, we'll just randomly pick one, and later we can manually modify it in the code. That is, here I choose ESP32-S3.



Next, choose any option (since we will replace it in the code later).

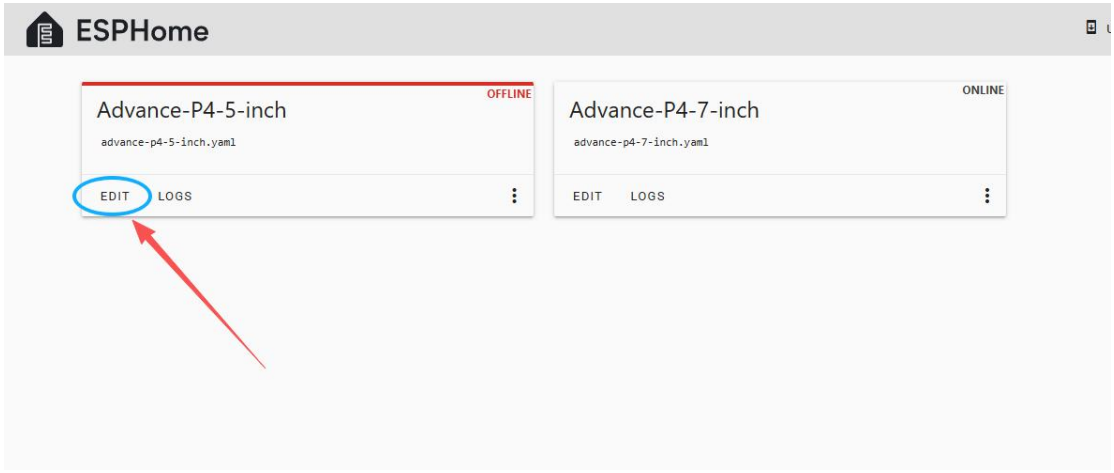


Here, click "SKIP".



7. Configuration Code

Then, return to the main interface, find the Advance_P4_5_inch you just created, click "EDIT", and enter the code editor.



This code is automatically generated based on the previous steps. Next, we will make replacements in it, which will help optimize the code for more efficient operation.

advance-p4-5-inch.yaml

```yaml
esphome:
  name: advance-p4-5-inch
  friendly_name: Advance-P4-5-inch

esp32:
  board: esp32-s3-devkitm-1
  framework:
    type: esp-idf

# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "d4yhS7jMb1+LEJ8wpz2KLSMJR73evpUJazYyY1ab6EU="

ota:
  - platform: esphome
    password: "7e67f5f46de710476467d7daed31b484"

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Advance-P4-5-Inch"
    password: "1zSU4rOJQyzw"

captive_portal:
```

You can click here to download the code in this course, which will help you achieve the related functions.

https://github.com/Elecrow-RD/-CrowPanel-Advanced-5inch-ESP32-P4-HMI-AI-Display-800x480-IPS-Touch-Screen/tree/master/example/V1.0/ESPHome/Code

Next, you can replace the relevant content in esphome and ESP32 as needed.

advance-p4-5-inch.yaml

```yaml
esphome:
  name: advance-p4-5-inch
  friendly_name: Advance-P4-5-inch

esp32:
  board: esp32-s3-devkitm-1
  framework:
    type: esp-idf

# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "d4yhS7jMb1+LEJ8wpz2KLSMJR73evpUJazYyY1ab6EU="

ota:
  - platform: esphome
    password: "7e67f5f46de710476467d7daed31b484"

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Advance-P4-5-Inch"
    password: "1zSU4rOJQyzw"

captive_portal:
```

Here, I have modified the esp32 section to be compatible with the esp32-p4 main controller, and added PSRAM. The operation of this code requires certain conditions.

advance-p4-5-inch.yaml

```yaml
esphome:
  name: advance-p4-5-inch
  friendly_name: Advance-P4-5-inch

esp32:
  variant: esp32p4
  engineering_sample: true
  cpu_frequency: 360MHZ
  flash_size: 16MB
  framework:
    type: esp-idf
    advanced:
      execute_from_psram: true
      enable_idf_experimental_features: true

psram:
  speed: 200MHz

# Enable logging
logger:
  level: debug
  logs:
    lvgl: info
  hardware_uart: UART0

# Enable Home Assistant API
api:
  encryption:
    key: "d4yhS7jMb1+LEJ8wpz2KLSMJR73evpUJazYyY1ab6EU="

ota:
  - platform: esphome
    password: "7e67f5f46de710476467d7daed31b484"
```

Since our ESP32-P4 chip does not have WiFi communication capabilities, its WiFi function is handled by the ESP32-C6 WiFi chip. Therefore, here we need to add the information of the ESP32-C6 and its pins on our board, so that the WiFi can connect.

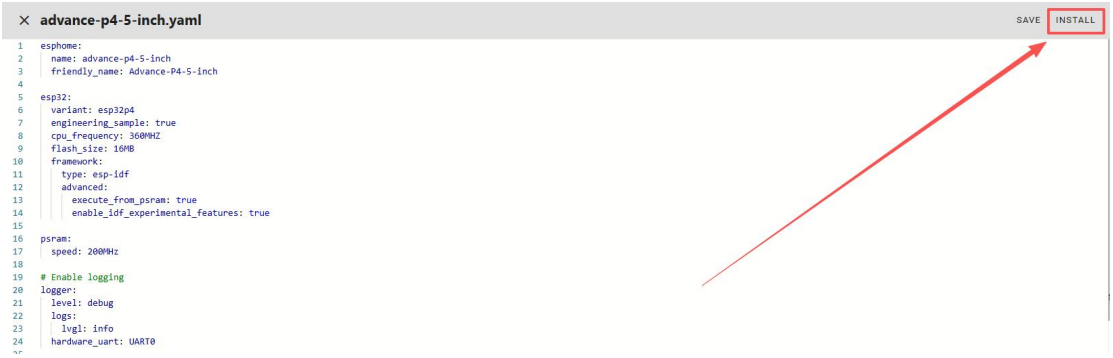
Remember to replace your own Wi-Fi name and password.

advance-p4-5-inch.yaml

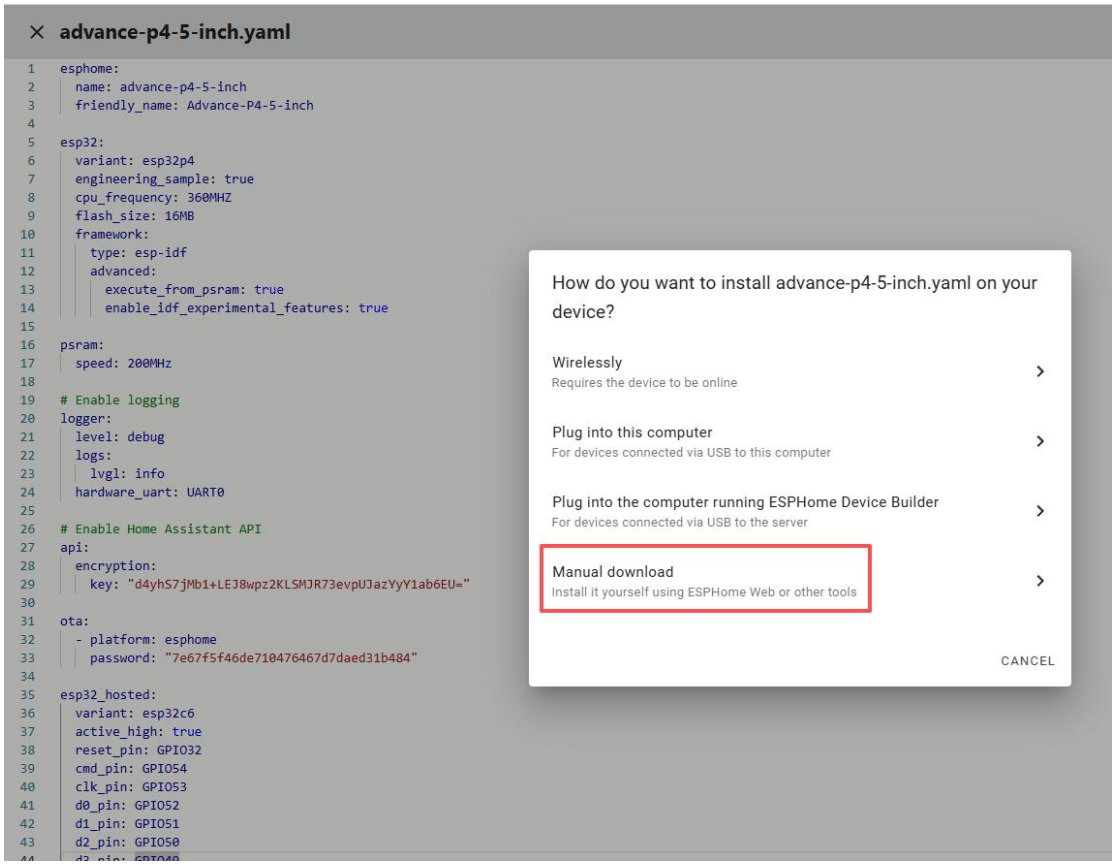
```yaml

# Enable Home Assistant API
api:
  encryption:
    key: "d4yhS7jMb1+LEJ8wpz2KLSMJR73evpUJazYyY1ab6EU="

ota:
  - platform: esphome
    password: "7e67f5f46de710476467d7daed31b484"

esp32_hosted:
  variant: esp32c6
  active_high: true
  reset_pin: GPIO32
  cmd_pin: GPIO54
  clk_pin: GPIO53
  d0_pin: GPIO52
  d1_pin: GPIO51
  d2_pin: GPIO50
  d3_pin: GPIO49

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Advance-P4-5-Inch"
    password: "1zSU4rOJQyzw"

captive_portal:
```

8. First upload of code

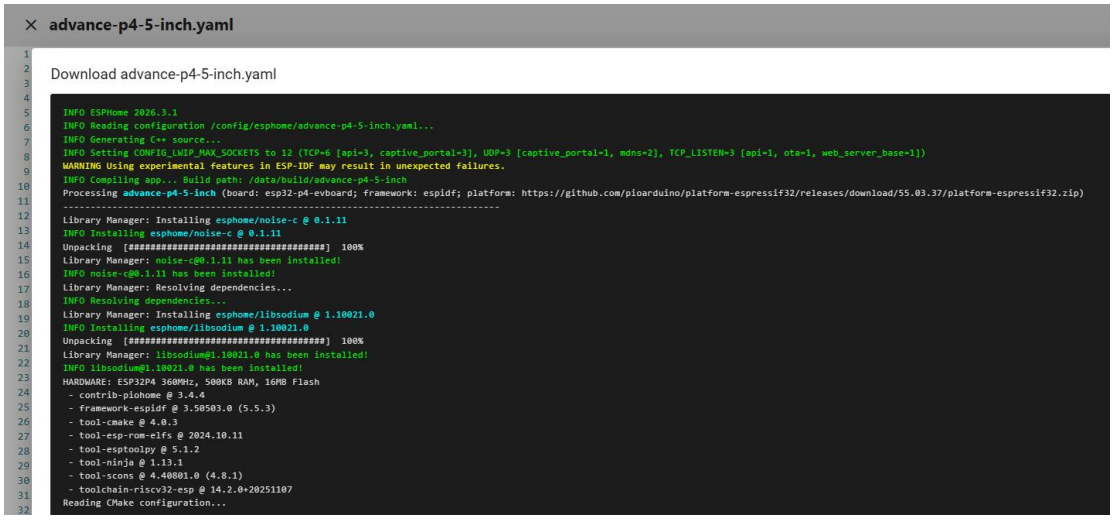
Once the code replacement is complete, click "INSTALL" in the top right corner.



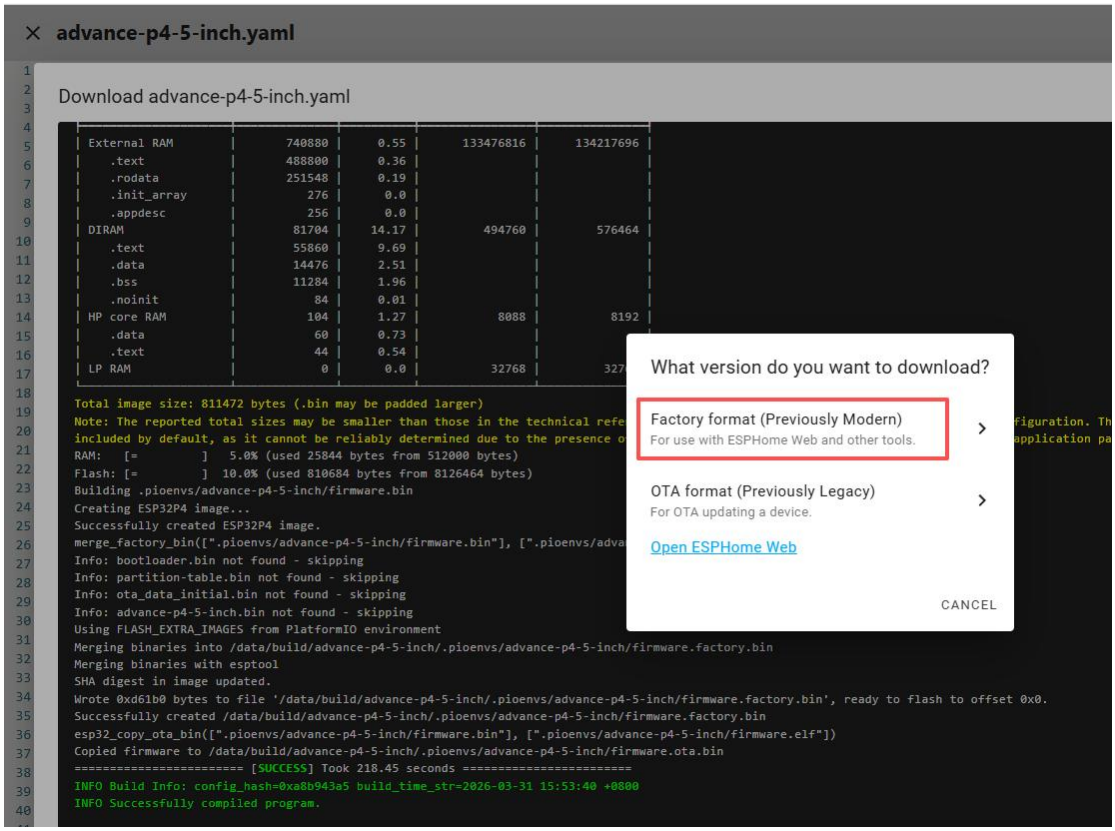
Select "Manual download".



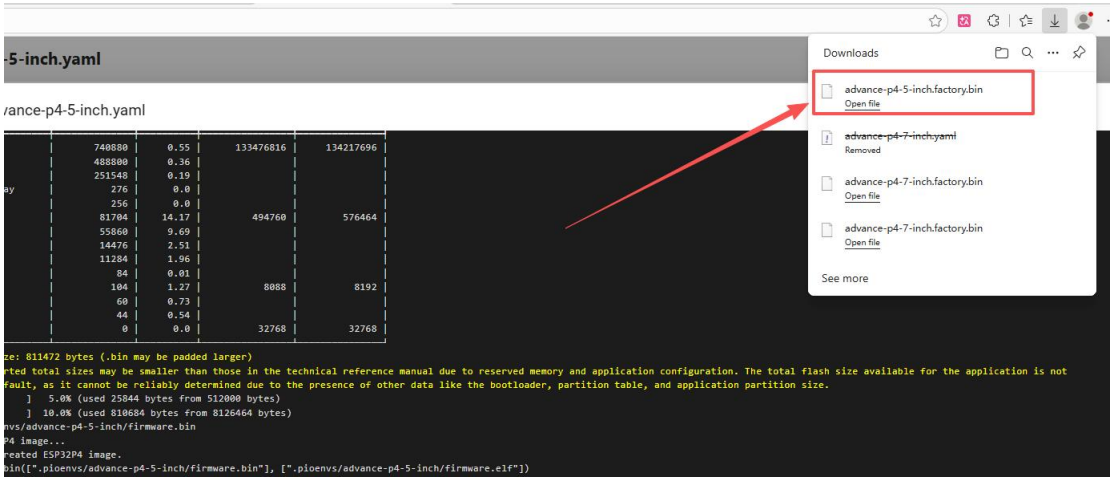
Wait for a few minutes until the installation is complete.The first installation and download process may take some time as it involves downloading necessary tools and libraries and compiling the code.



Then, after the download is completed, select "Factory format".



Once the download is complete, you will see the .bin file.



Remember the path of this .bin file.

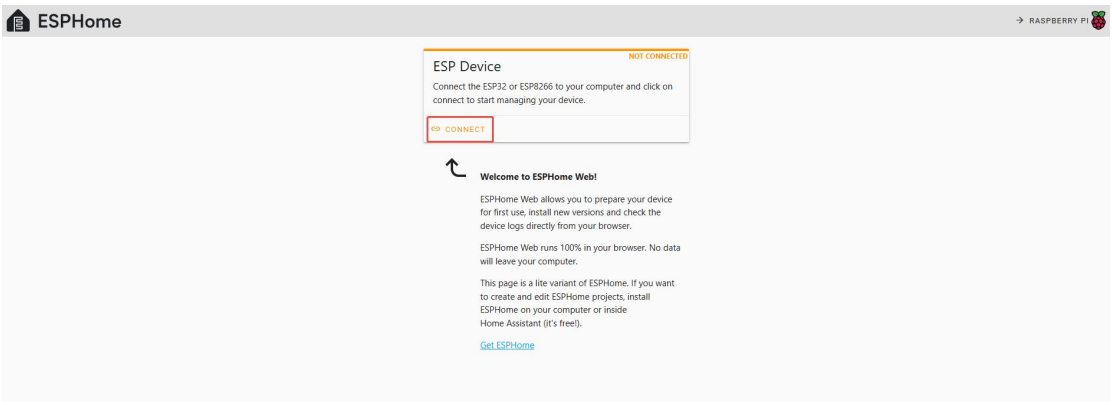
Open the following website:

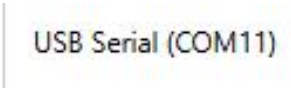
https://web.esphome.io/?dashboard_wizard

Next, we will flash this .bin file into the CrowPanel Advanced 5inch ESP32-P4 HMI AI Display .

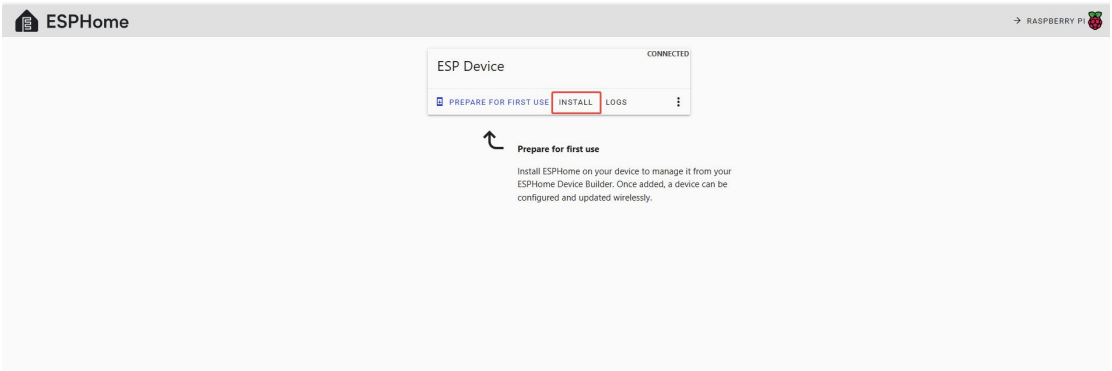
Connect the CrowPanel Advanced 5inch ESP32-P4 HMI AI Display to your computer.

Click "Connect", select the COM port, and connect it.

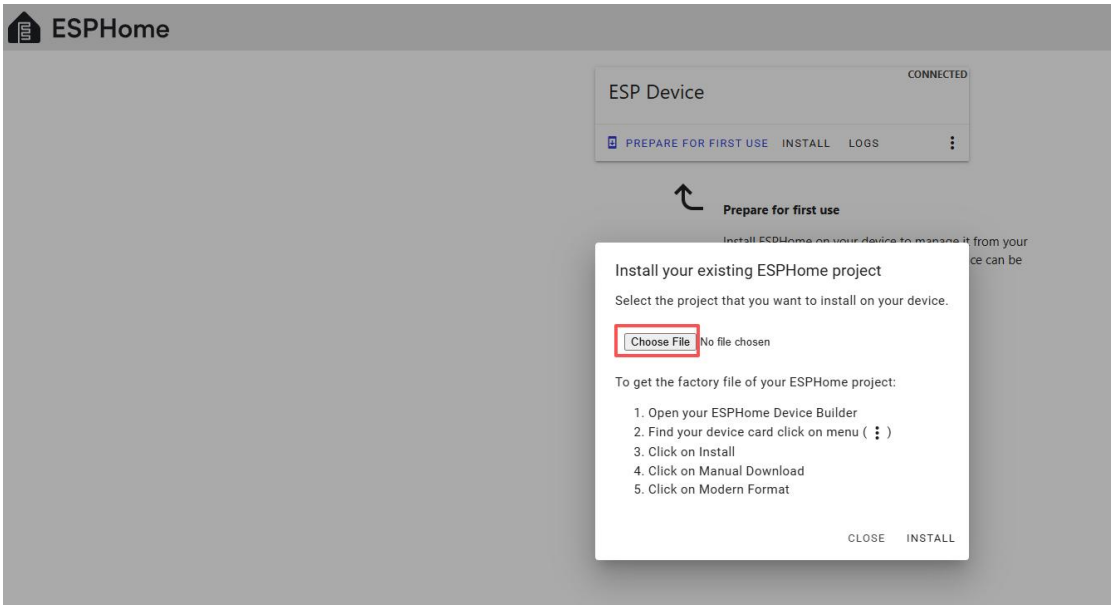


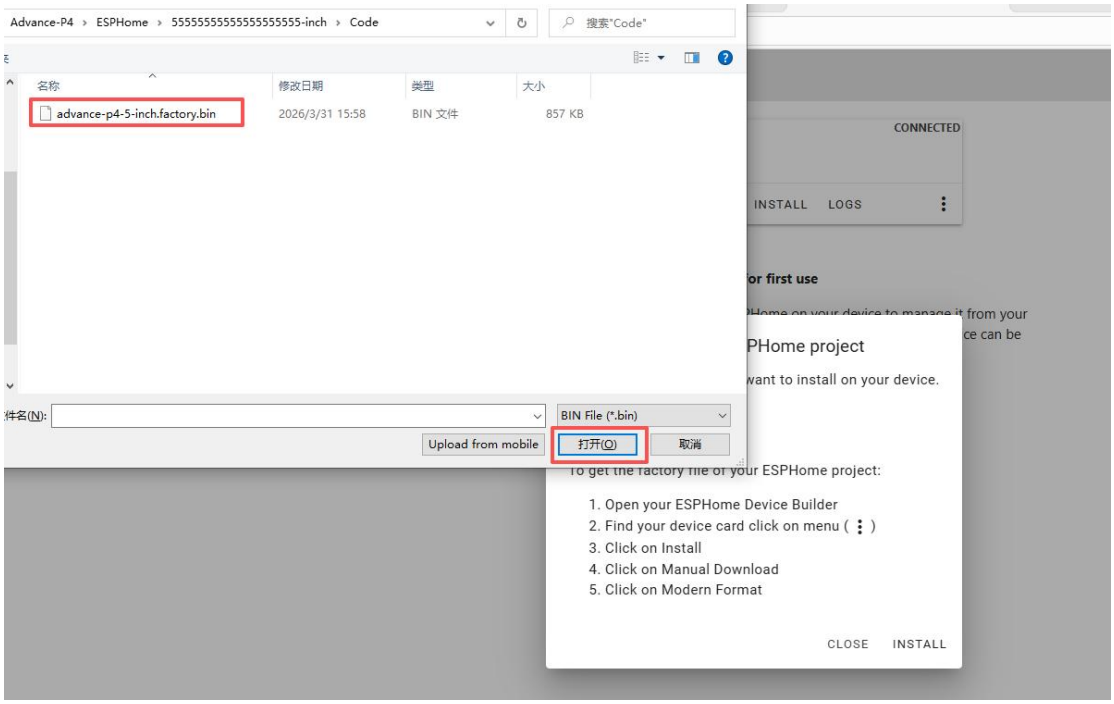


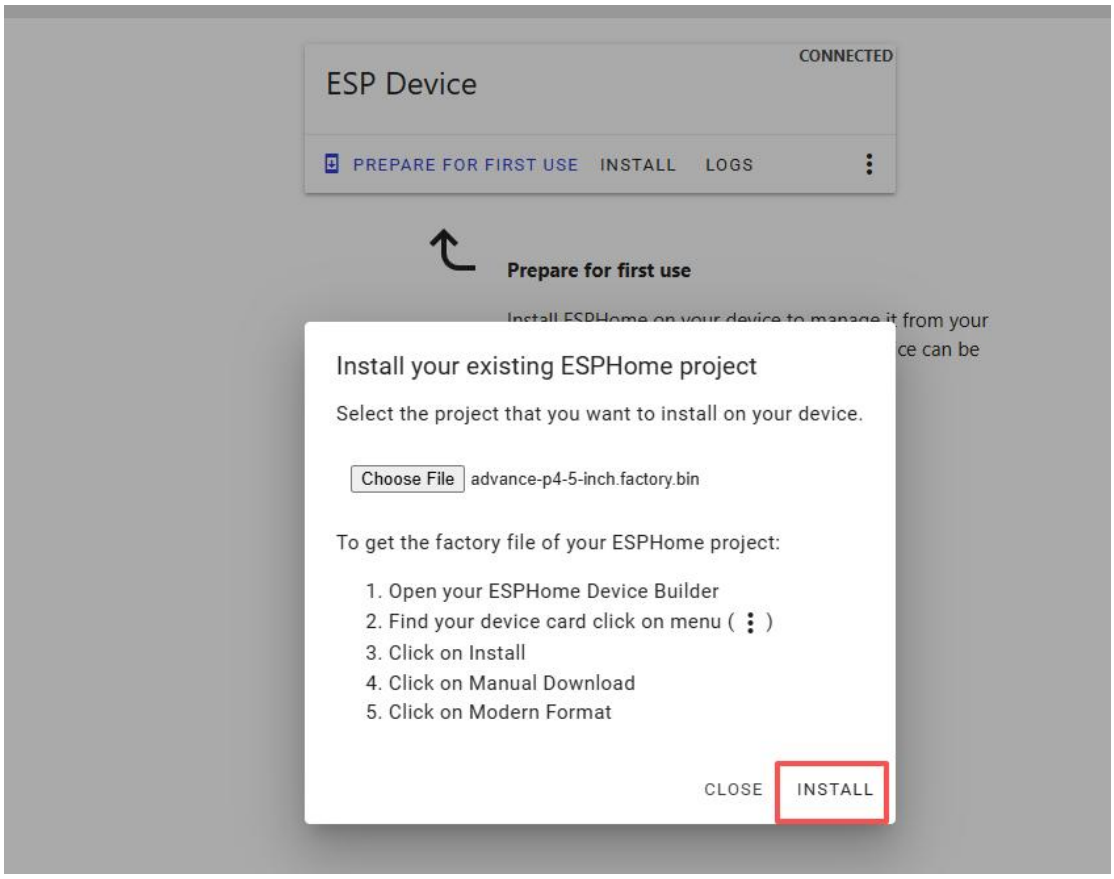
After connecting the CrowPanel Advanced 5inch ESP32-P4 HMI AI Display , click "Install".



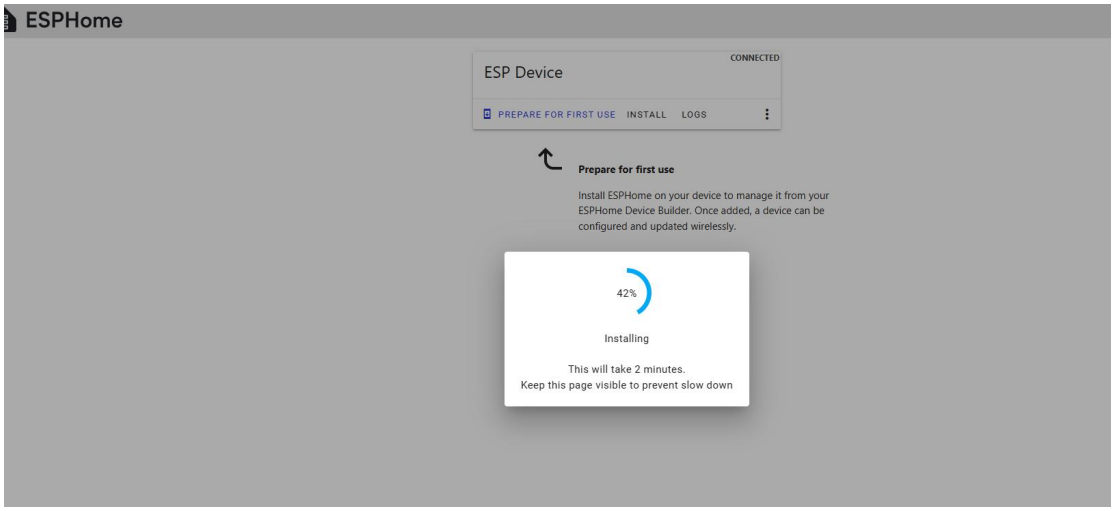
Add the .bin file you just downloaded, then click "Install".



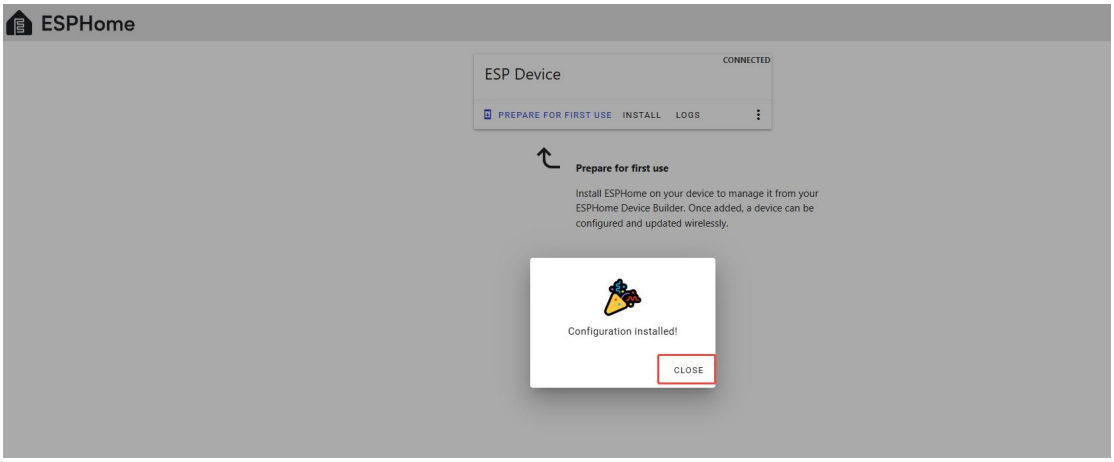




Wait for a few minutes.

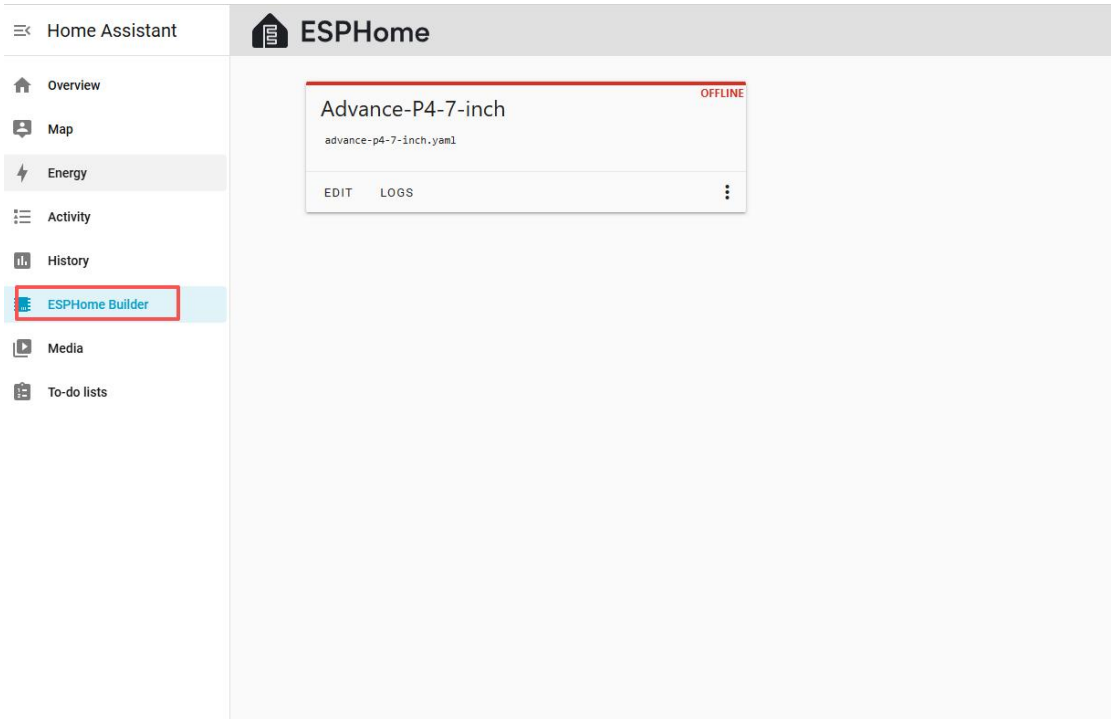


After the installation is complete, click "Close".

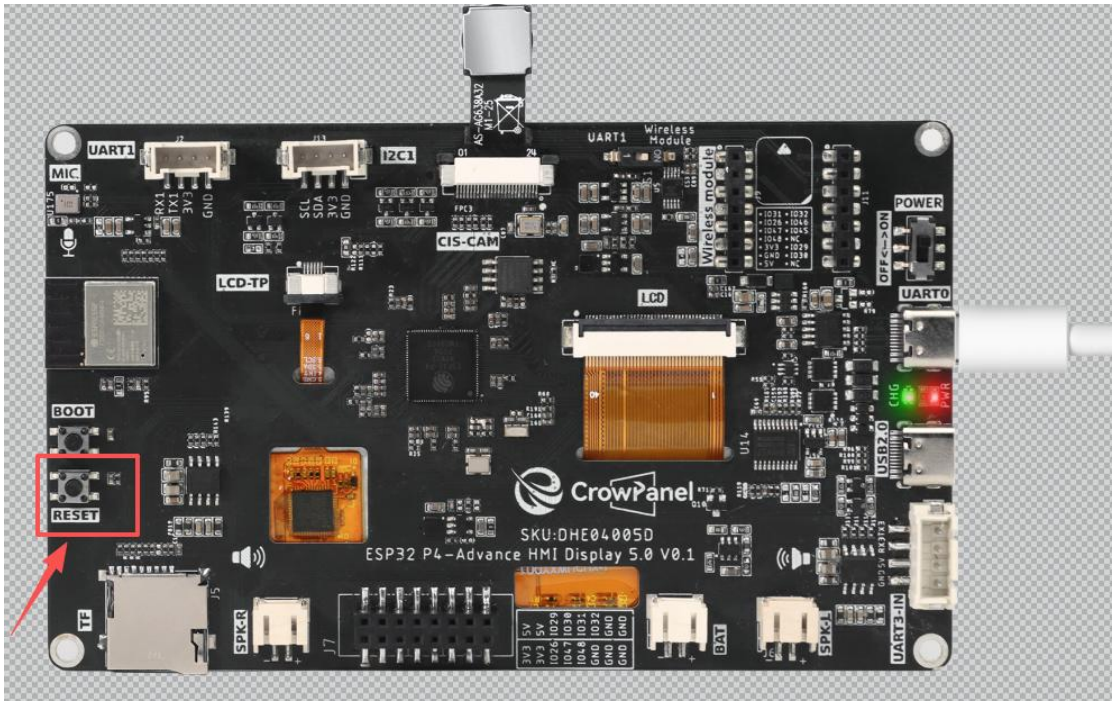


9. Observe the Wi-Fi connection status of the equipment

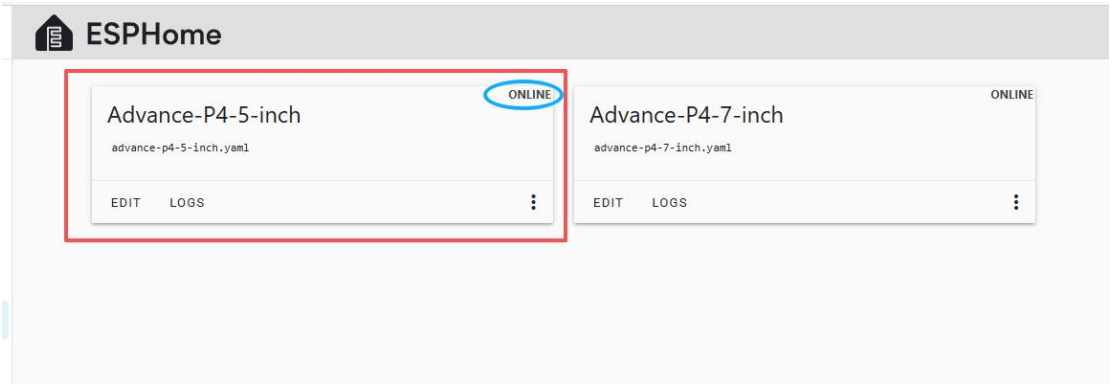
After successfully flashing the .bin file, return to the ESPHome page in Home Assistant.



Press the RESET button on the back of the product to reset it once.



And you should see the device you created earlier show as ONLINE in the top right corner.



If the "ONLINE" status is not displayed, please make sure that your Raspberry Pi 5, CrowPanel Advanced 5inch ESP32-P4 HMI AI Display , and the WIFI you are using are all within the same local network.

Once your device is activated, you can begin writing code to implement your features.

First, this session focuses on collecting temperature and humidity data, which can be viewed historically in the ESPHome backend. It also adds the function of remotely controlling the light.

We hope the features in this session will help you better understand the convenience of ESPHome in smart homes.

Next, let's get started.

10.upload pictures

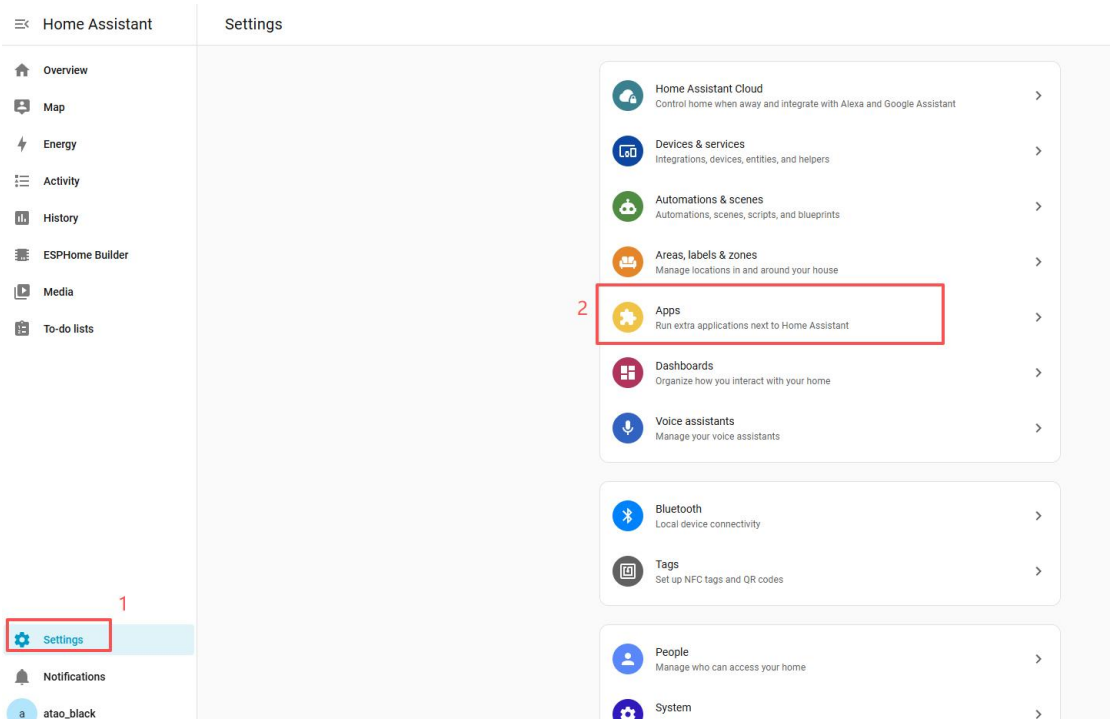
You can see the materials on the screen, which we need to use on the ESPHome platform, so they need to be uploaded to the ESPHome platform.

Click here to download the materials shown on our screen.

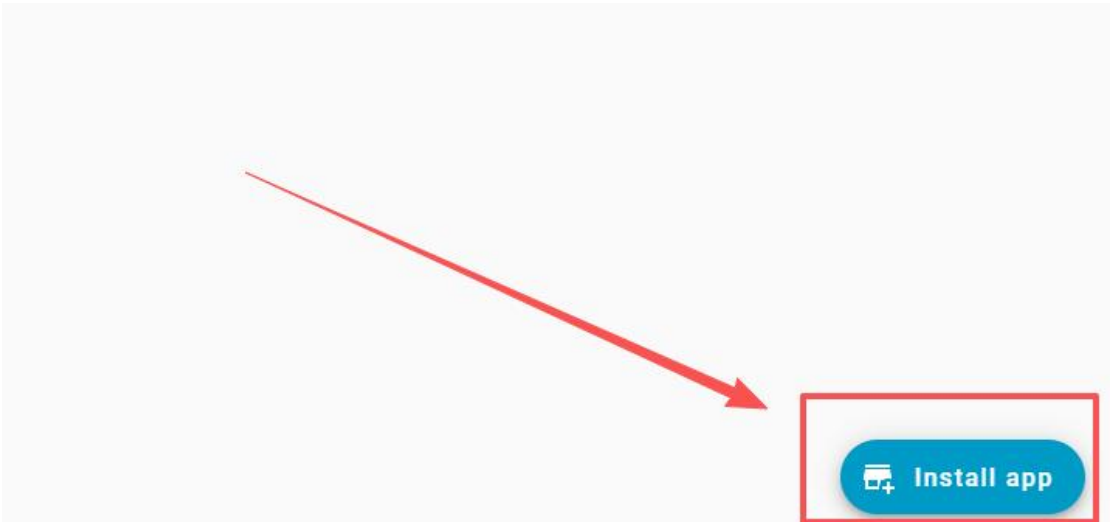
https://github.com/Elecrow-RD/-CrowPanel-Advanced-5inch-ESP32-P4-HMI-AI-Display-800x480-IPS-Touch-Screen/tree/master/example/V1.0/ESPHome/Materials

After downloading the materials we provide, you need to download a tool to upload these materials to the ESPHome platform.

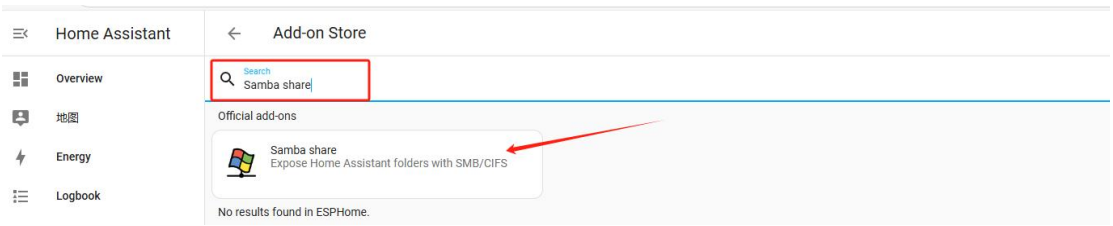
Click Settings and select Apps



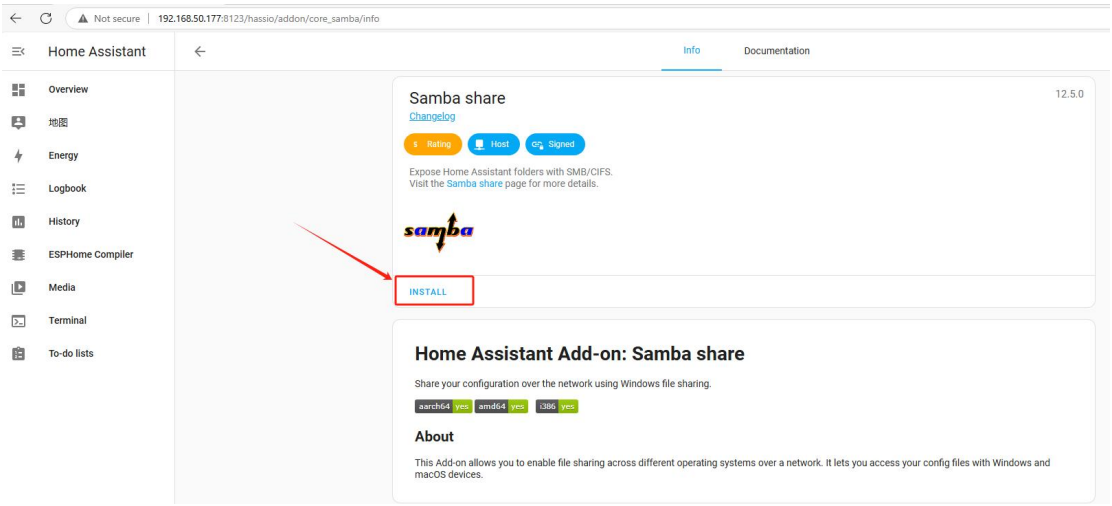
Then, click the “Install app” at the bottom right corner.



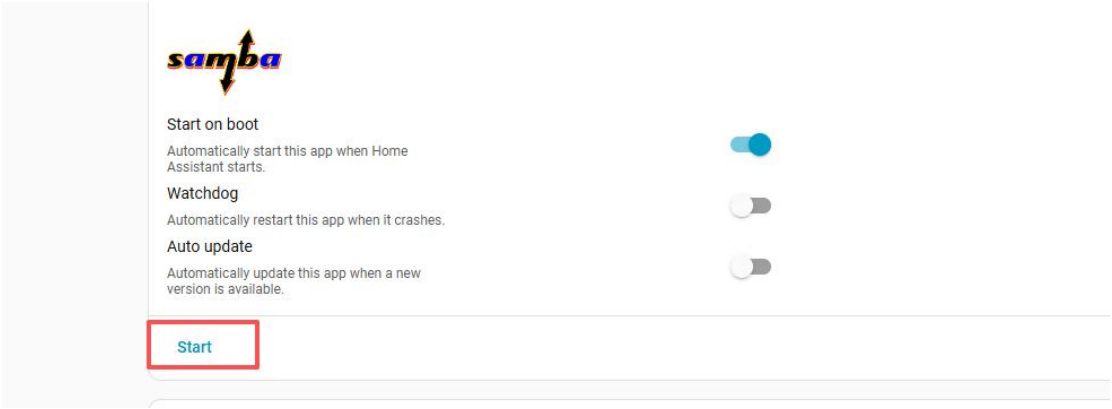
Search for Samba share at the top



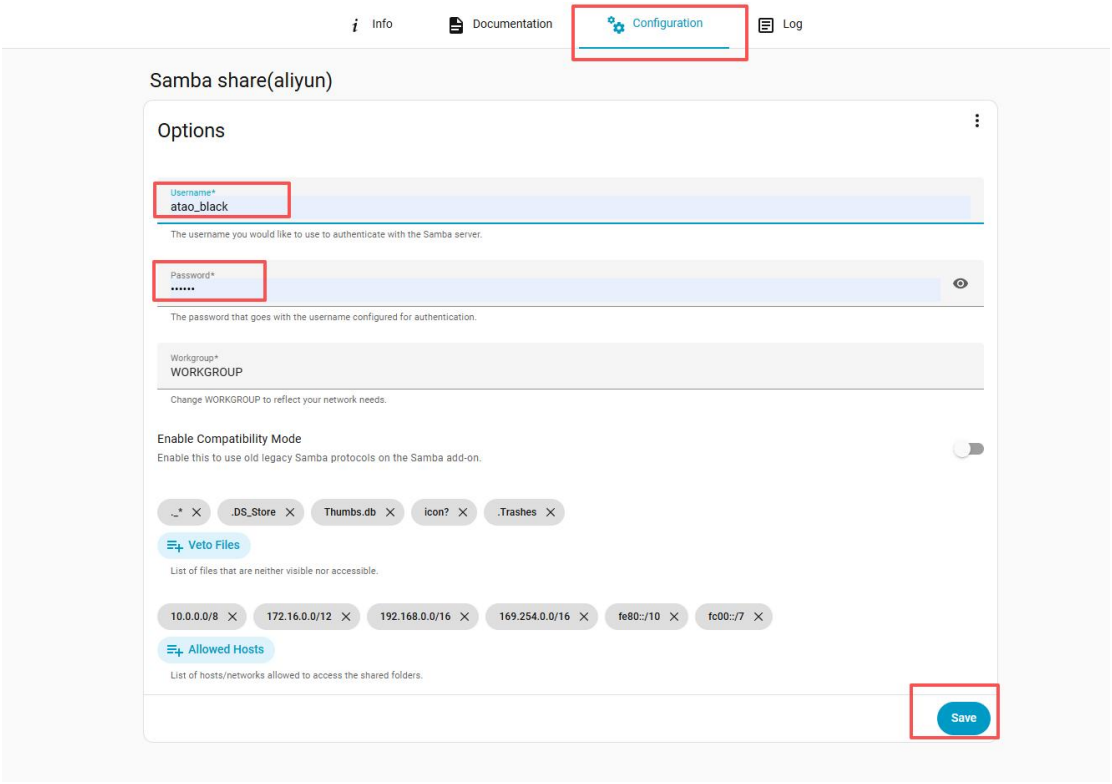
Click INSTALL to install it



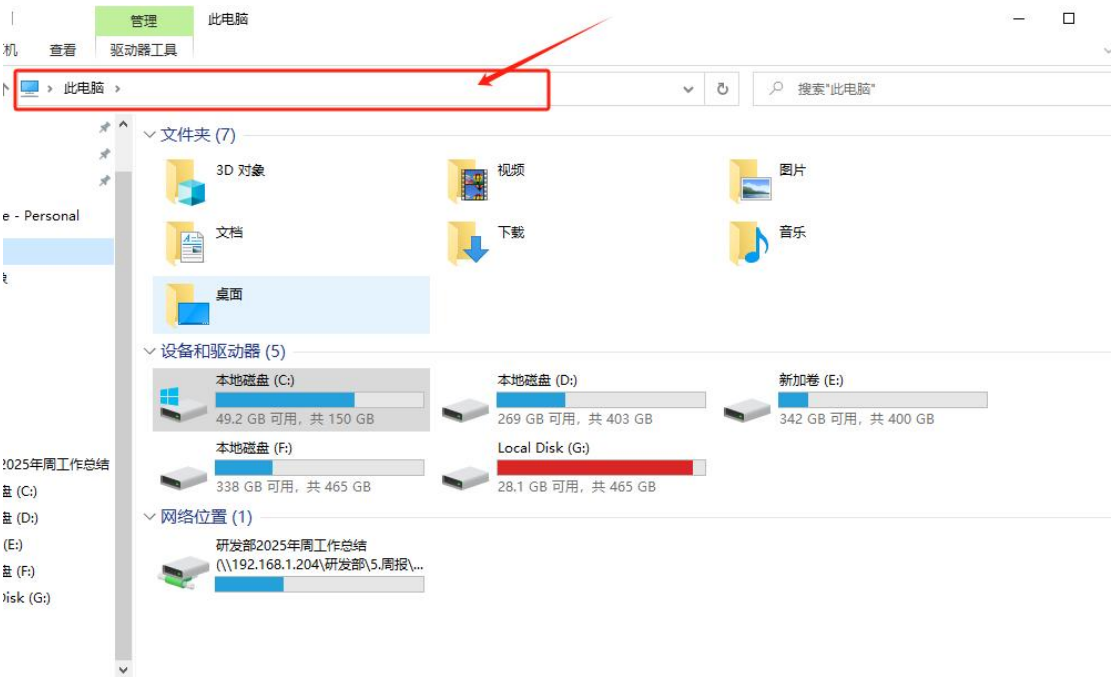
After the installation is completed, remember to start it.



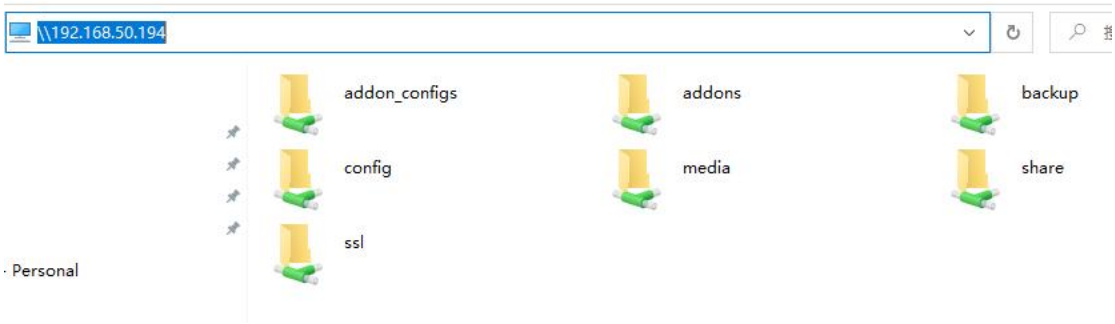
After installation, configure your Samba share with your own username and password—please remember them!



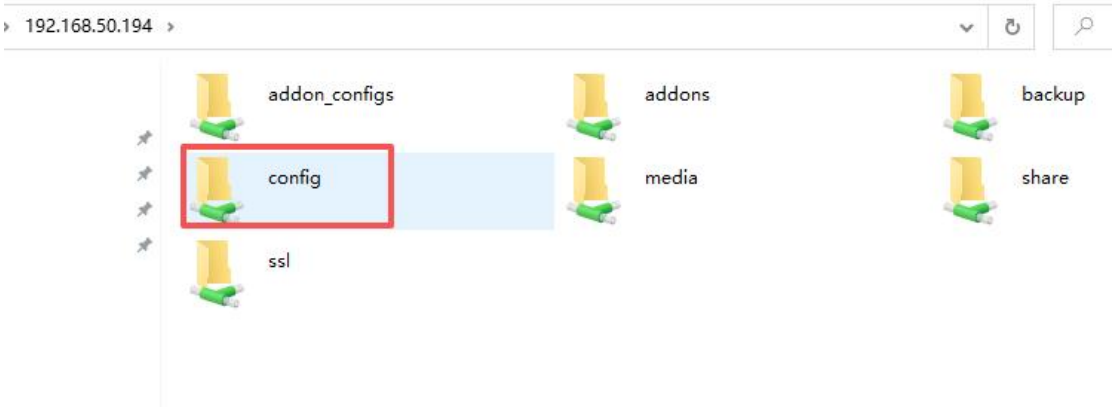
After saving, go to the File Explorer on your computer



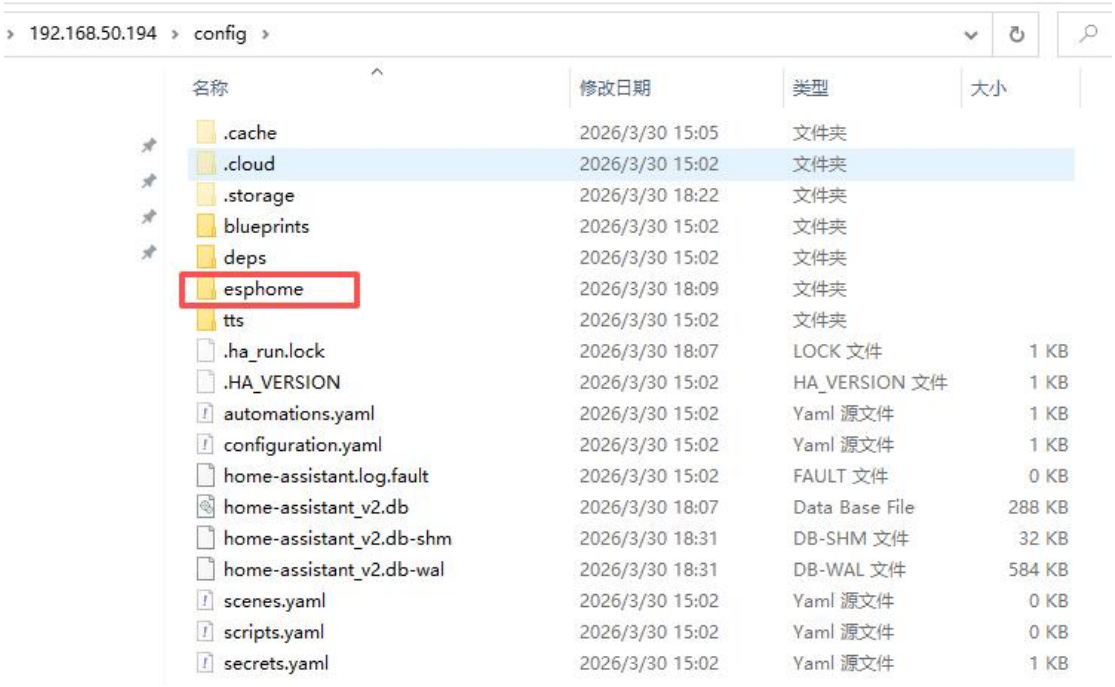
Enter \\ + your Home Assistant IP address



Go to config



Then choose esphome



Put the materials you just downloaded into the esphome folder

That completes the mission of Samba share.

11. Edit the code and complete the functionality

Next, you can edit your code to implement the simple smart home functions we mentioned.

Click "EDIT" to start writing the code.

You can use the code we provided earlier.

After downloading the code, you can copy it into your project.

Next, let’s talk about things to pay attention to while using the code.

Let's take a look at the code.

1.The ESPHome section

advance-p4-5-inch.yaml

```
esphome:
  name: advance-p4-5-inch
  friendly_name: Advance-P4-5-inch
  on_boot:
    priority: 600.0
    then:
      - lambda: |-
          uint8_t cmd[2] = {0x20, 100};
          id(bus_a).write(0x2F, cmd, 2);
```

When the device is just powered on, it automatically sends an instruction via the I2C bus to the screen backlight control chip, setting the screen brightness to 100, thereby lighting up the screen and maintaining a relatively bright state.

Specifically, "on_boot" indicates "execute on boot", meaning that the content within this section will automatically run once the ESP32-P4 is powered on or restarted; priority: 600.0 indicates the execution priority, with a higher value indicating earlier execution. Here, it is set to 600, indicating that it will run at a relatively earlier stage during the system initialization process (this can prevent the screen from remaining completely black for a long time).

Next is lambda, which essentially is a piece of C++ code embedded in ESPHome to perform more low-level and flexible operations; in this code, an array of length 2 named cmd is defined, with the two values being 0x20 and 100. You can think of it as a "control command", where 0x20 usually represents the "register address or instruction number for controlling the backlight brightness", and 100 is the brightness value to be set (generally ranging from 0 to 100, with 100 being the brightest).

The last line, id(bus_a).write(0x2F, cmd, 2), means: through the I2C bus bus_a that has been defined earlier, send this command to the device with the address 0x2F, and send 2 bytes of data; here, 0x2F can be understood as the "address of the backlight control chip", just like the recipient's address in an express delivery, only if the address is correct, the chip will receive and execute this command.

2.esp32

This piece of code serves to inform ESPHome which chip you are using and how this chip should operate with certain performance and parameters. It can be regarded as "setting the hardware foundation and operation mode for the system".

```
esp32:
  variant: esp32p4
  engineering_sample: true
  cpu_frequency: 360MHZ
  flash_size: 16MB
  framework:
    type: esp-idf
    advanced:
      execute_from_psram: true
      enable_idf_experimental_features: true
```

Firstly, variant: esp32p4 indicates that you are using the ESP32-P4 chip, which is a relatively new and high-performance chip at present;
Then, engineering_sample: true means that this chip still belongs to an "engineering sample" (not yet fully officially mass-produced), so some special support needs to be enabled; otherwise, many functions may not work properly.

Next, cpu_frequency: 360MHZ indicates that the CPU operating frequency is set to 360 MHz, which is faster than the ordinary ESP32, equivalent to "turning on the high-performance mode" for the device, suitable for your project with a 5-inch large screen (LVGL interface);
flash_size: 16MB tells the system that your storage space is 16MB, so this allows you to know how much program and image resources can be stored during compilation.

The following framework section is more crucial. It specifies to use ESP-IDF (the official underlying development framework), instead of Arduino, so as to obtain stronger underlying control capabilities and better performance;
In advanced, execute_from_psram: true indicates that the program can run in PSRAM (external large memory), which is very important for large-screen UI and images. Otherwise, insufficient

memory will cause a direct crash;

Finally, enable_idf_experimental_features: true indicates to enable some "experimental features" of ESP-IDF. Since ESP32-P4 is still relatively new, many functions are still in the testing stage, and this option needs to be enabled to use them normally.

3.psram + logger + api + ota

This section of code is mainly responsible for performing system-level resource configuration + debugging output + network communication capabilities (remote control and upgrade). It can be understood as: enabling the device to "have sufficient memory to run the interface, be able to output debugging information, and also be able to connect to the network for control and remote upgrade".

```
22  psram:
23    speed: 200MHz
24
25  # Enable logging
26  logger:
27    level: debug
28    logs:
29      lvgl: info
30    hardware_uart: UART0
31
32  # Enable Home Assistant API
33  api:
34    encryption:
35      key: "d4yhS7jMb1+LEJ8wpz2KLSMJR73evpUJazYyY1ab6EU="
36
37  ota:
38    - platform: esphome
39      password: "7e67f5f46de710476467d7daed31b484"
```

Firstly, PSRAM: Speed: 200MHz refers to configuring external memory (PSRAM). You can think of it as "extra large memory". Since this project has a 5-inch screen, images, and an LVGL interface, all of which consume a lot of memory, PSRAM must be used. And here, the speed is set to 200MHz, which means making this "external memory" run faster to ensure the interface does not become unresponsive. Without this part, there is a high probability that there will be insufficient memory or the program will run very slowly.

Then comes the logger, which is the logging system:

```
logger:
  level: debug
```

Indicates the activation of the highest level of debugging output, where almost all details of the program's operation will be printed out. This is highly suitable for troubleshooting during the development stage;

logs: lvgl: info indicates that the logs for LVGL (interface system) are set to the info (information level), avoiding the display of too many details;

hardware_uart: UART0 indicates that the logs are output through UART0 (the content you can see in your computer's serial port monitor).

In simple terms: This part is "allowing you to see what is happening inside the device".

Further down is the API:

```
32  # Enable Home Assistant API
33  api:
34    encryption:
35      key: "d4yhS7jMb1+LEJ8wpz2KLSMJR73evpUJazYyY1ab6EU="
36
```

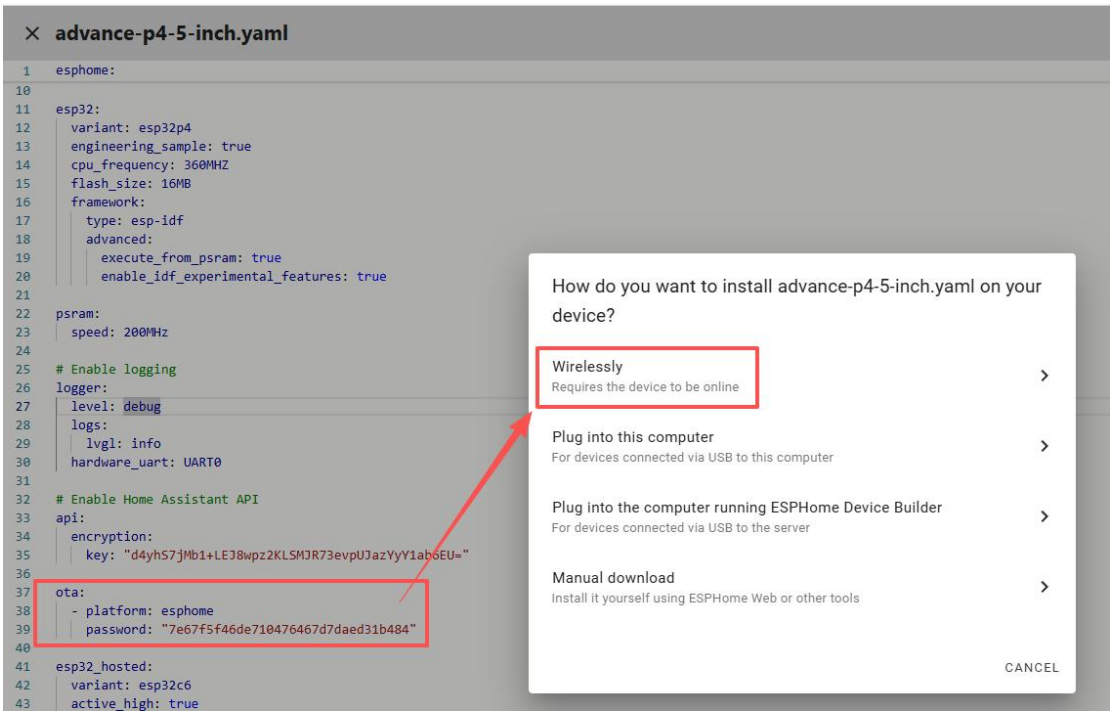
This is used to connect the device to Home Assistant.
After enabling it, you can remotely control this device from your phone or computer (such as turning on the lights, checking the temperature and humidity).
An encryption key has also been added here, meaning the communication is encrypted to prevent others from casually controlling your device.
It can be understood as: a "remote control entry has been added to the device, and it is encrypted and secure".
This section is generated when creating the file and should not be modified.

Finally, it's ota:

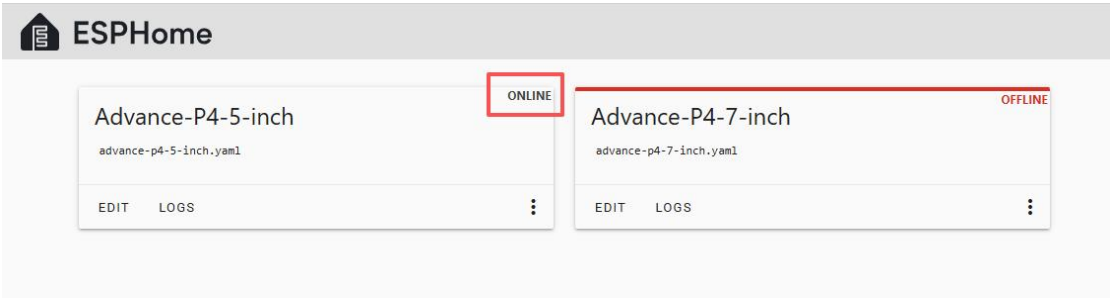
```
37  ota:
38    - platform: esphome
39      password: "7e67f5f46de710476467d7daed31b484"
40
```

This is the wireless upgrade feature (OTA), which is what you will be able to use in the future.
A password has been set, indicating that you will need to enter the password for firmware updates in the future to ensure that others cannot tamper with your device.

With this feature, you won't need to plug in a USB cable anymore. Instead, you can directly update the program via WiFi.



(The prerequisite is that your computer and the Wi-Fi are in the same local network, and your device is in the ONLINE state, indicating that the Wi-Fi connection is successful.)



4.wifi

This piece of code is actually accomplishing a very crucial task: enabling the WiFi-less ESP32-P4 to "borrow" the networking capabilities of ESP32-C6, and configuring the wireless network connection and backup hotspot.

```yaml
esp32_hosted:
  variant: esp32c6
  active_high: true
  reset_pin: GPIO32
  cmd_pin: GPIO19
  clk_pin: GPIO18
  d0_pin: GPIO14
  d1_pin: GPIO15
  d2_pin: GPIO16
  d3_pin: GPIO17

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Advance-P4-7-Inch"
    password: "eW0iGlqyJxnC"

captive_portal:
```

Firstly, the "esp32_hosted" section is the core. It means that the main control ESP32-P4 will "control" an external ESP32-C6, treating C6 as a WiFi network card, because P4 itself does not have WiFi functionality. The set of pins (such as cmd_pin, clk_pin, d0~d3) are the "data channels" for communication between the two chips, similar to a high-speed data line, used to transmit network data; reset_pin allows P4 to control the restart of C6, ensuring that communication can be restored in case of abnormality.

Next, the WiFi section is for setting the wireless network that the device needs to connect to (account and password). Although it is written here, in fact, it is C6 that is connecting to WiFi; if the connection fails, the ap will make the device automatically become a hotspot (the name is Advance-P4-5-Inch), and you can connect to it with your mobile phone for configuration.

The default name and password of Wifi can be modified here.

secrets.yaml

```yaml
# Your Wi-Fi SSID and password
wifi_ssid: "elecrow888"
wifi_password: "elecrow2014"

```

The final "captive portal" is a function designed to be used in conjunction with this hotspot. When you connect to the hotspot, a web page will automatically pop up, allowing you to enter the new WiFi information.

5. Temperature and Humidity

This code is designed to enable the device to connect a temperature and humidity sensor (AHT20) via a communication line called I2C, and to continuously read environmental data.

```
i2c:
  - id: bus_a
    sda: GPIO45
    scl: GPIO46
    frequency: 400kHz

sensor:
  - platform: aht10
    variant: AHT20
    temperature:
      name: "Room Temperature"
      id: room_temp
      unit_of_measurement: "°C"
      accuracy_decimals: 1
    humidity:
      name: "Room Humidity"
      id: room_hum
      unit_of_measurement: "%"
      accuracy_decimals: 1
    update_interval: 1s

```

The i2c configuration in the first part is equivalent to establishing a "communication channel". It uses GPIO45 (data line) and GPIO46 (clock line) to communicate with the external sensor at a speed of 400kHz, ensuring fast and stable data transmission;
The sensor part in the latter part is informing the system that you are connecting an AHT20 type sensor, which can measure both temperature and humidity simultaneously. It has defined two data items: one is the temperature (variable name is room_temp, with units in degrees Celsius and retaining 1 decimal place), and the other is the humidity (variable name is room_hum, with units in percentage and also retaining 1 decimal place);

Finally, through update_interval: 1s, it specifies to automatically read the sensor data once every 1 second. Overall, the function of this code is: allowing the device to communicate with the temperature and humidity sensor through the specified pin, and obtaining the current environmental temperature and humidity data at a frequency of once per second, for subsequent interface display or remote viewing purposes.

6.touchscreen

This touchscreen configuration defines the usage method of the GT911 capacitive touchscreen within the ESPHome system.

```yaml
touchscreen:
  - platform: gt911
    id: my_touchscreen
    i2c_id: bus_a
    reset_pin: 36
    interrupt_pin: 42
    update_interval: 50ms
    transform:
      swap_xy: false
      mirror_x: false
      mirror_y: false
    on_touch:
      - logger.log:
          format: "Touch at (%d, %d)"
          args: [touch.x, touch.y]
      - lambda: |-
          ESP_LOGI("cal", "x=%d, y=%d, x_raw=%d, y_raw=%d",
          touch.x,
          touch.y,
          touch.x_raw,
          touch.y_raw
          );
```

It communicates with the main controller via the I2C bus 'bus_a', with reset_pin: 36 used for hardware reset of the touchscreen and interrupt_pin: 42 for detecting touch events. update_interval: 50ms indicates that the touch state is scanned once every 50 milliseconds to ensure timely response.

The transform configuration allows for transformation of touch coordinates, such as swapping X/Y or mirroring flip. Here, all are set to false, indicating to maintain the original direction.

The on_touch defines the touch callback: every time a touch occurs, the logger.log is used to print the screen coordinates of the touch point, and through lambda, debug information is output, including the calibrated coordinates touch.x, touch.y and the original raw ADC readings touch.x_raw, touch.y_raw, facilitating debugging and touch calibration.

7.switch

This switch configuration defines a GPIO-controlled LED switch and is linked with the LVGL graphical interface.

```yaml
switch:
  - platform: gpio
    pin: GPIO48
    name: "LED"
    id: led_output
    on_turn_on:
      # Switch bulb diagram
      - lvgl.image.update:
          id: lvgl_light
          src: light_on
      # Switch the status label text to ON
      - lvgl.label.update:
          id: label_light_state
          text: "ON"
    on_turn_off:
      # Switch bulb diagram
      - lvgl.image.update:
          id: lvgl_light
          src: light_off
      # Switch the status label text to OFF
      - lvgl.label.update:
          id: label_light_state
          text: "OFF"
```

This switch configuration in ESPHome defines an LED switch based on GPIO48. It not only controls the on-off of the physical LED but also synchronizes the display and status feedback with the LVGL graphical interface.

When the user or program triggers the switch to open (on_turn_on), ESPHome sets GPIO48 to output a high level, thereby lighting up the actual connected LED, and triggers the LVGL control to update: first, switch the corresponding image control lvgl_light on the screen to the pre-loaded light_on image to give a visual effect of the bulb lighting up on the interface;

Then update the text of the status text control label_light_state to "ON", clearly indicating the current state of the LED to the user. On the contrary, when the switch is closed (on_turn_off), GPIO48 outputs a low level to turn off the LED, and the LVGL interface synchronously switches the image to light_off and displays "OFF" as the status label, ensuring that the physical light and the screen display are completely consistent.

```yaml
- platform: template
  name: "Backlight Power"
  id: backlight_switch
  optimistic: true
  restore_mode: RESTORE_DEFAULT_ON
  internal: true
  turn_on_action:
    - lambda: |-
        uint8_t cmd[2] = { 0x20, 100 };
        id(bus_a).write(0x2F, cmd, 2);
  turn_off_action:
    - lambda: |-
        uint8_t cmd[2] = { 0x20, 0 };
        id(bus_a).write(0x2F, cmd, 2);
```

The latter part is a template type virtual switch (not directly connected to GPIO), named "Backlight Power", which is mainly used to control the screen backlight. Its characteristics are optimistic: true (does not read the actual state, directly assumes the operation is successful) and restore_mode: RESTORE_DEFAULT_ON (the device is default on with backlight on after a reboot), and internal: true indicates that this switch will not be displayed in the front-end interface. When you turn on this switch, a piece of lambda code will be executed, sending {0x20, 100} to the device at address 0x2F via I2C to set the brightness to 100 (the screen becomes brighter), and when turned off, sending {0x20, 0} to set the brightness to 0 (the screen goes off).

8.spi + display

```yaml
spi:
  - id: spi_bus
    clk_pin: GPIO26
    mosi_pin: GPIO47

display:
  - platform: mipi_rgb
    model: custom
    id: rpi_disp
    update_interval: never
    auto_clear_enabled: false
    color_order: RGB
    pclk_frequency: 16MHz
    dimensions:
      width: 800
      height: 480
    de_pin:
      number: 2
    hsync_pin:
      number: 40
    vsync_pin:
      number: 41
    pclk_pin: 3
    pclk_inverted: true
    hsync_back_porch: 4
    hsync_front_porch: 8
    hsync_pulse_width: 8
    vsync_back_porch: 4
    vsync_front_porch: 8
    vsync_pulse_width: 8
    data_pins:
      red:
        - 19         #r3
        - 18         #r4
        - 17        #r5
        - 16        #r6
        - 15        #r7
      blue:
```

The previous spi section is quite simple. It defines an SPI bus named spi_bus, using GPIO26 as the clock and GPIO47 as the data output (MOSI). However, in your current code, there are no devices currently using it. It can be understood as "to be used in the future, such as connecting an SD card or other peripherals".

The truly core part is the "display": platform: mipi_rgb indicates that you are using a parallel RGB screen (not SPI screen), which has a fast refresh rate but requires many pins to cooperate; model: custom indicates that you are using custom parameters (not a ready-made driver), so all timings

need to be configured by yourself; update_interval: never indicates that it does not automatically refresh (because you are using LVGL to actively control the refresh), auto_clear_enabled: false indicates that it will not automatically clear the screen each time (to avoid flickering and improve performance).

Next, dimensions: 800x480 defines the screen resolution, while color_order: RGB and pclk_frequency: 16MHz indicate the color order and pixel clock frequency (which can be understood as "how many pixels are transmitted per second"). Then, the most crucial set of "timing control signals": de_pin (data enable), hsync_pin (horizontal synchronization), vsync_pin (vertical synchronization), pclk_pin (pixel clock), these signals jointly determine when the screen starts drawing a row, when it switches to the next row, and when it refreshes the entire screen; the parameters such as hsync_back_porch / front_porch / pulse_width and vsync_* can be understood as "waiting time and rhythm control during the screen refresh process".

If these parameters are incorrect, there will be a flickering, offset or no display. Further down is data_pins, where it defines which GPIOs are used to transmit data for the three colors of RGB (for example, red uses 5 lines (r3~r7), green uses 6, and blue uses 5), which is actually the RGB565 color format (a total of 16 bits), with each bit transmitted in real time through one line, so many pins are needed.

The final init_sequence is the initialization command, 0x01 usually indicates software reset, 0x3A, 0x66 indicate setting the pixel format (here corresponding to RGB565), ensuring that the screen and ESP32 output data formats are consistent.

Putting all this logic together is: The ESP32-P4, based on these pin connections and timing parameters, sends RGB pixel data row by row to the screen in parallel at high speed, and the screen then correctly refreshes the complete 800×480 image according to these synchronization signals.

9.image

The function of this code can be understood as: Load all the image resources used in your project into the device in advance, and uniformly convert them into the format suitable for screen display, so that they can be called at any time in the LVGL interface.

```
image:
  - file: "small_temp.png"
    id: small_temp
    resize: 200x200
    type: RGB565
  - file: "small_hum.png"
    id: small_hum
    resize: 200x200
    type: RGB565
  - file: "small_logo.png"
    id: small_logo
    resize: 200x100
    type: RGB565
  - file: "light.png"
    id: light_on
    resize: 200x200
    type: RGB565
    transparency: alpha_channel
  - file: "no_light.png"
    id: light_off
    resize: 200x200
    type: RGB565
    transparency: alpha_channel
```

Specifically, each item below represents a configuration for an image. For example, "file: 'small_temp.png'" indicates the filename of the image (usually located in the project directory), "id: small_temp" is the "internal name" given to this image, and later in LVGL, this id is used to reference it; "resize: 200x200" means that the image will be resized to the specified size during compilation, which can reduce memory usage, improve display speed, and avoid performance overhead caused by resizing at runtime; "type: RGB565" indicates that the image is converted to a 16-bit color format (consistent with the color depth of your screen), so it will not cause errors during display and saves more memory.

The two light images (light_on and light_off) have an additional "transparency: alpha_channel", indicating that they have a transparent background (for example, the icon of the light is circular, but the background is transparent). This way, when displayed on the interface, there will be no "white square" but can be well overlaid on the background.

The overall process is: during compilation, convert the PNG image → convert to RGB565 → compress according to the specified size → store in the device → at runtime, call by id and display on the screen. This ensures the display effect while also taking into account the limited memory and performance of the embedded device.

10.lvgl

This LVGL configuration in ESPHome defines the core rendering and control management of the touchscreen interface.

```yaml
lvgl:
  buffer_size: 50%
  color_depth: 16
  log_level: INFO
  bg_color: 0xFFFFFF
  text_font: montserrat_26
  widgets:
    - image:
        id: lvgl_logo
        src: small_logo
        x: 300
        y: 20
    - image:
        id: lvgl_temp
        src: small_temp
        x: 150
        y: 250
    - image:
        id: lvgl_hum
        src: small_hum
        x: 400
        y: 250
    - image:
        id: lvgl_light
        src: light_off
        x: 650
        y: 250
    - label:
        id: label_temp
        x: 90
        y: 350
        text: "Temp: --°C"
    - label:
        id: label_hum
        x: 350
        y: 350
        text: "Humi: --%"
    - label:
        id: label_light_state
        x: 650
        y: 350
        text: "OFF"
```

The function of this code can be understood as: using the LVGL graphics library to build a complete user interface (UI) on the screen, including background, image icons, and text labels, and placing them on the screen at specific coordinates.

This results in the formation of the entire interface layout that you see. In the previous global configuration, buffer_size: 50% indicates that half of the memory is used as a graphics buffer to improve the smoothness of interface refresh; color_depth: 16 indicates the use of RGB565 color (consistent with the screen); bg_color: 0xFFFFFF indicates that the entire screen background is white; text_font: montserrat_26 specifies that the default font size is 26 (used for all text display).

The widgets below are the core, which define all the "components" on the interface: the first four are image (pictures), namely logo, temperature icon, humidity icon, and the status icon of the light, each component has an id (for convenient control by subsequent code), src points to the previously defined image resources, x and y indicate its position on the screen (the top left corner

is the origin, and the unit is pixels); for example, lvgl_light initially uses light_off, indicating that the default light is in the off state.

The last three are label (text labels), used to display temperature, humidity, and the status of the light, such as "Temp: --°C" and "Humi: --%" are initial placeholder contents (not yet reading sensor data), "OFF" indicates that the light is initially off. In general, this can be understood as: on an 800×480 screen, place the icons and text at specific coordinates, display an "initial interface" first, and then update these contents dynamically through code (such as temperature values, light status, etc.).

11.interval

This interval configuration in ESPHome enables the function of periodically refreshing the temperature and humidity values on the LVGL interface. It executes a lambda code block every 1 second, thereby achieving real-time display of sensor data.

```
interval:
  - interval: 1s
    then:
      - lambda: |-
          char temp_str[20];
          char hum_str[20];
          snprintf(temp_str, sizeof(temp_str), "Temp: %.1f°C", id(room_temp).state);
          snprintf(hum_str, sizeof(hum_str), "Humi: %.1f%%", id(room_hum).state);
          lv_label_set_text(id(label_temp), temp_str);
          lv_label_set_text(id(label_hum), hum_str);
```

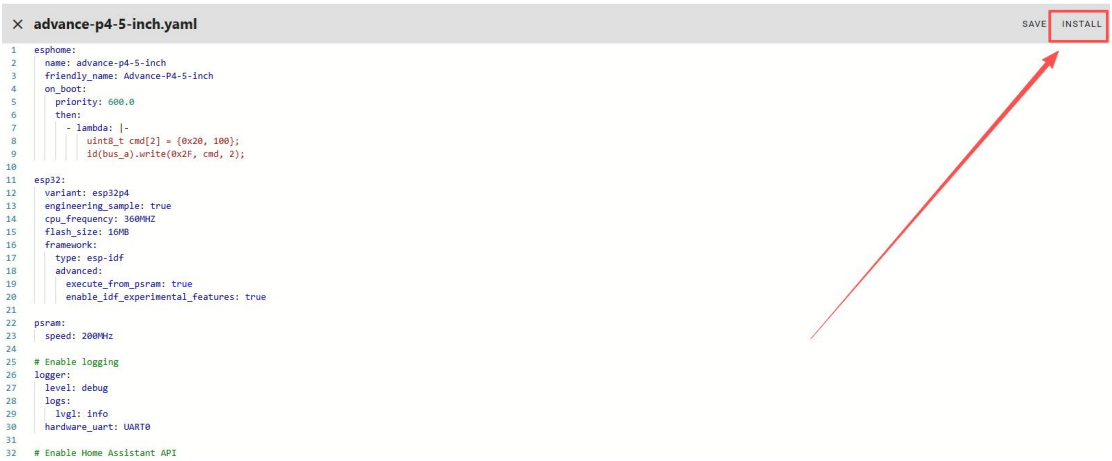
The specific logic is as follows: Firstly, define two character arrays, temp_str and hum_str, to store the formatted temperature and humidity strings;

Then, use the snprintf function to format the status value of the room_temp sensor into the string "Temp: xx.x°C", and format the status value of room_hum into the string "Humi: xx.x%". Ensure that the values retain one decimal place and generate text that can be directly displayed;
Finally, call the lv_label_set_text function of LVGL to update the text content of the label_temp and label_hum control widgets to the latest formatted temperature and humidity strings, thereby reflecting the current environmental data in real time on the touch screen interface.

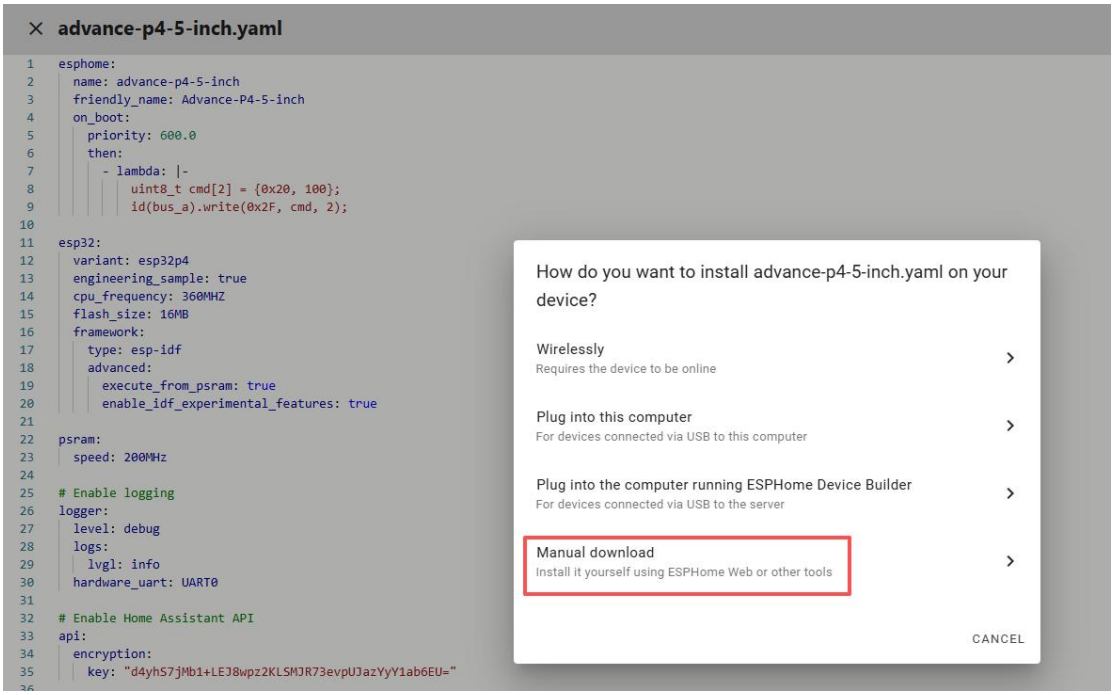
The entire process is automatically executed by a timer and does not require manual intervention. It achieves synchronous updates of sensor data and interface display, ensuring that the temperature and humidity information displayed to the user is always the latest. At the same time, by fully leveraging the control ID system of LVGL and the automation mechanism of ESPHome, the interface refresh is efficient, smooth, and easy to expand.

12.Upload the complete code

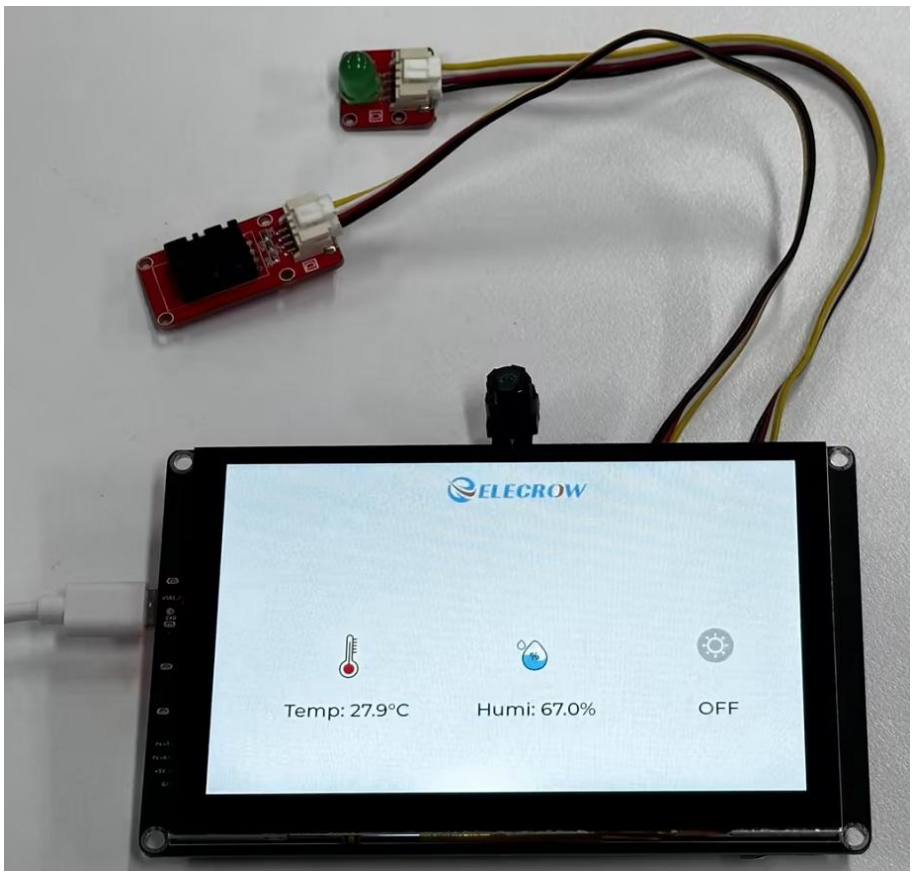
Now, all the codes have been prepared. Next, we will upload the codes.



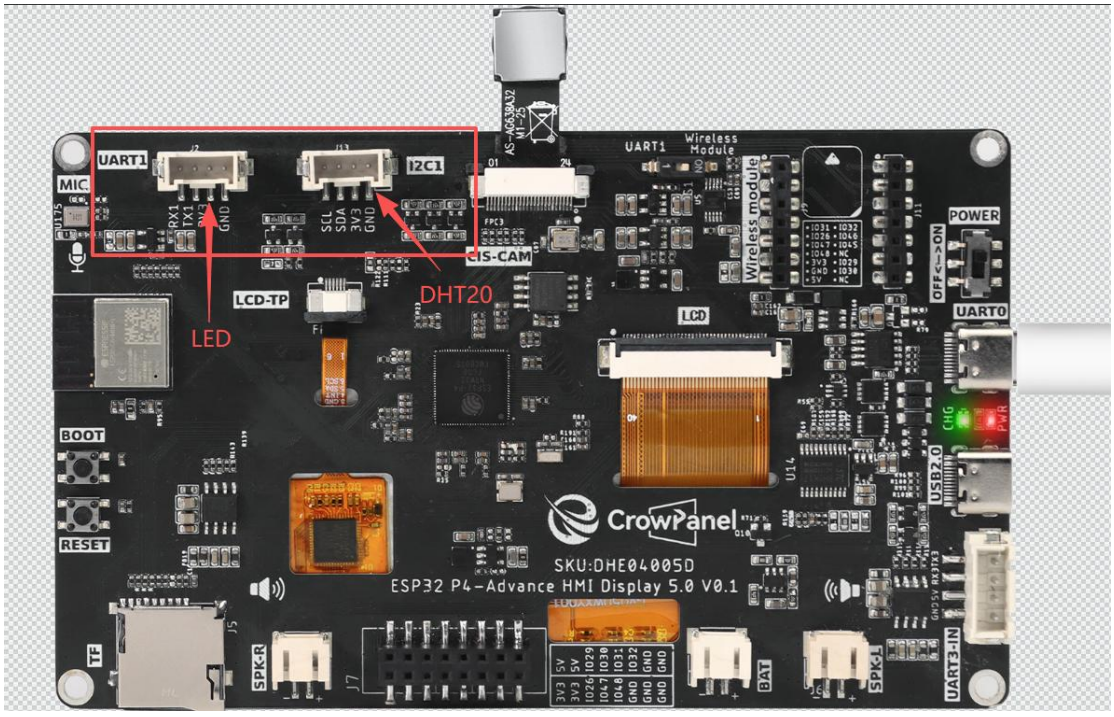
Then select "Manual download"



Following the previous steps for "8. First upload of code", the code was successfully uploaded. After uploading the code, you will be able to see the target interface displayed on the screen.



Since you have used the DHT20 temperature and humidity sensor and the LED light, you need to connect these two sensors on the back.

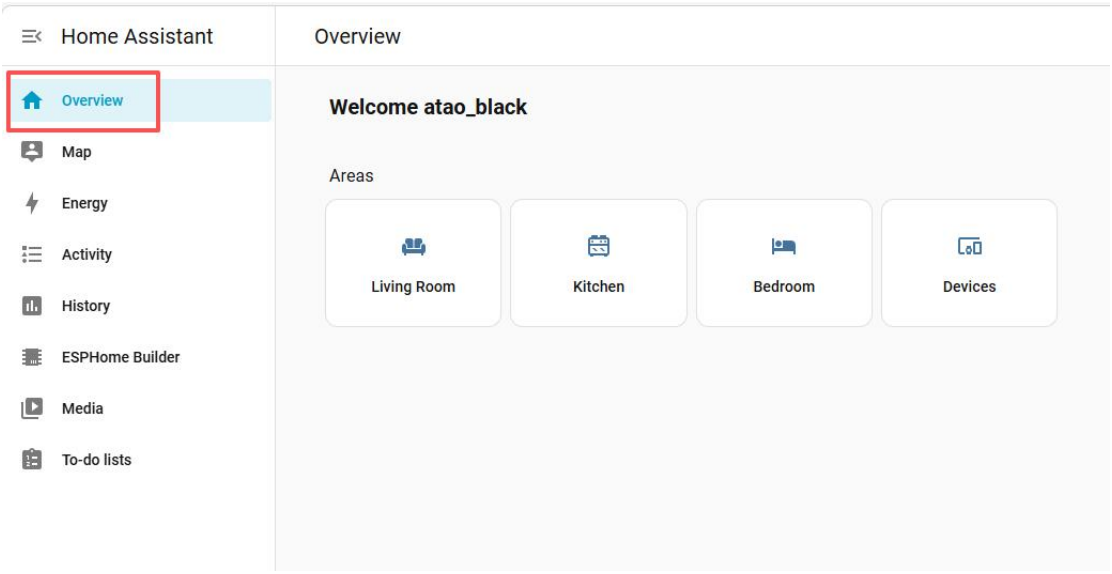


Then you will be able to see the real-time display of temperature and humidity data on the screen.

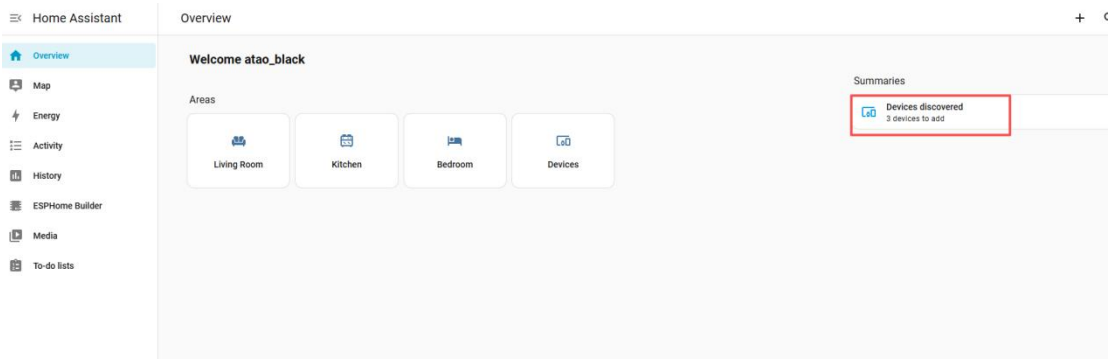
13.Remote observation and control

This is the local data we have observed. Next, let's take a look at how to remotely view the temperature and humidity data on the esphome platform and how to remotely control the LEDs.

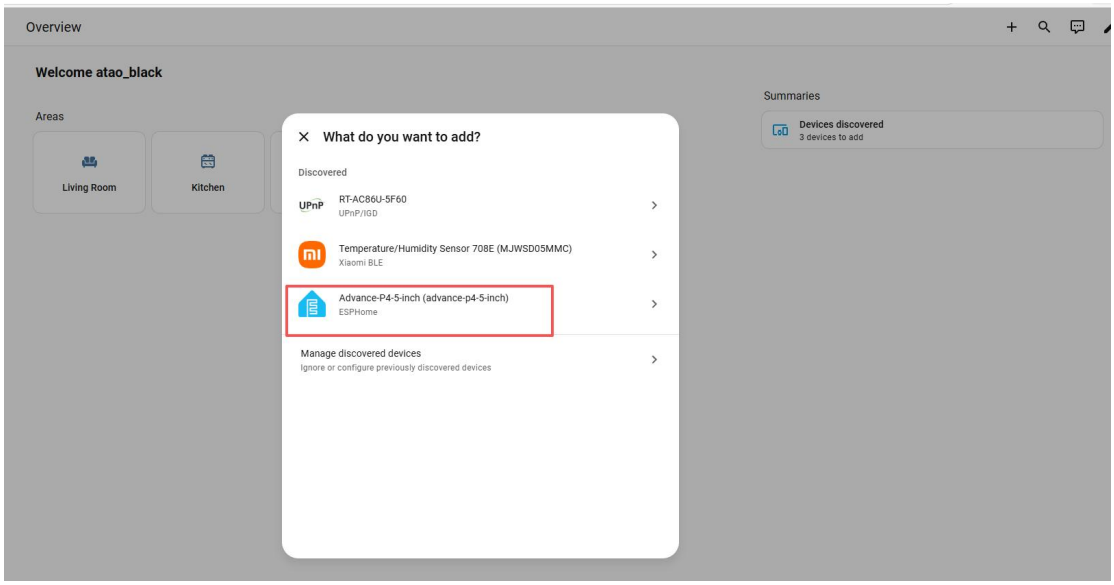
Let's return to the initial interface.



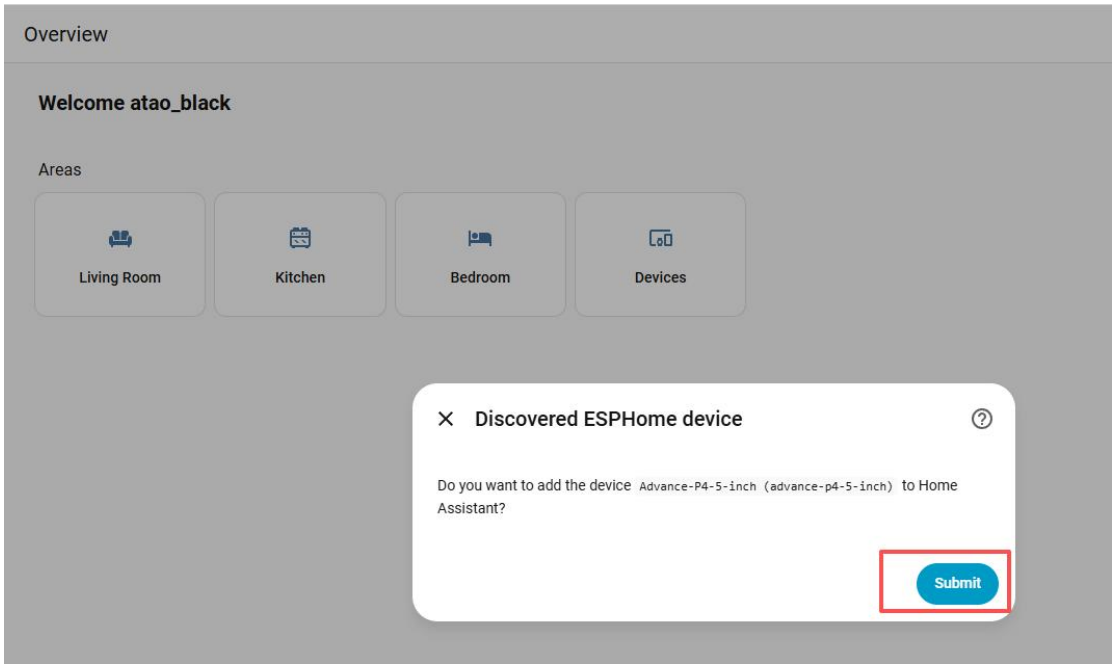
Once your code is uploaded successfully, you will be able to see your device information here.



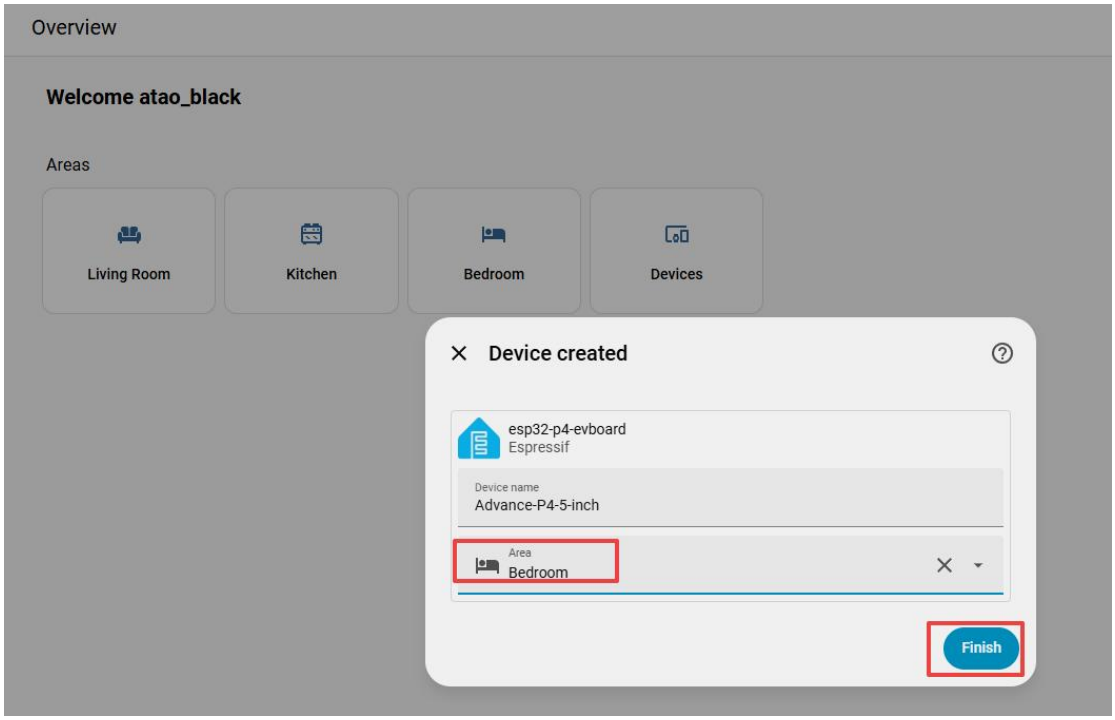
This is the system's scan of the devices nearby you.



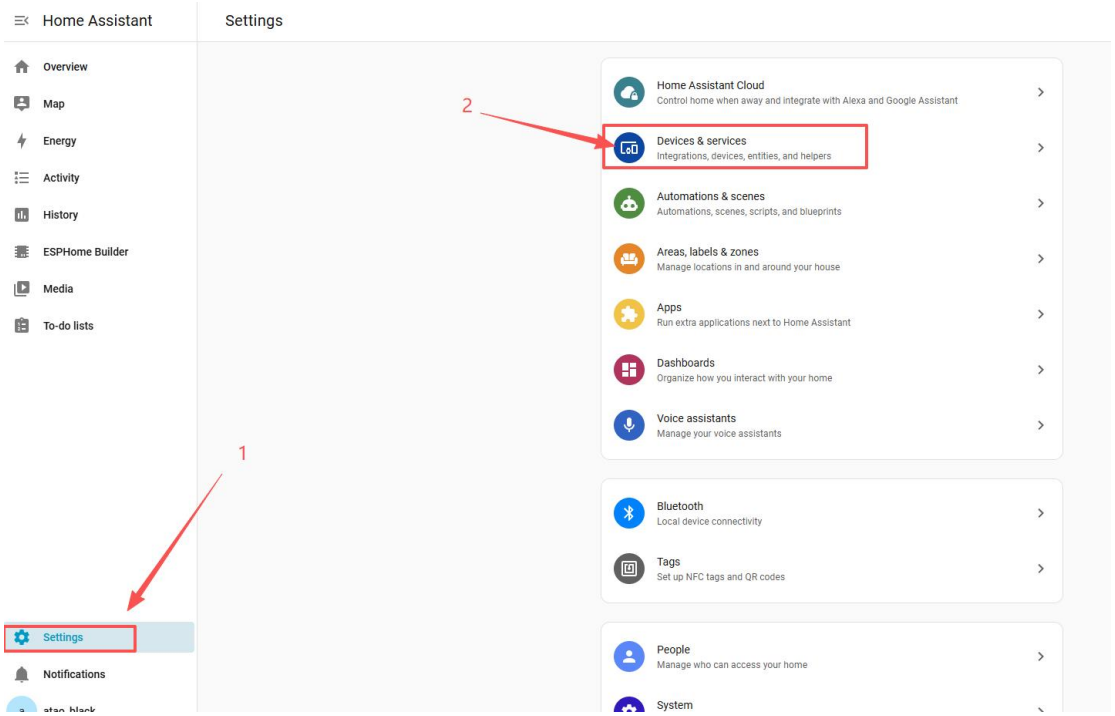
Click on our device and confirm the addition.



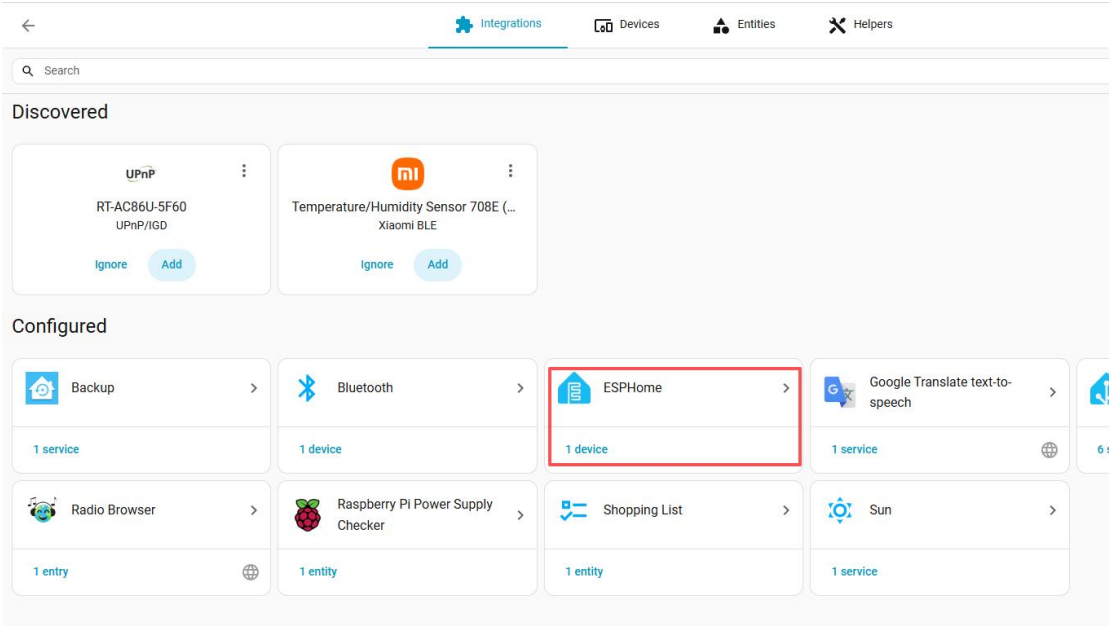
You can also create a zone for it. Here, I choose to place it in "bedroom".



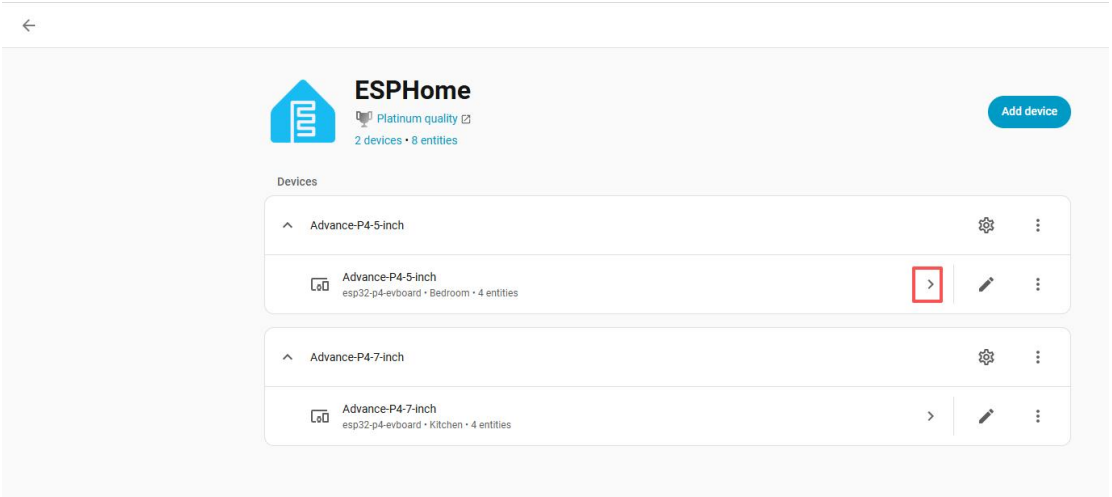
After completion, go to settings and select devices

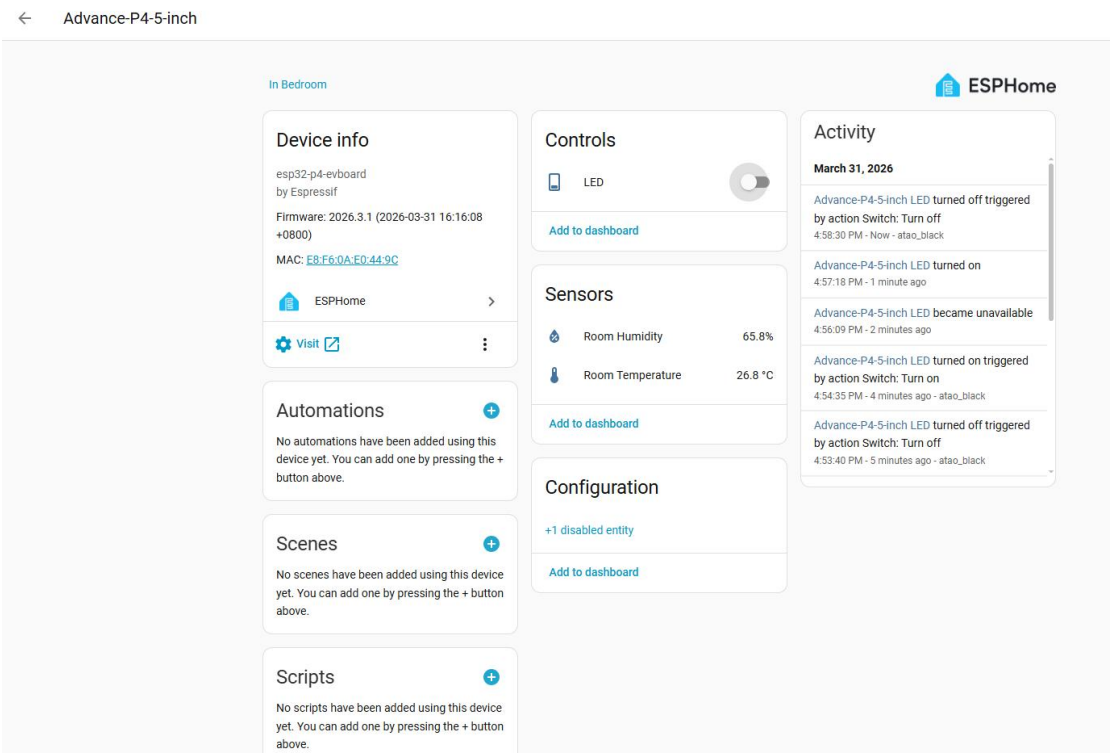


Choose ESPHome

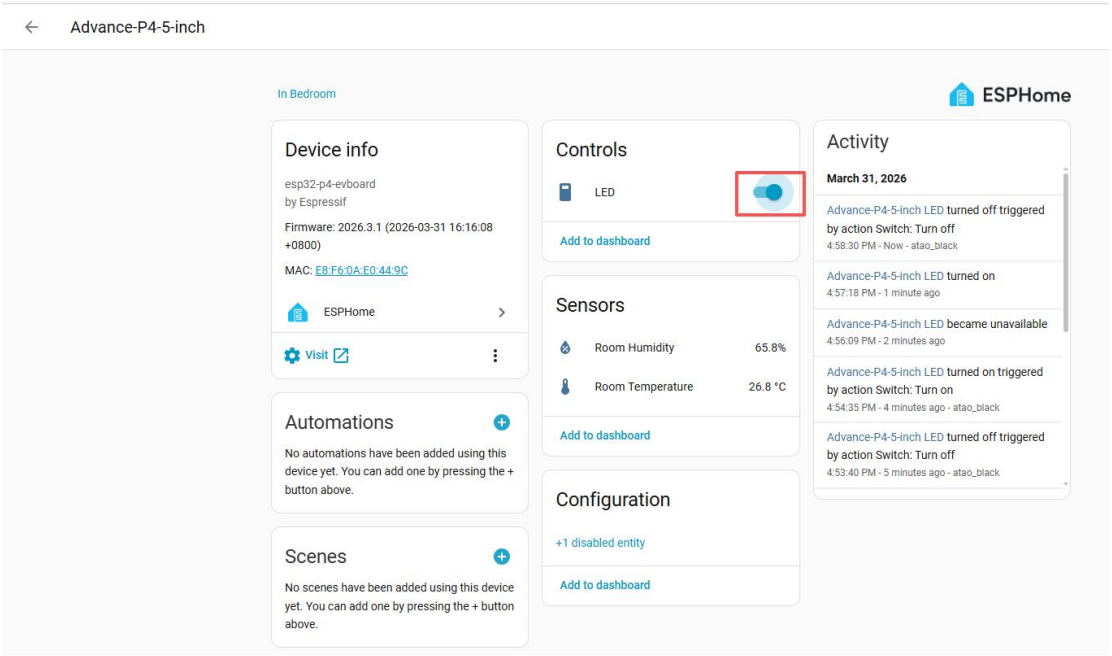


Here you can see the specific details of the equipment we just added.

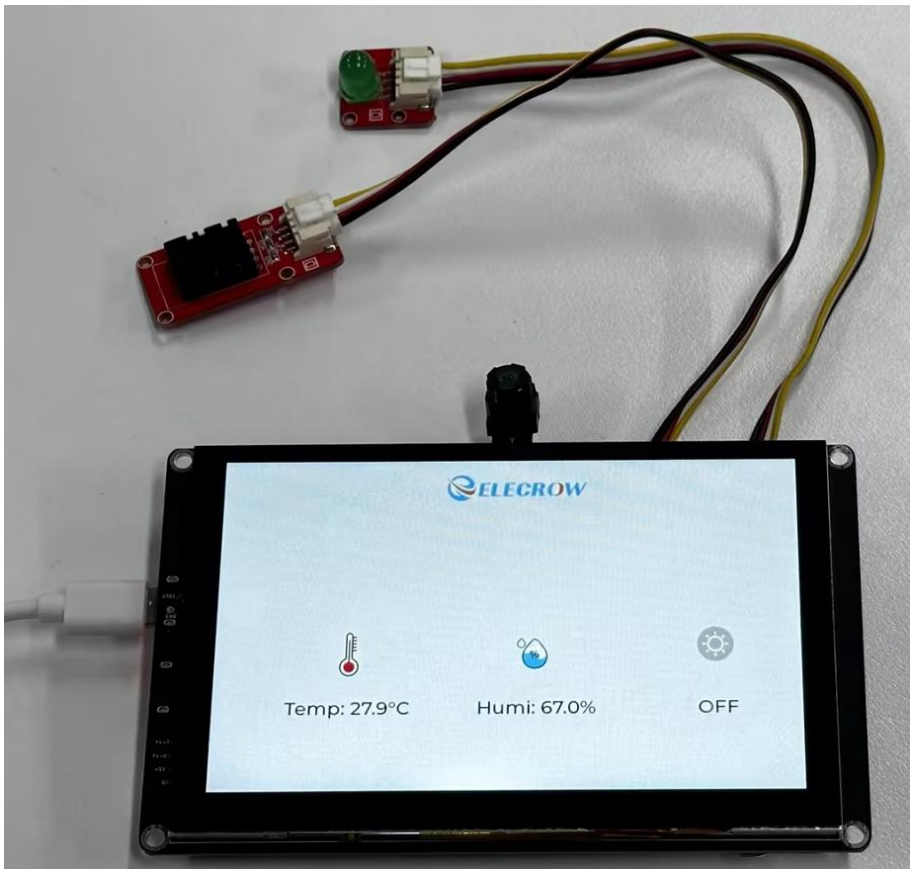




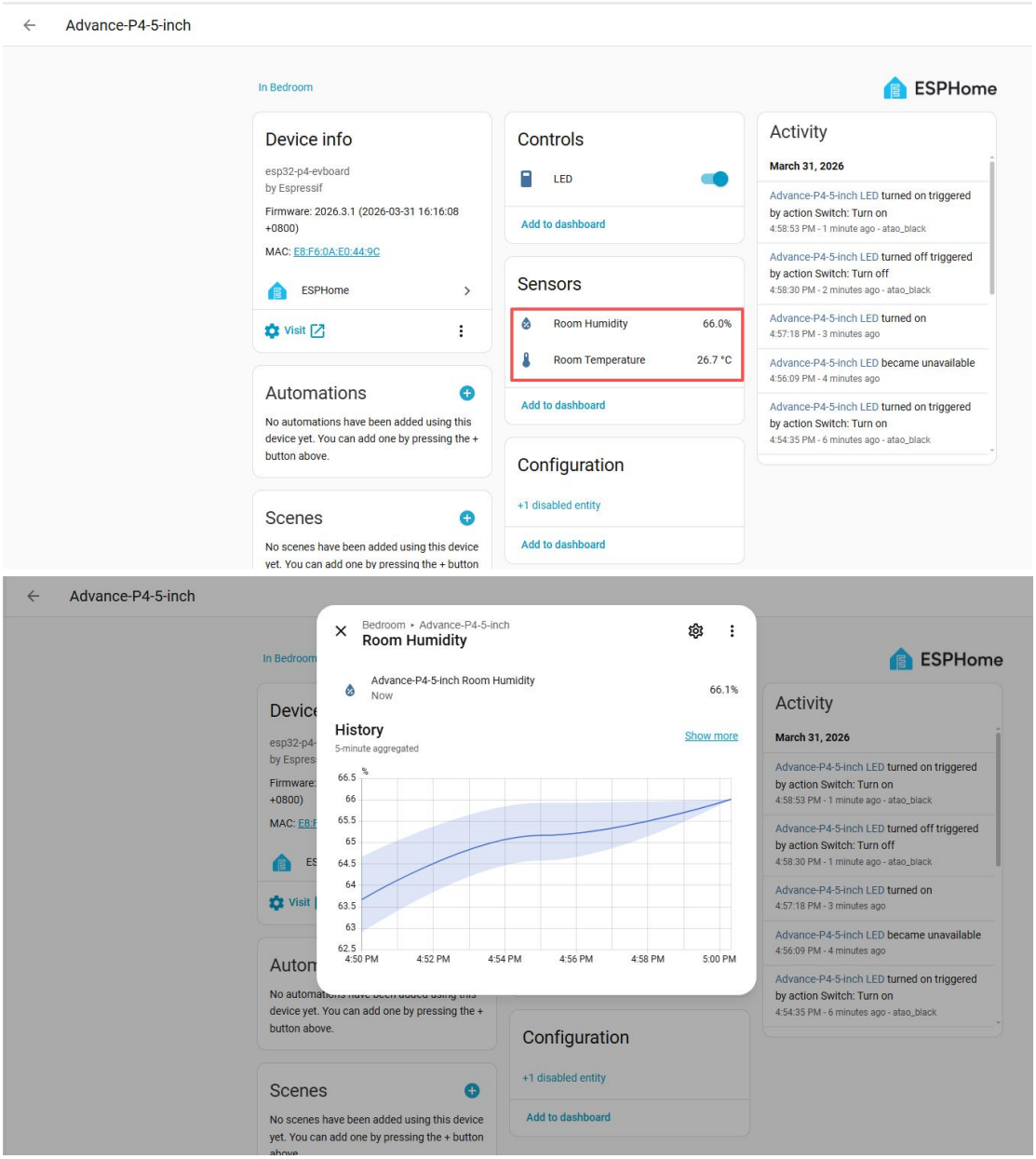
Next, when you click the switch of the LED, you will be able to see that the indicator light on the screen turns on, and the feedback from the LED is also quite obvious.

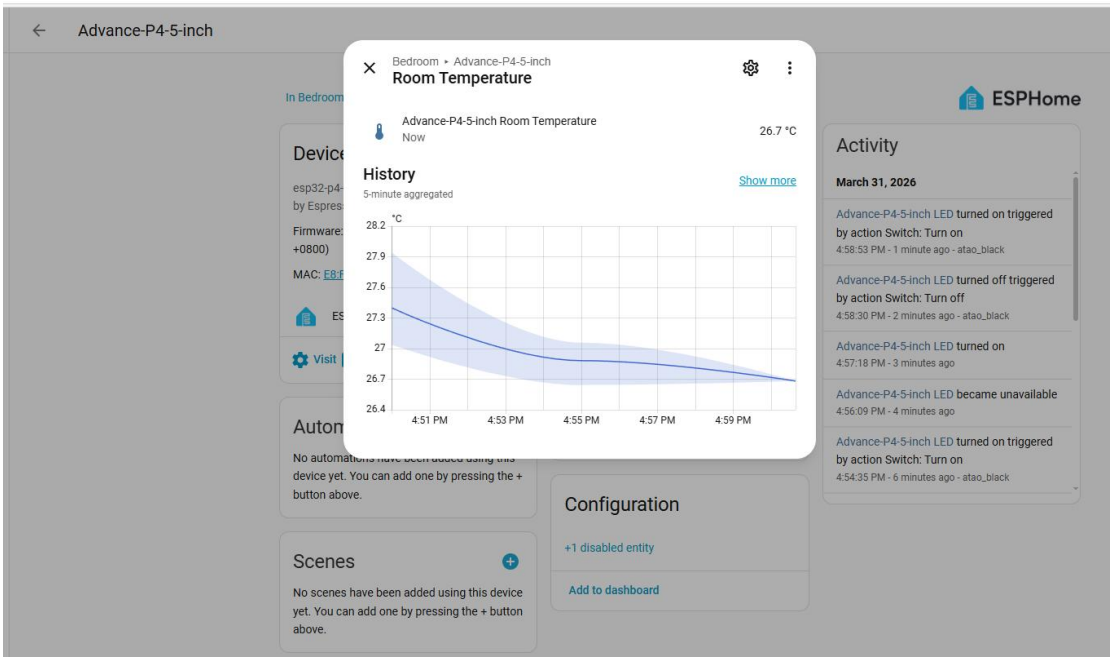


Next, when you click the switch of the LED, you will be able to see that the display light on the screen turns off, and the feedback from the LED is also quite obvious.

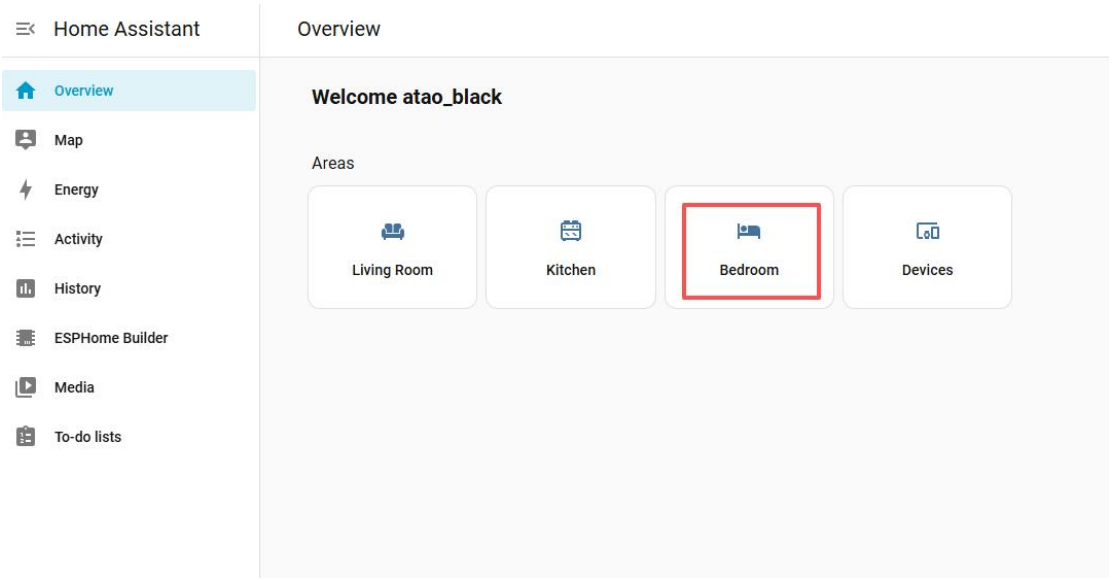


And you can also see the historical data of temperature and humidity collection from here.

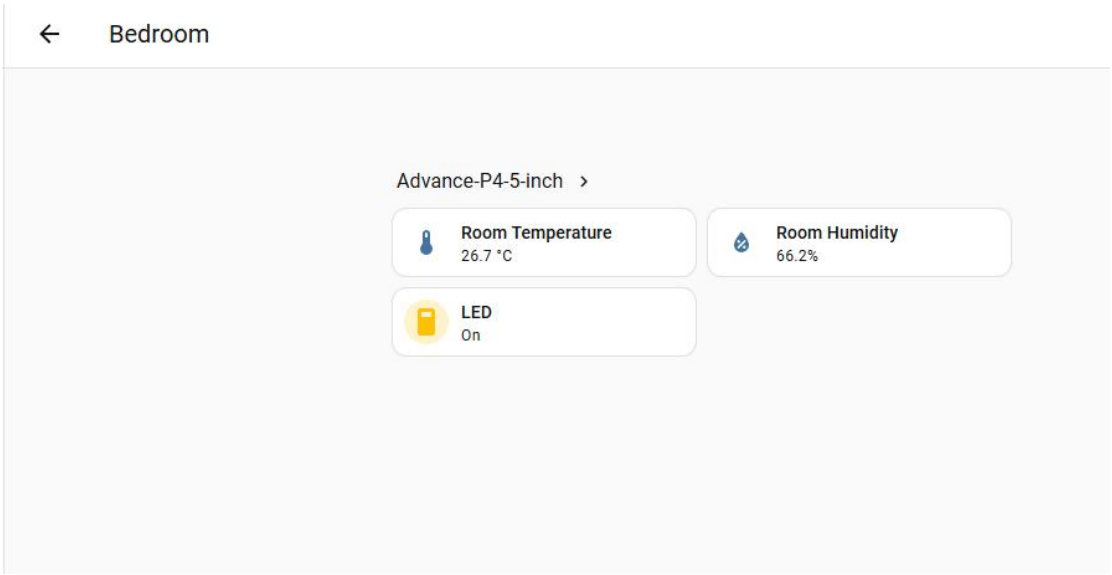


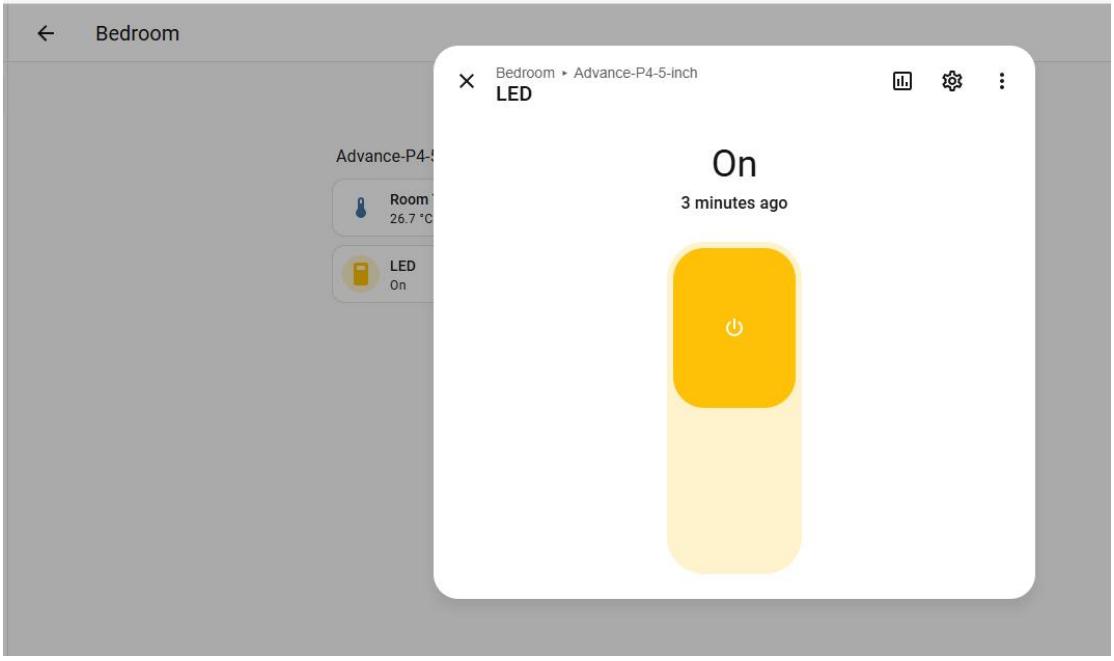


And then you go back to the main interface and arrive at the bedroom.



It can control the LED and also display the temperature and humidity data.


Bedroom
Advance-P4-5-inch
Room Temperature
26.7 °C
Room Humidity
66.2%
LED
On


Bedroom
Bedroom ▸ Advance-P4-5-inch
LED
On
3 minutes ago