Advance-P4_ESPHome-Tutorial

1.course introduction

In this class, we will teach you how to use and operate the CrowPanel Advanced ESP32-P4 HMI AI Display (7-inch / 9-inch / 10.1-inch) through ESPHome.

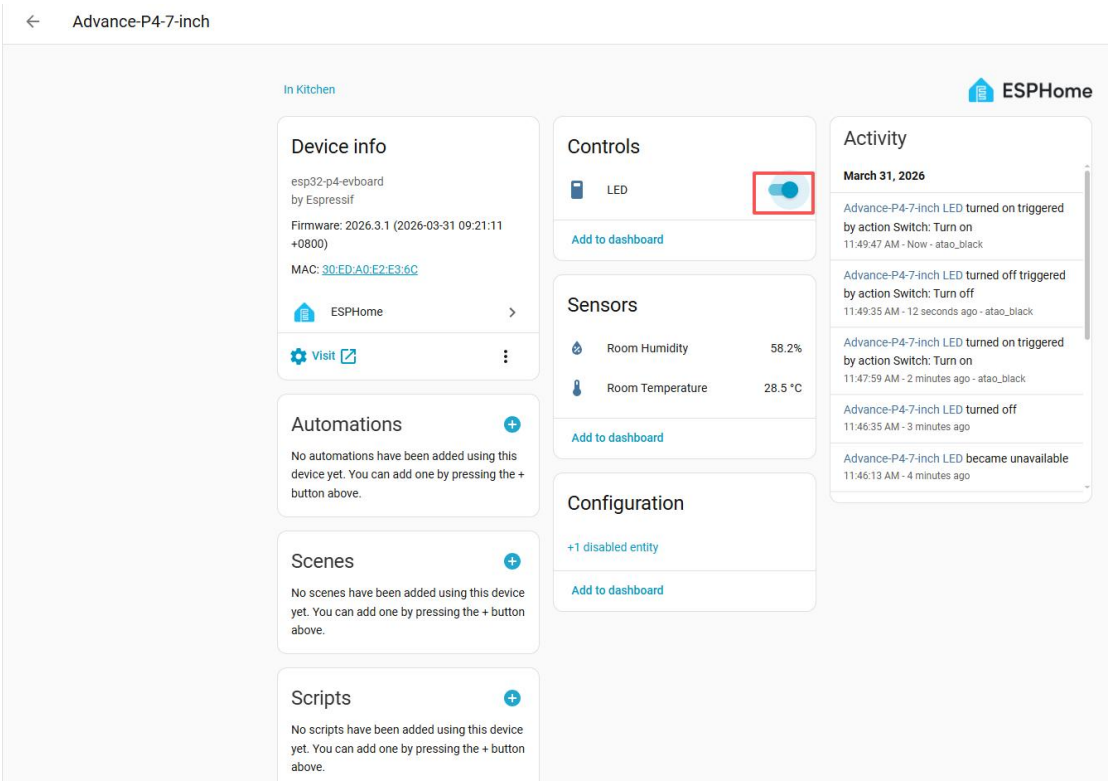
So let's move on to learn how to install the ESPHome environment, allowing you to edit code on it to achieve reading temperature and humidity data and remotely control the on and off of LEDs.

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.

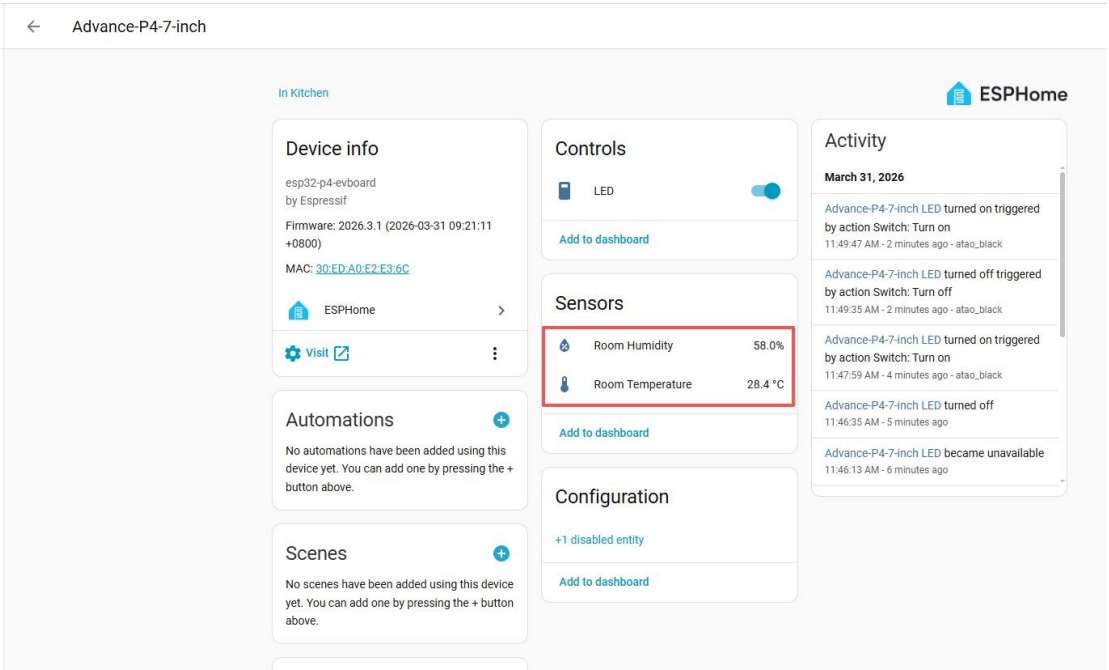
3.Project operation effect illustration

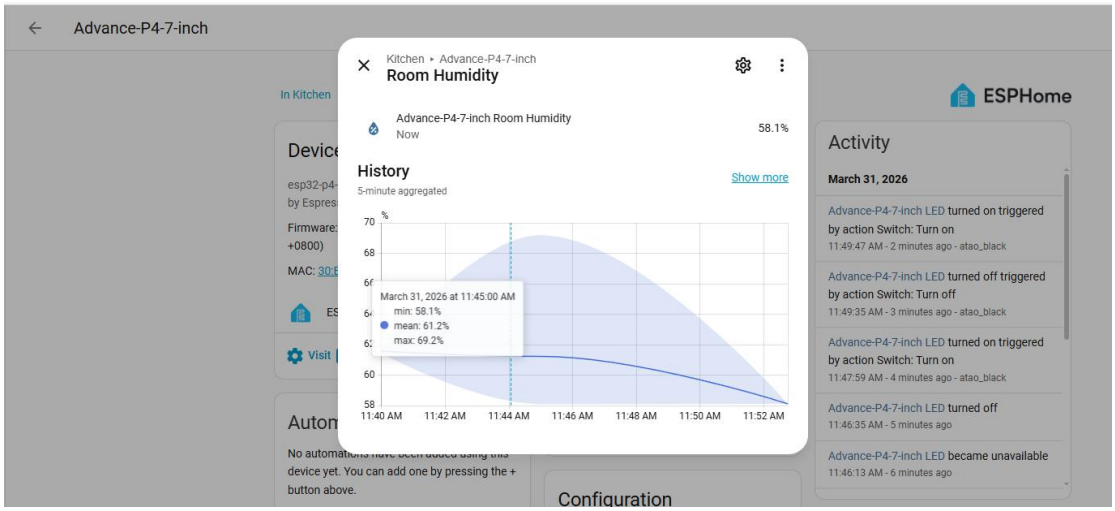
When you click the switch of the LED, you will be able to see that the indicator light on the screen is on, and the feedback from the LED will also be obvious.

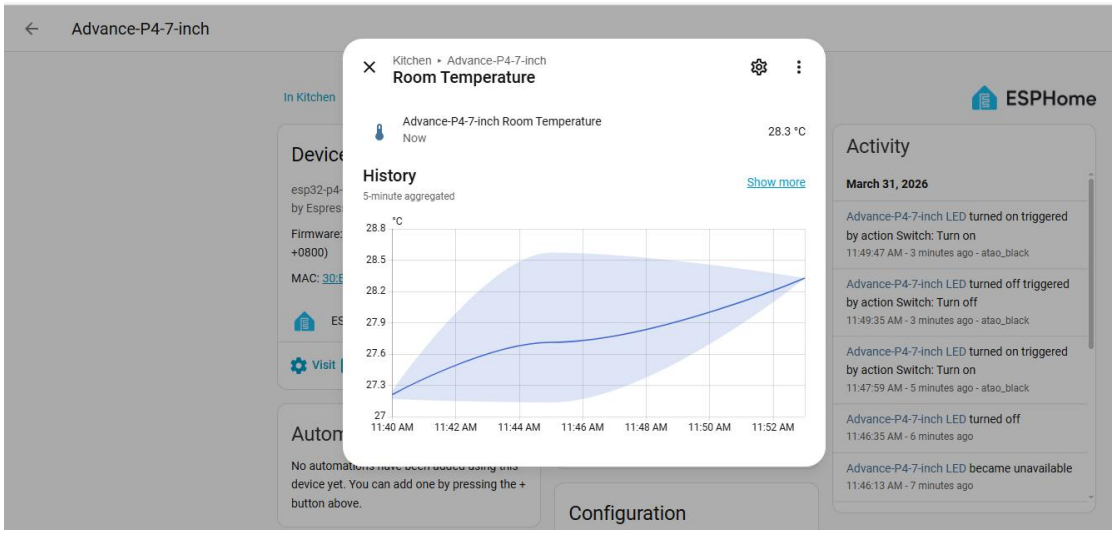


Next, when you click the switch of the LED, you will be able to see that the display light on the screen turns off, and the feedback from the LED is also quite obvious.

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.

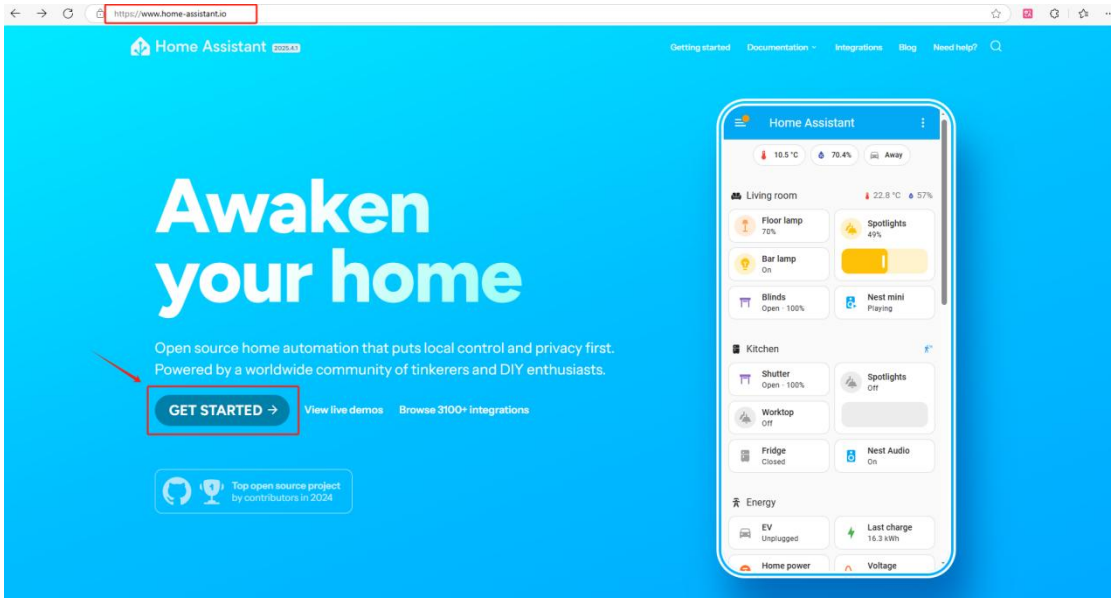
Together, Home Assistant and ESPHome turn your rotary screen into a flexible, customizable smart device: ESPHome gives the screen its "basic skills" (display, knob/touch response), and HA connects those skills to your entire smart home—so the screen can both control your devices and show you what's happening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

5.How to install software

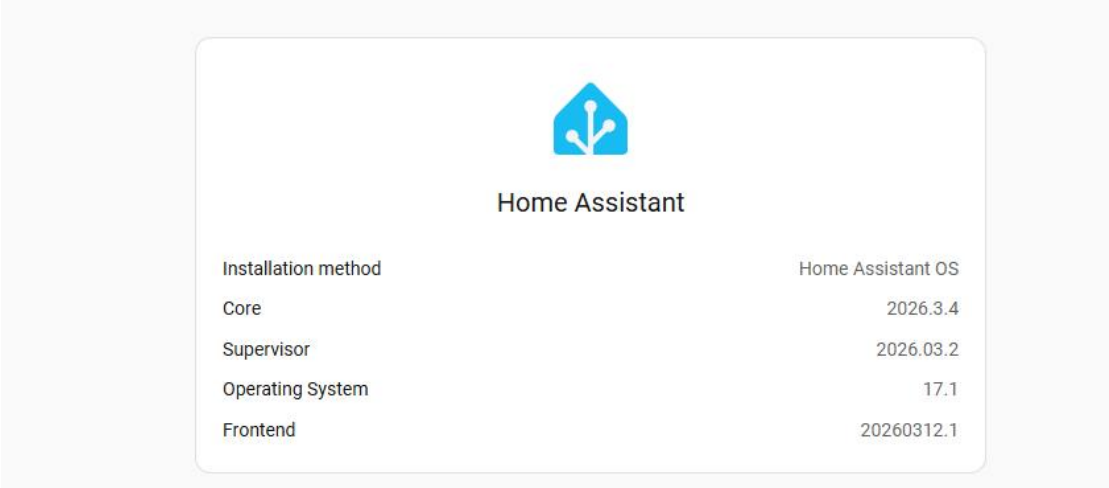
First, we need to prepare:
a Raspberry Pi 5;
a 64GB SD card;
a CrowPanel Advanced ESP32-P4 HMI AI Display（7inch / 9inch / 10.1inch）;
A humidity and temperature sensor;
An LED light;

Download HomeAssitant

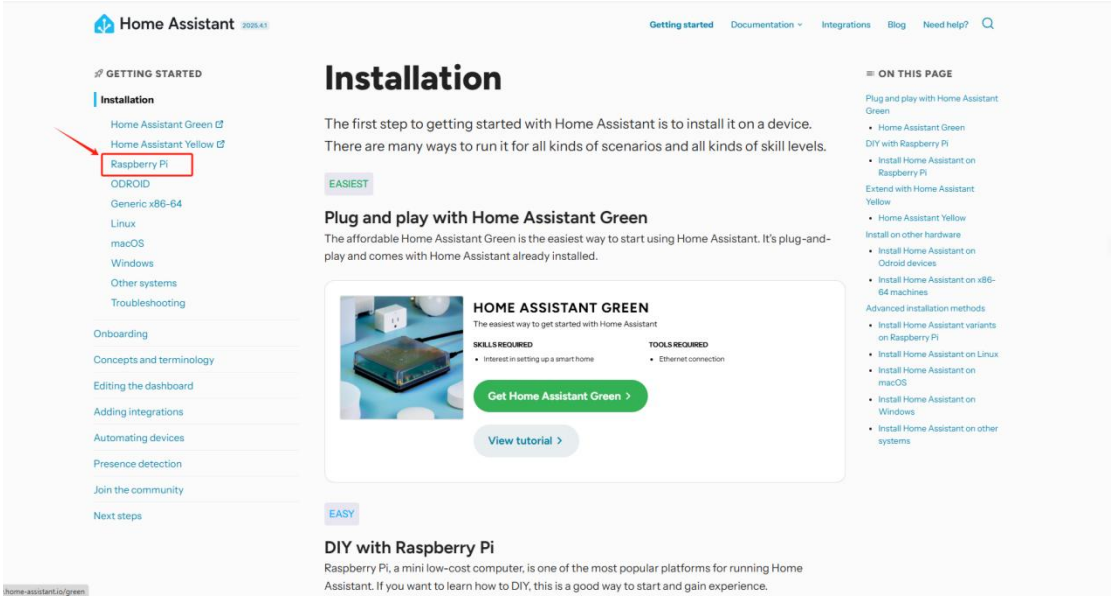
Open the official website of HomeAssitant：https://www.home-assistant.io/



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 ESP32-P4 HMI AI Display （7inch / 9inch / 10.1inch） 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:

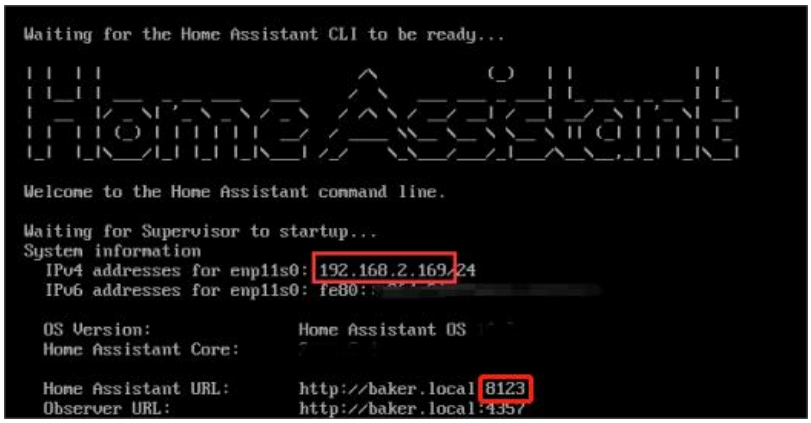
① Your computer

② CrowPanel Advanced ESP32-P4 HMI AI Display（7inch / 9inch / 10.1inch）

③ Raspberry Pi with the Home Assistant system

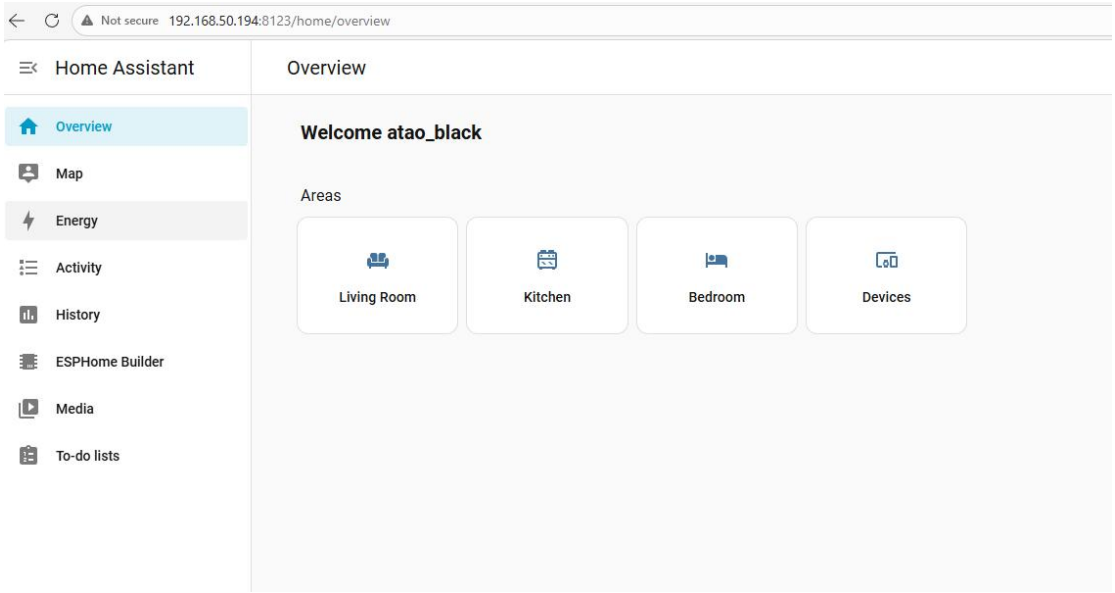
Note: The Raspberry Pi must be connected to a screen to view information. Once everything is ready, power on the Raspberry Pi and wait for it to load. The screen will display the following

image.
(You need to connect a display screen to the Raspberry Pi in order to see these information.)

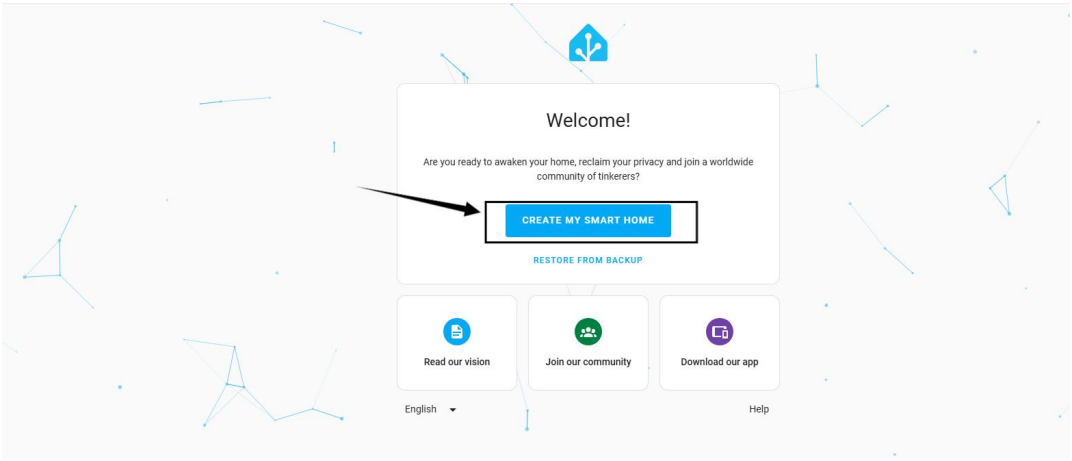


Remember the IP address and the Home Assistant URL.

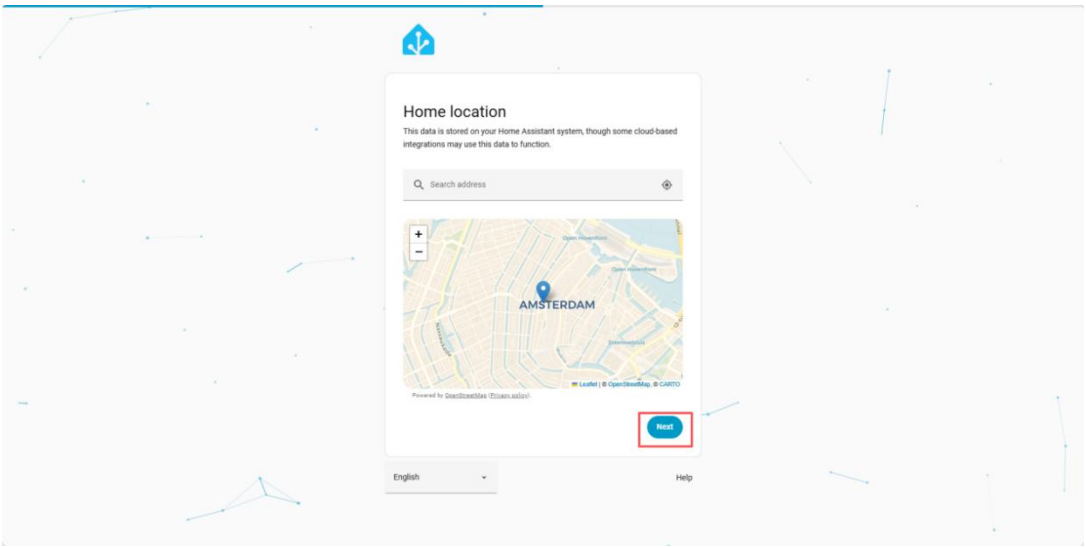
In my case, the IP address is: 192.168.50.194
So I enter **192.168.50.194:8123** in the browser.



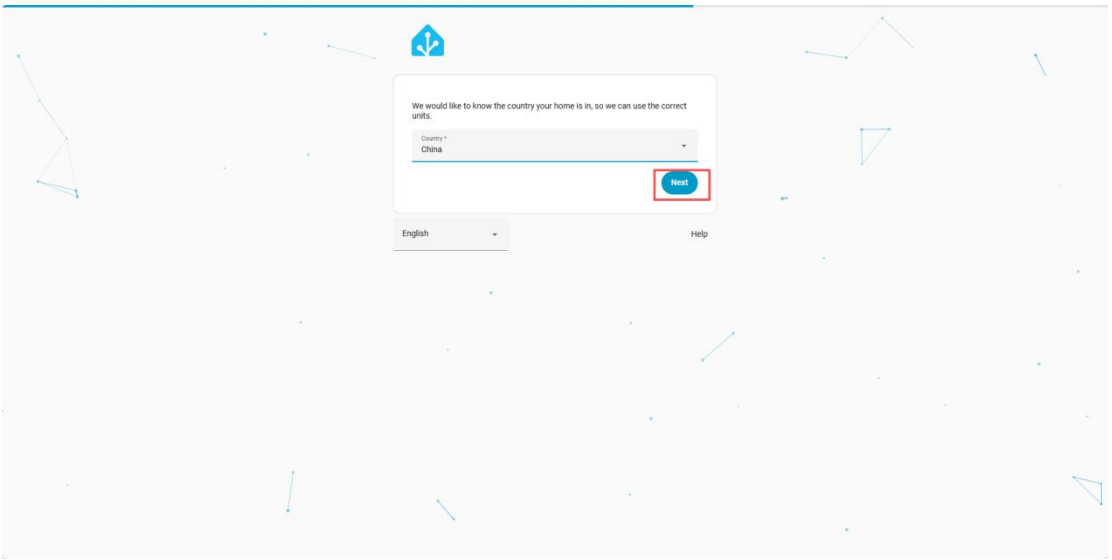
Next, start configuring the Home Assistant system.



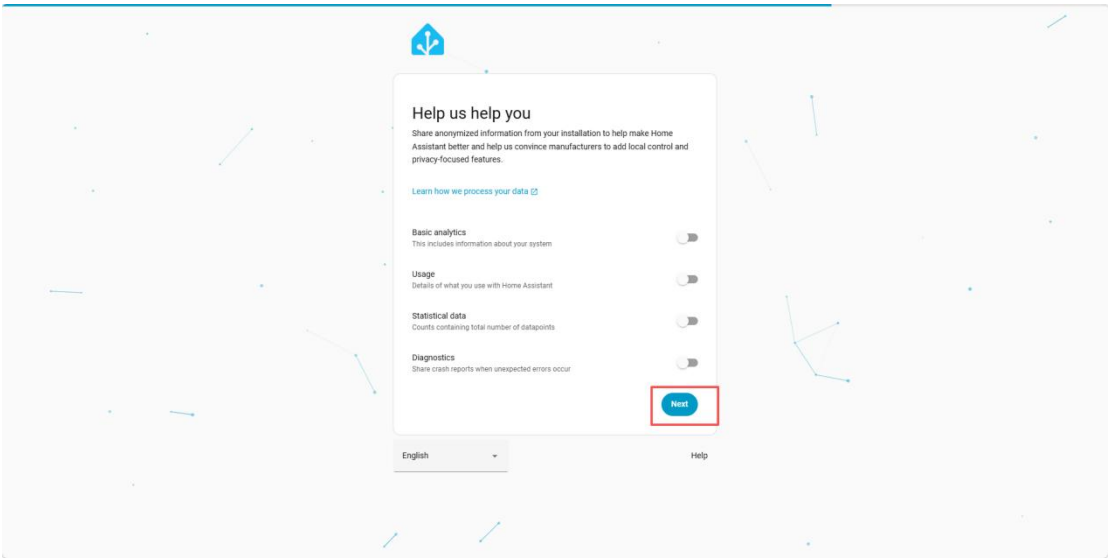
You can either manually set your location or allow it to be detected automatically.



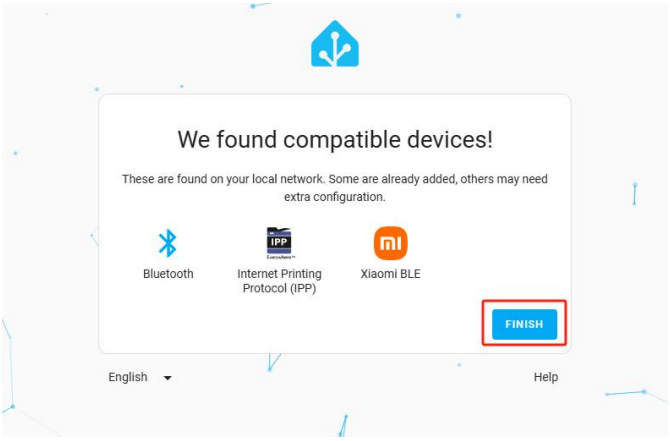
Set the country where you are located.



Keep it as default here.

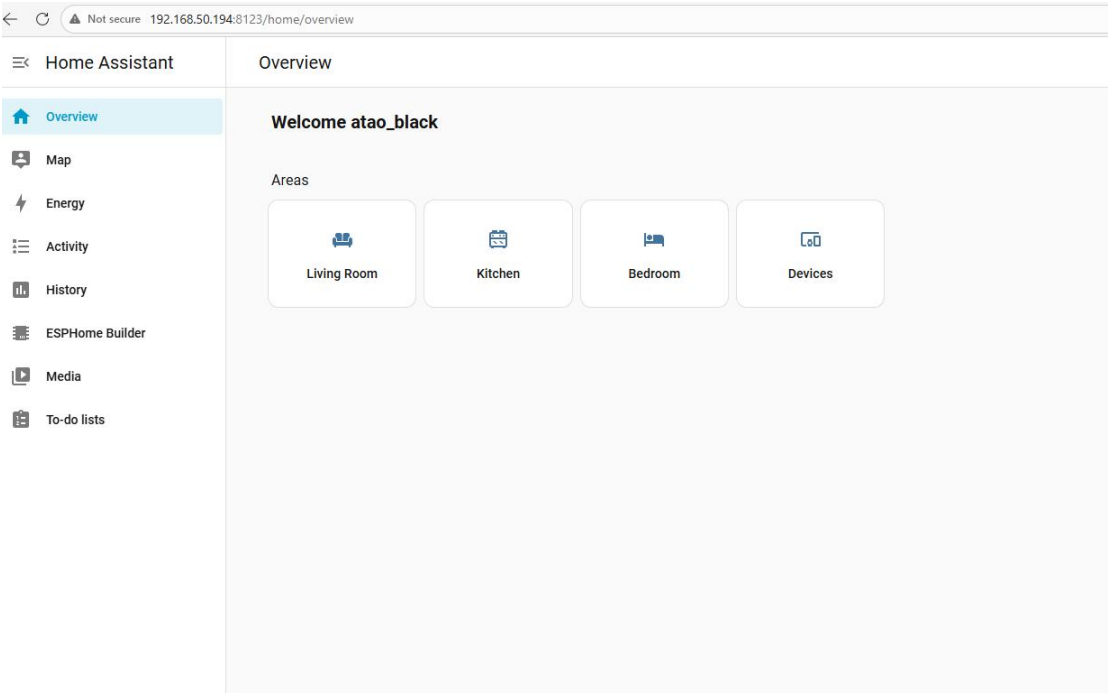


You can either add smart devices now or click **Finish** to add them later.
Here, we will add the devices later.

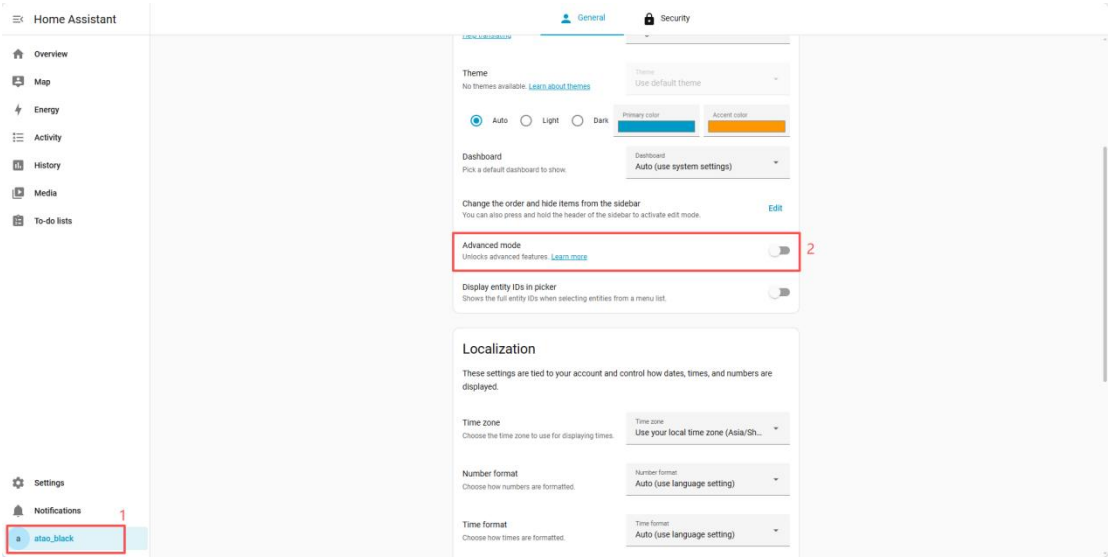


Add ESPHome

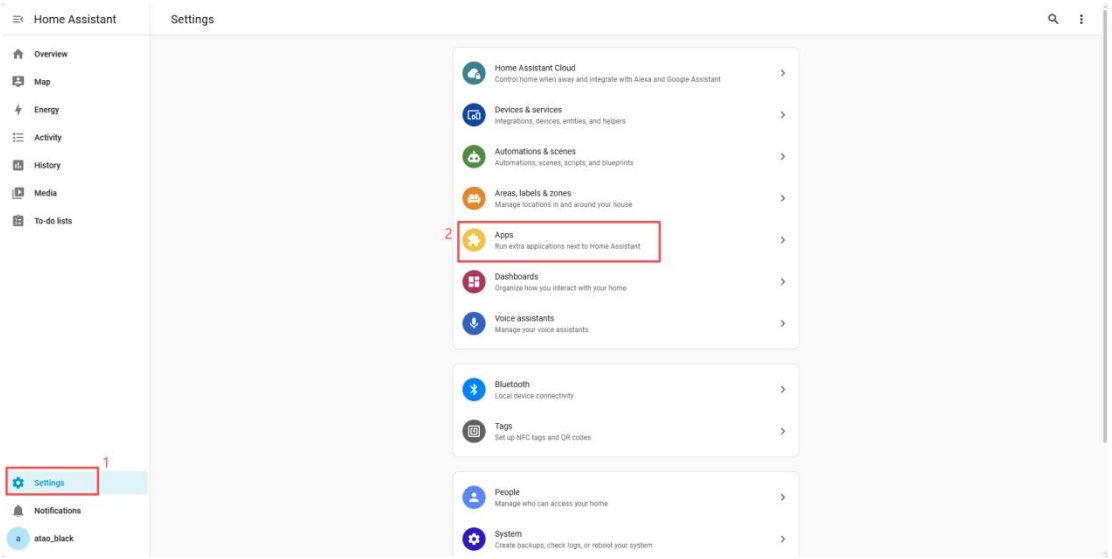
This will bring you to the main interface of Home Assistant.



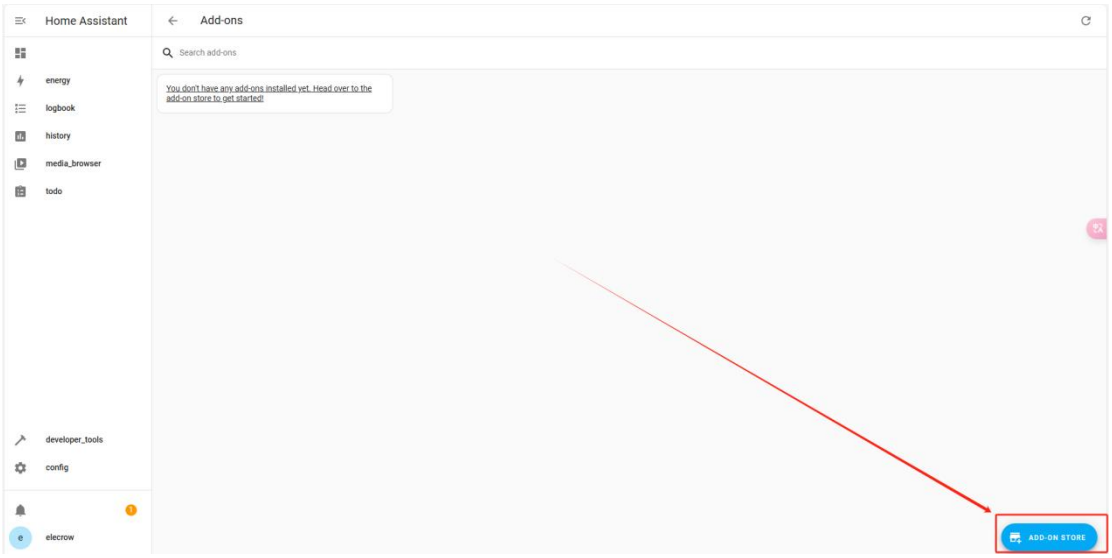
Click on the username and enable "Advanced Mode."

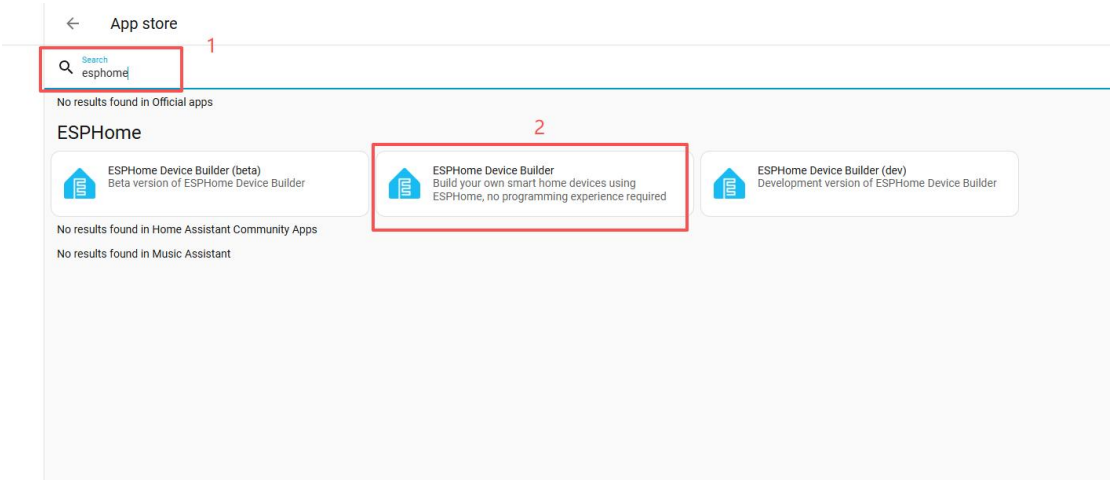


Click on "Settings" and then "Apps."

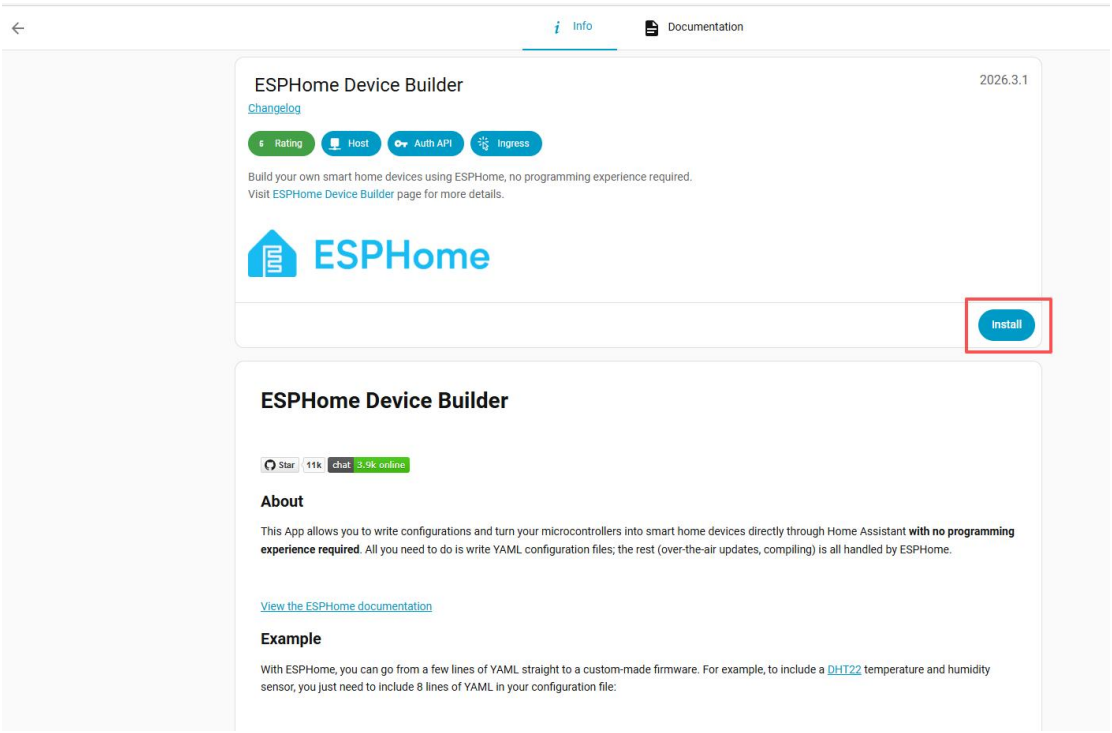


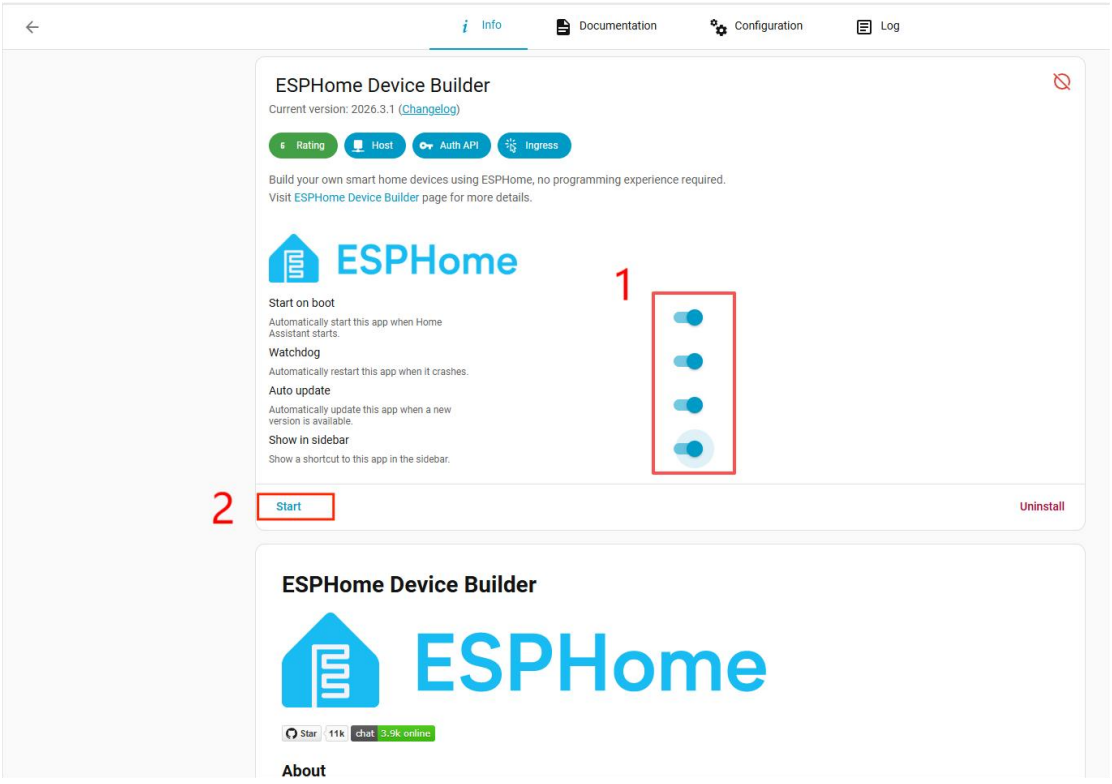
Click on "Apps" and type **ESPHome** to install it.





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



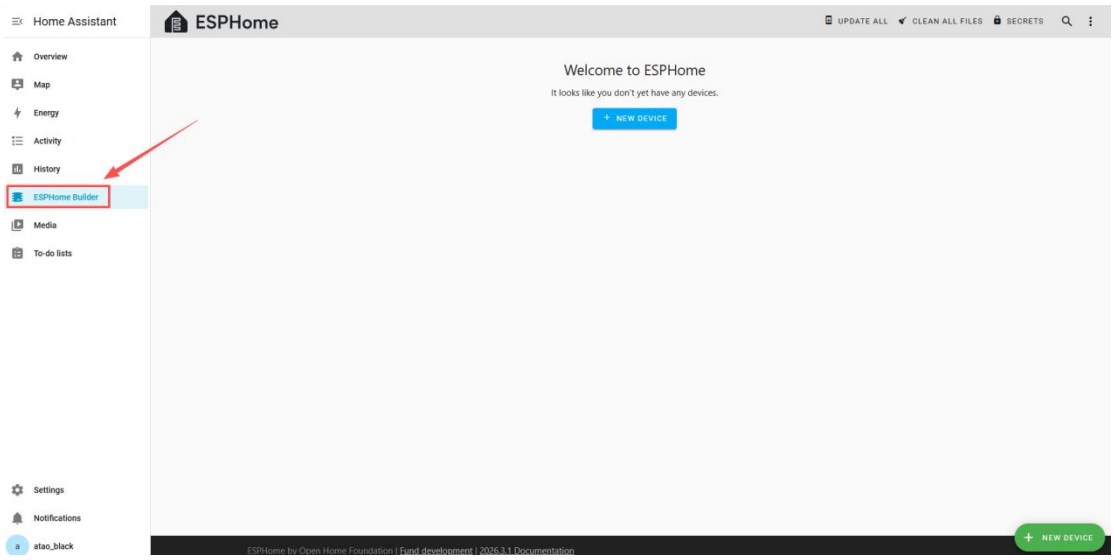


So we have successfully installed ESPHome. From now on, we can use ESPHome to edit the code and achieve the functions we want.

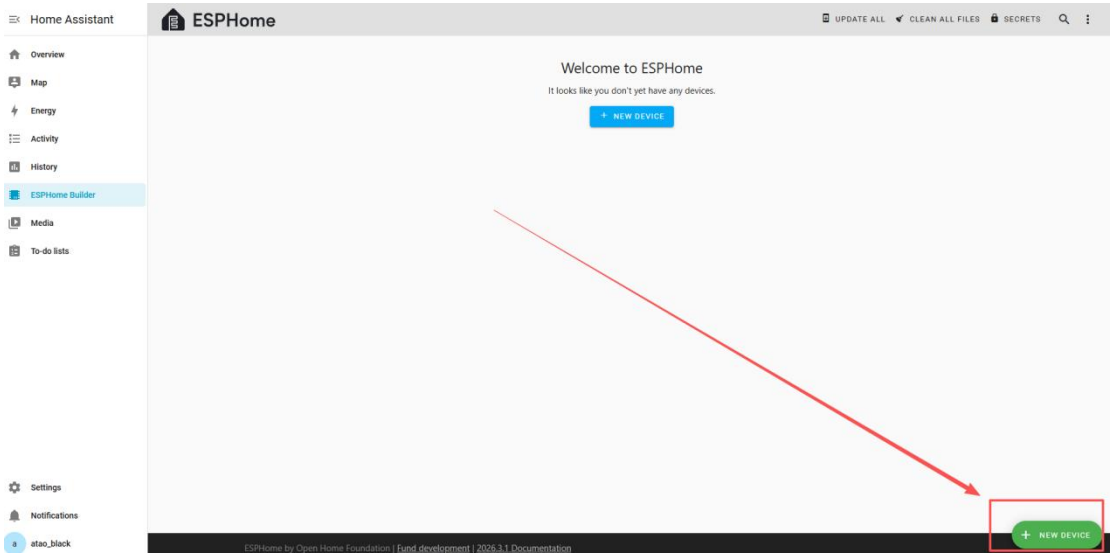
6. New Project

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

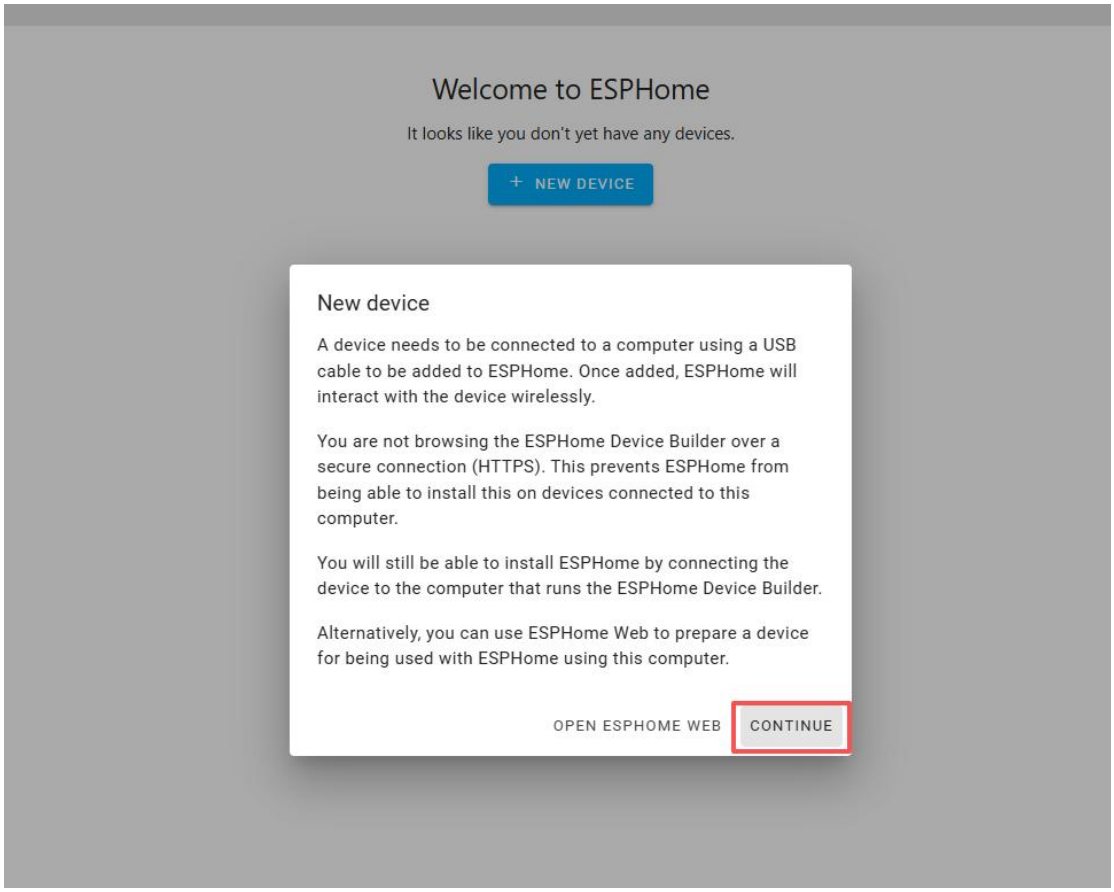
Come to ESPHome Builder



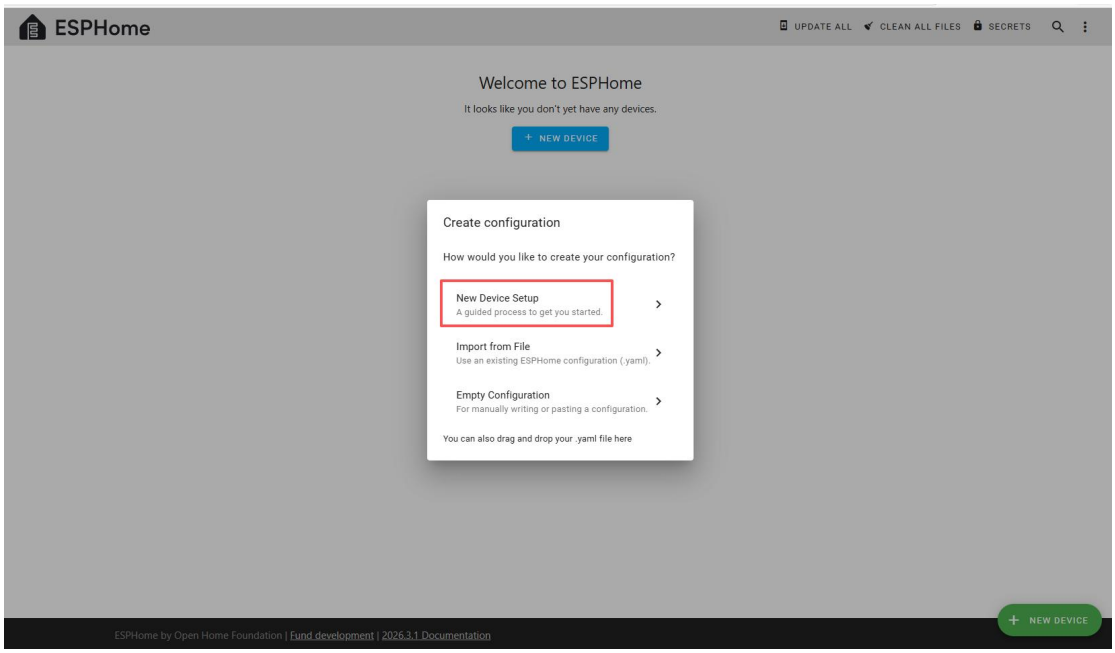
Click to add a new device



Click "Continue"

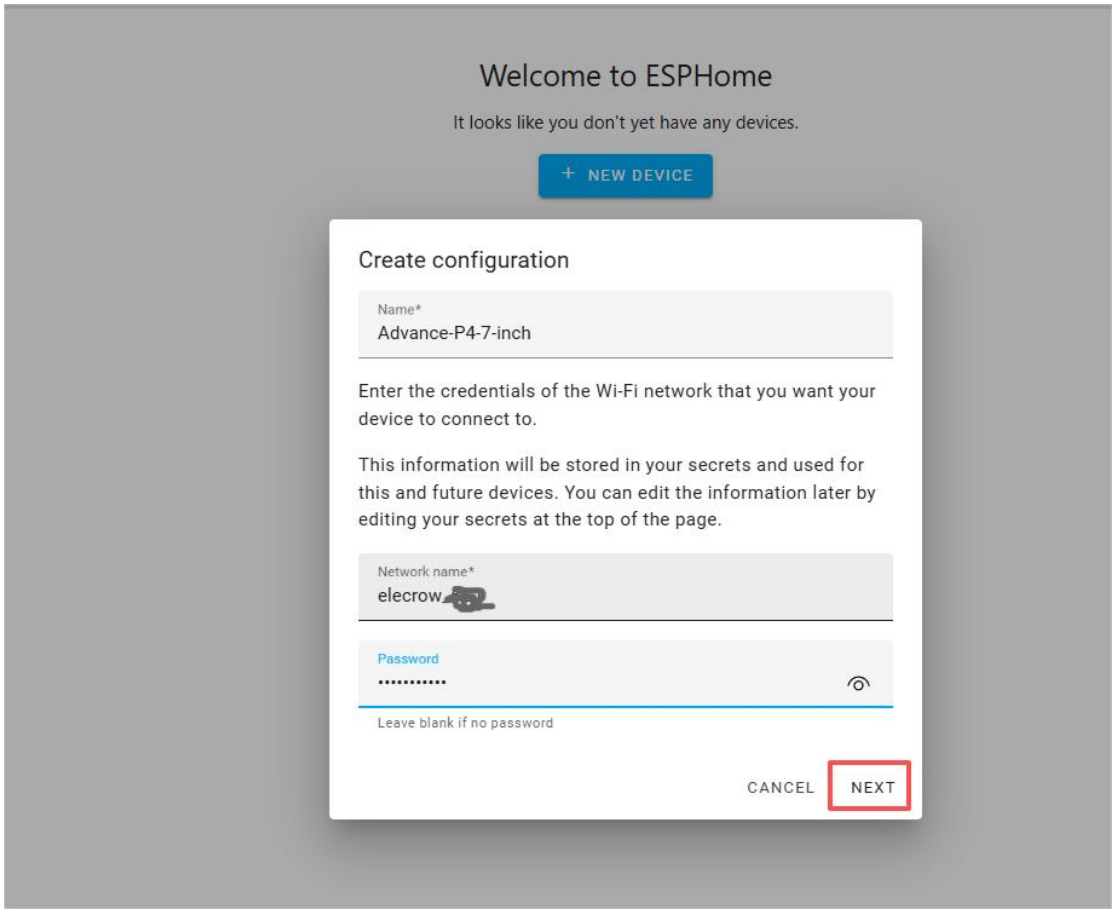


Add new equipment

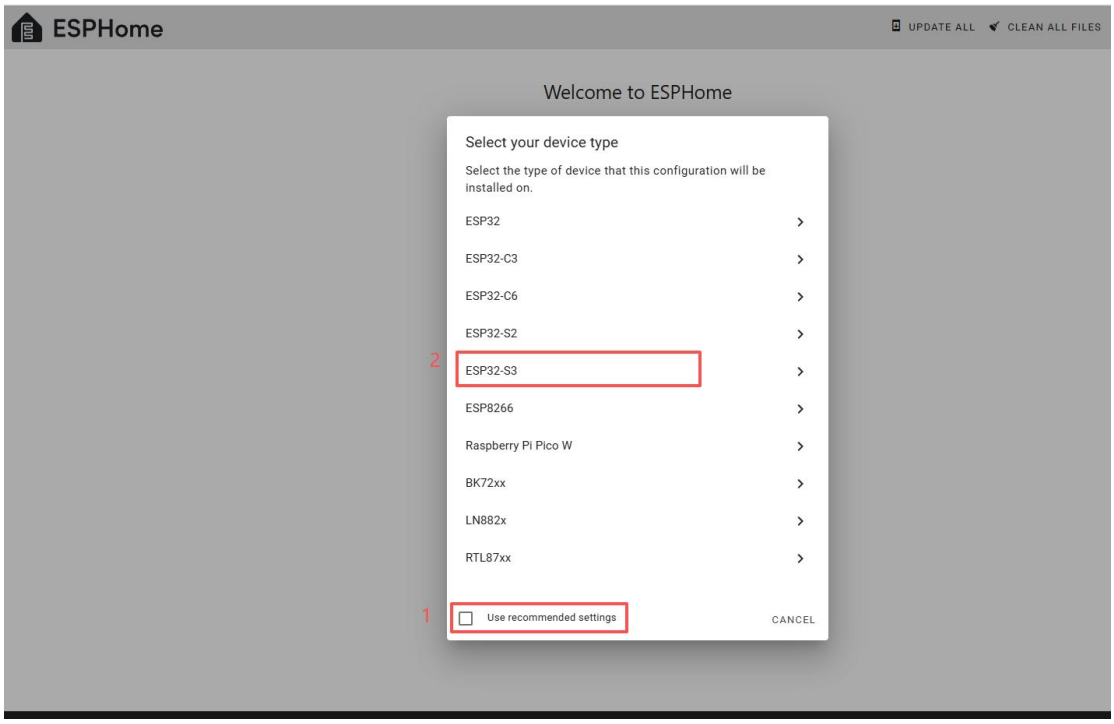


Set a name for this project, as well as the name and password of the Wi-Fi that you will have the device connect to.

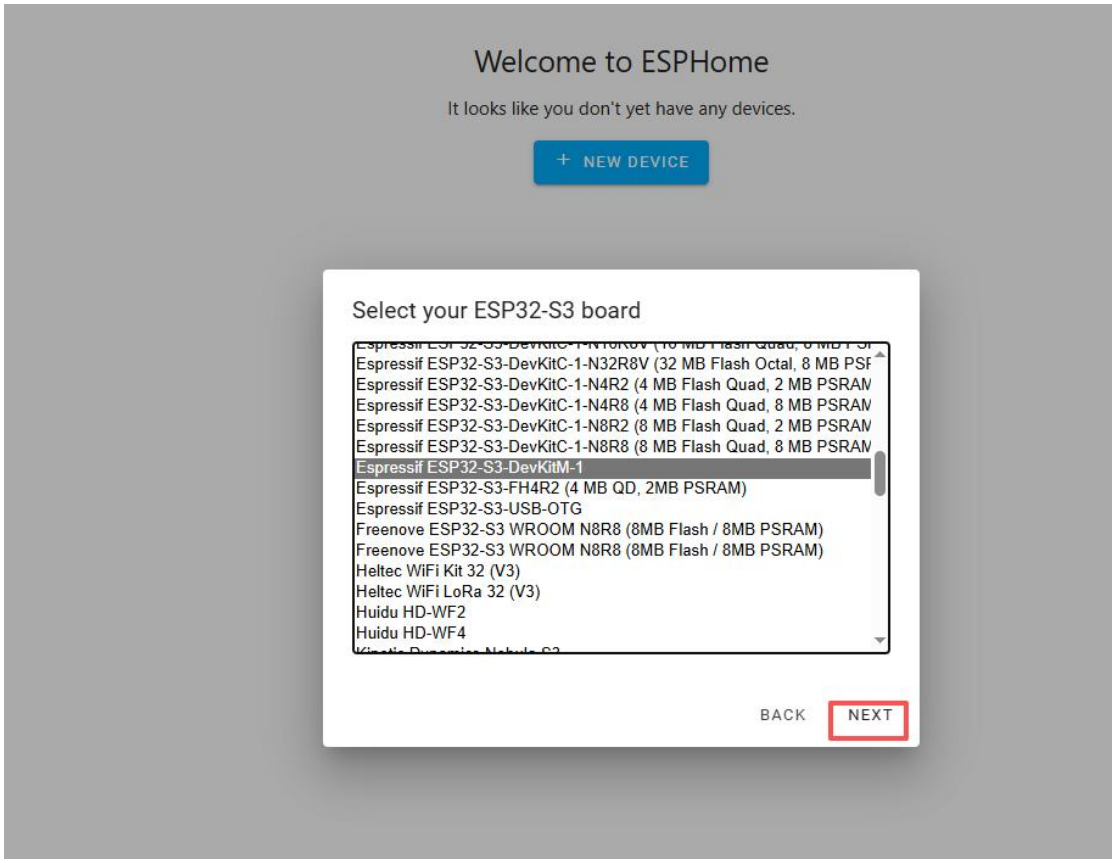
(This Wi-Fi must be in the same local network as your computer host, and make sure your Wi-Fi is on the 2.4GHz frequency band, as the ESP32-C6 Wi-Fi module used by ESP32-P4 operates on 2.4GHz.)



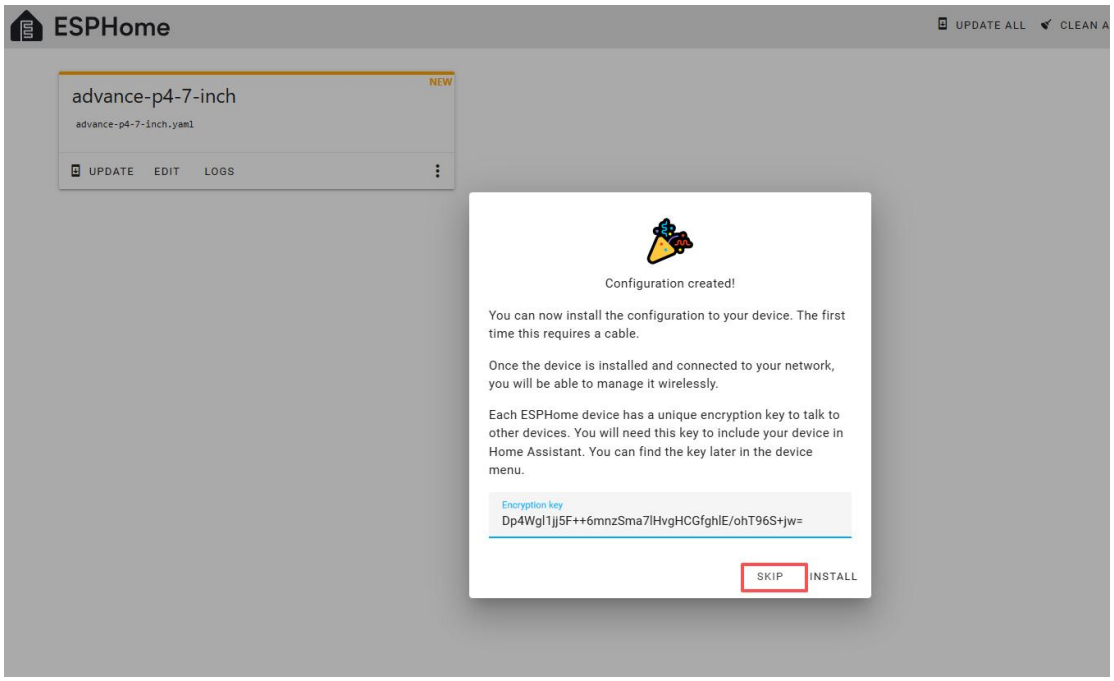
Here, do not check "Use recommended settings."
Here, we should have chosen the main control chip ESP32-P4 for the CrowPanel Advanced ESP32-P4 HMI AI Display (7inch / 9inch / 10.1inch), but currently the official has not provided the interface yet. So for now, we'll just randomly pick one, and later we 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).

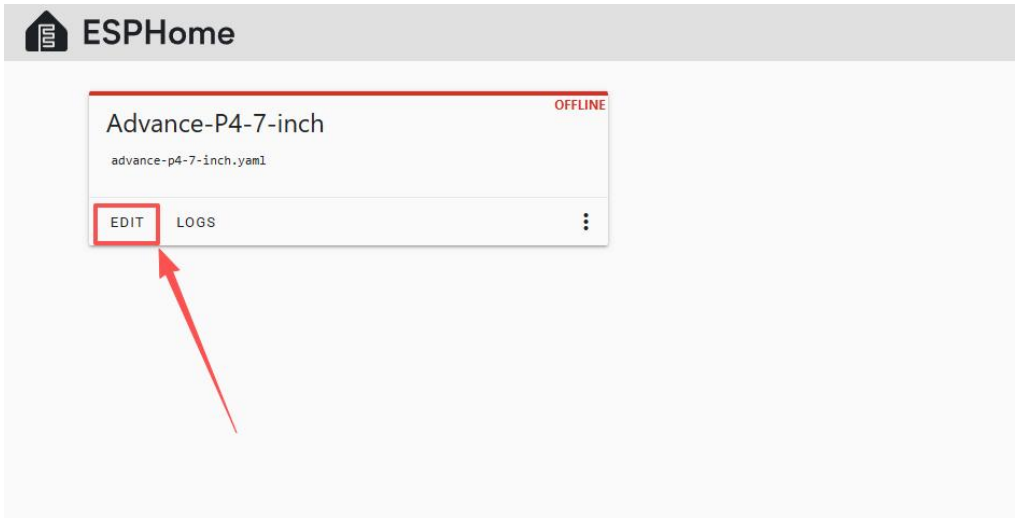


Here, click "SKIP".



7. Configuration Code

Then, return to the main interface, find the Advance_P4_7_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-p4-7-inch.yaml

```yaml
esphome:
  name: advance-p4-7-inch
  friendly_name: Advance-P4-7-inch

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

# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "Dp4Wgl1jj5F++6mnzSma7lHvgHCGfghlE/ohT96S+jw="

ota:
  - platform: esphome
    password: "bf1c7324cdcd0f41044c7722cbf5a048"

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:

```

You can click here to download the code in this course, which will help you achieve the related functions.(7-inch / 9-inch / 10.1-inch All three sizes are compatible.)

https://github.com/Elecrow-RD/CrowPanel-Advanced-7inch-ESP32-P4-HMI-AI-Display-1024x600-IPS-Touch-Screen/tree/master/example/V1.0/ESPHome/Code

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

advance-p4-7-inch.yaml

```yaml
esphome:
  name: advance-p4-7-inch
  friendly_name: Advance-P4-7-inch

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

# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "Dp4Wgl1jj5F++6mnzSma7lHvgHCGfghlE/ohT96S+jw="

ota:
  - platform: esphome
    password: "bf1c7324cdcd0f41044c7722cbf5a048"

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:

```

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-p4-7-inch.yaml

```yaml
esphome:
  name: advance-p4-7-inch
  friendly_name: Advance-P4-7-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:

# Enable Home Assistant API
api:
  encryption:
    key: "Dp4Wgl1jj5F++6mnzSma7lHvgHCGfghlE/ohT96S+jw="

ota:
  - platform: esphome
    password: "bf1c7324cdcd0f41044c7722cbf5a048"
```

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.

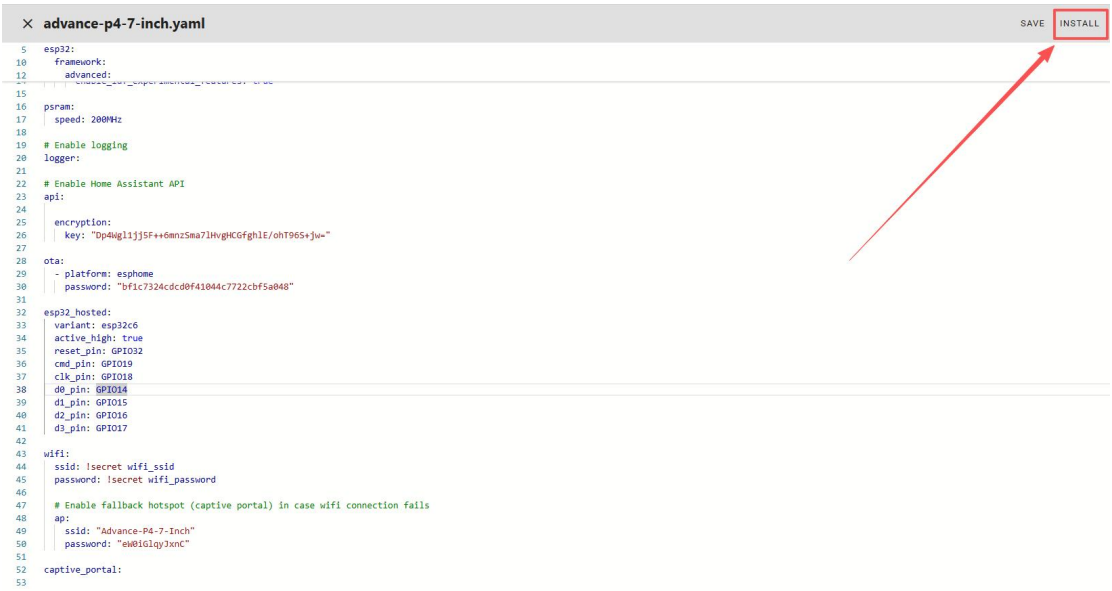
Remember to replace your own Wi-Fi name and password.

advance-p4-7-inch.yaml

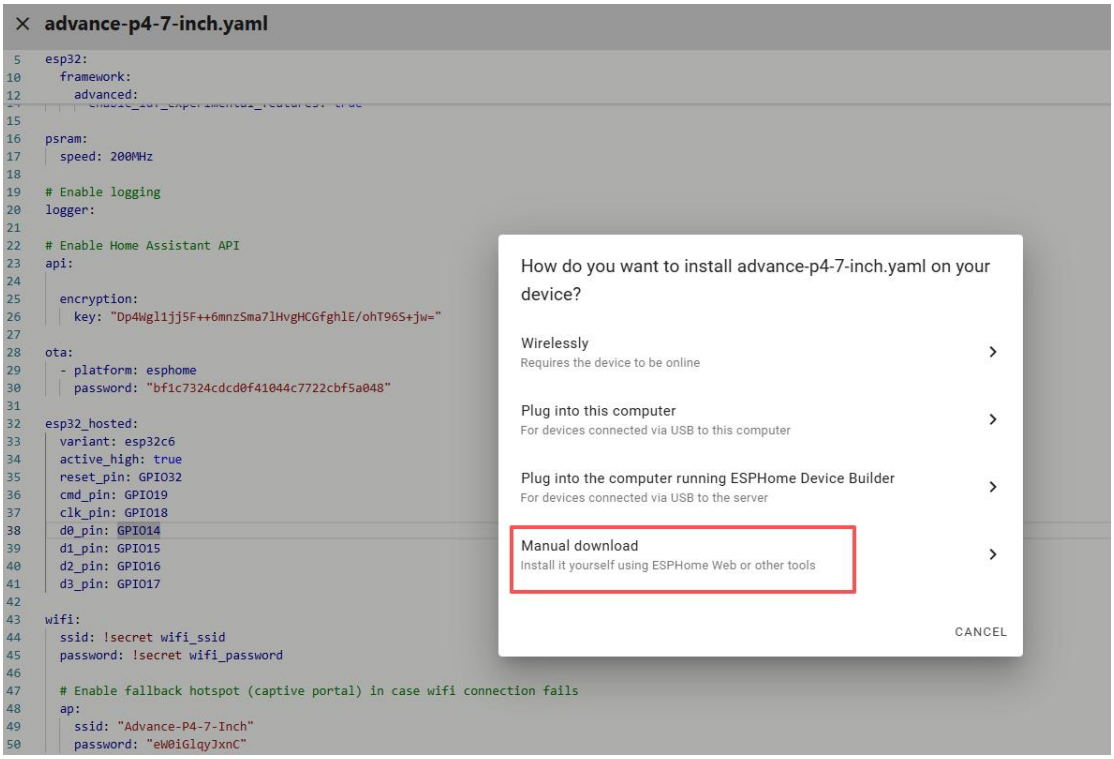
```yaml
esp32:
  framework:
    advanced:

psram:
  speed: 200MHz

# Enable logging
logger:

# Enable Home Assistant API
api:

  encryption:
    key: "Dp4Wgl1jj5F++6mnzSma7lHvgHCGfghlE/ohT96S+jw="

ota:
  - platform: esphome
    password: "bf1c7324cdcd0f41044c7722cbf5a048"

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"
```

8. First upload of code

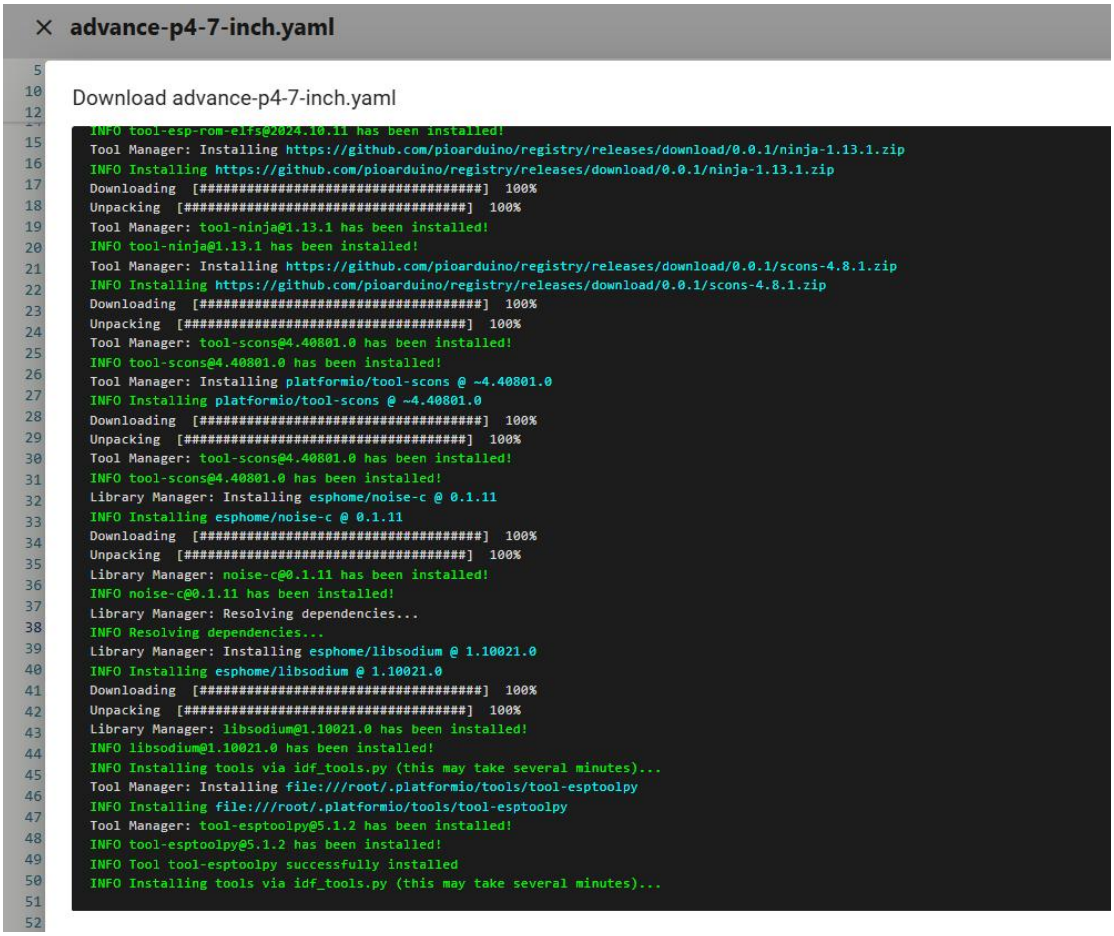
Once the code replacement is complete, click "INSTALL" in the top right corner.



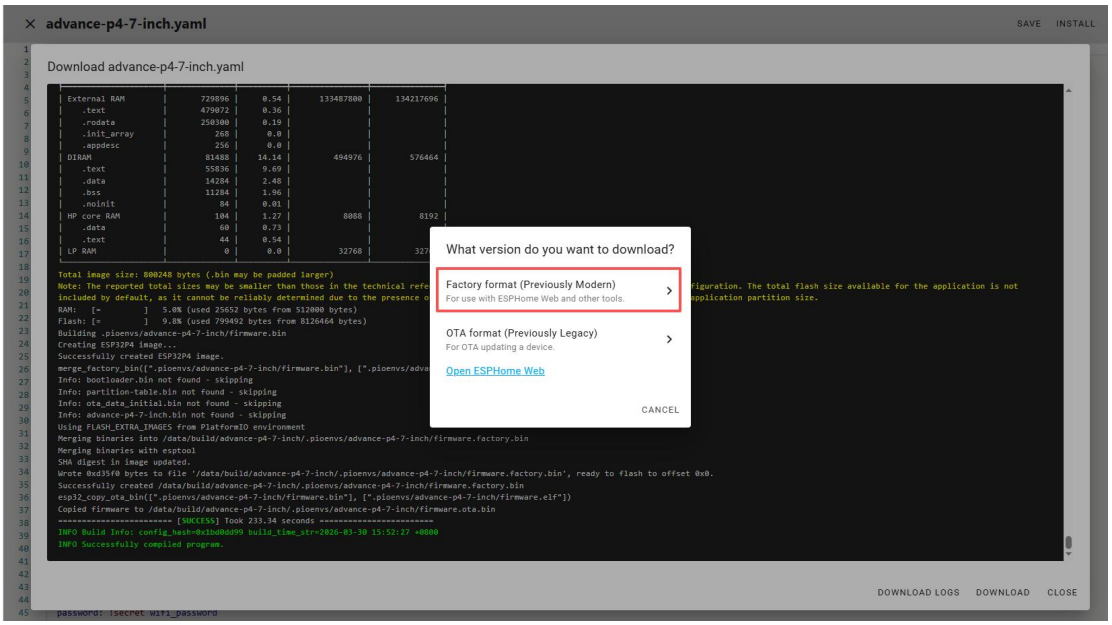
Select "Manual download".



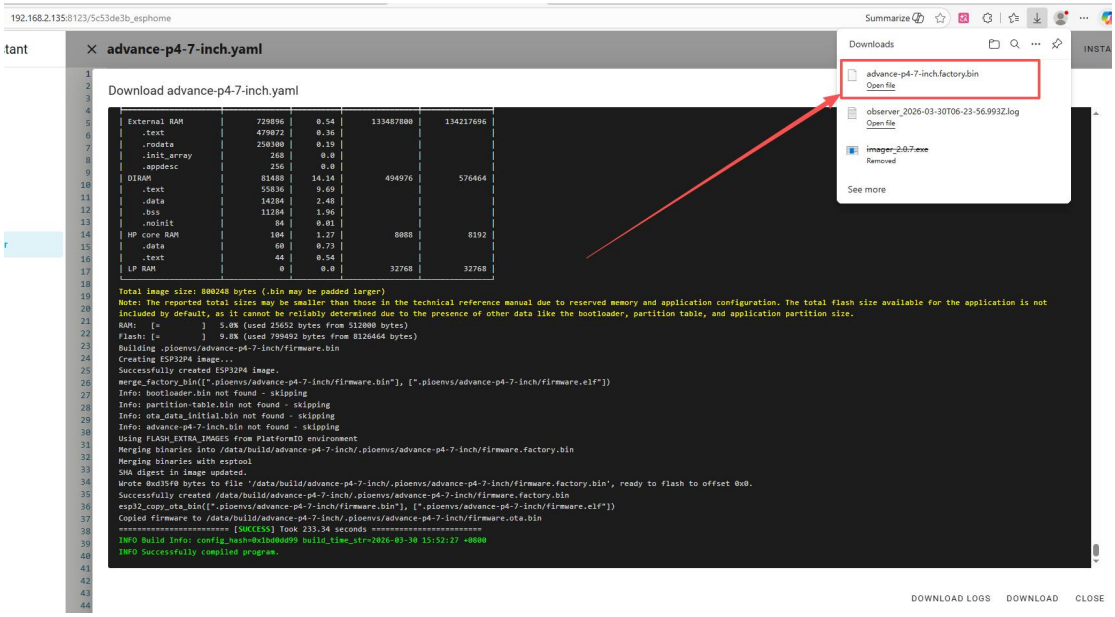
Wait for a few minutes until the installation is complete.The first installation and download process may take some time as it involves downloading necessary tools and libraries and compiling the code.



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.

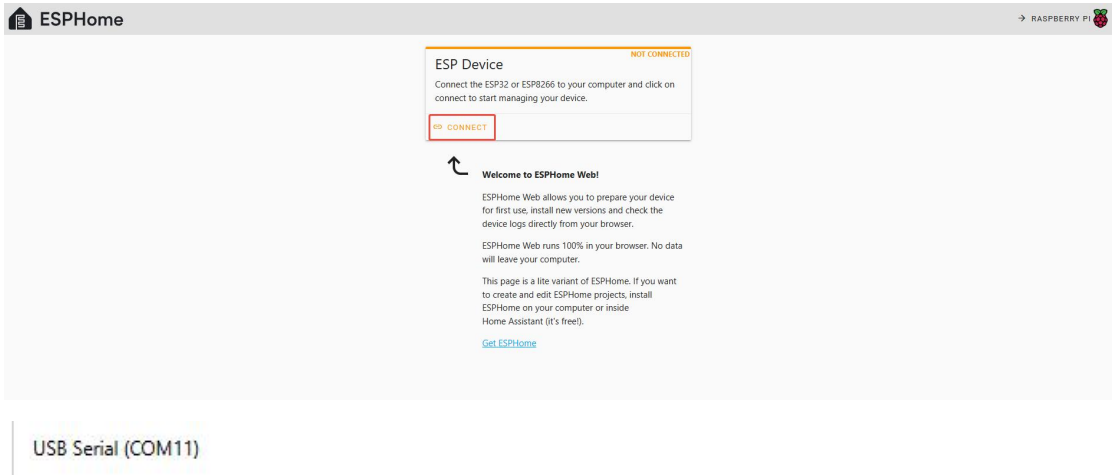
Open the following website:

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

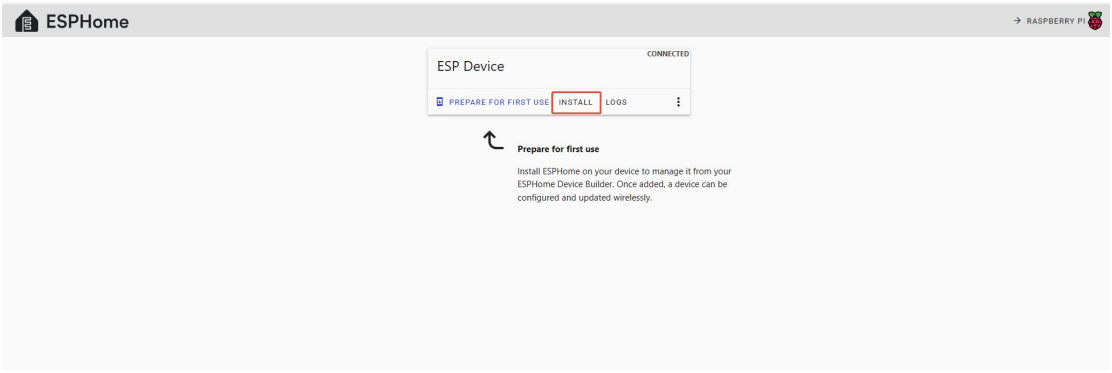
Next, we will flash this .bin file into the CrowPanel Advanced ESP32-P4 HMI AI Display（7inch / 9inch / 10.1inch）.

Connect the CrowPanel Advanced ESP32-P4 HMI AI Display（7inch / 9inch / 10.1inch） to your computer.

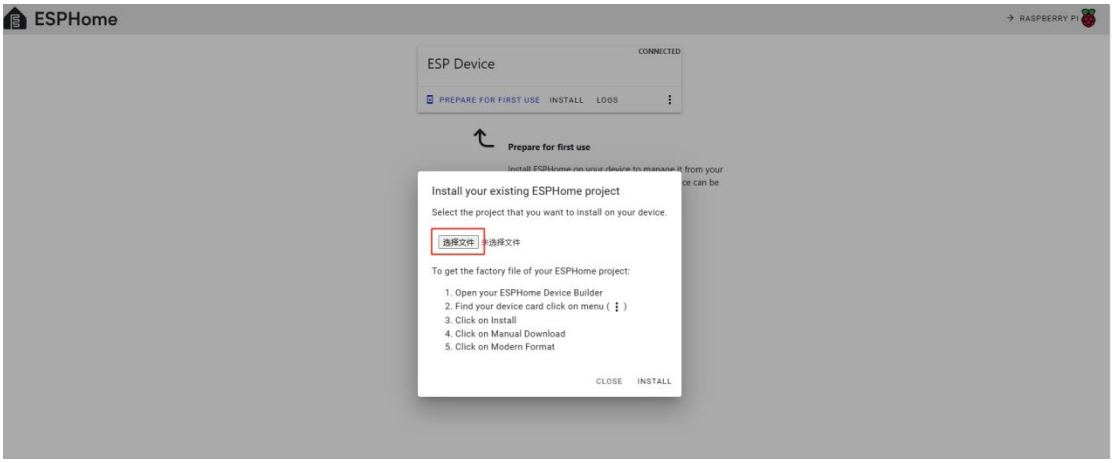
Click "Connect", select the COM port, and connect it.

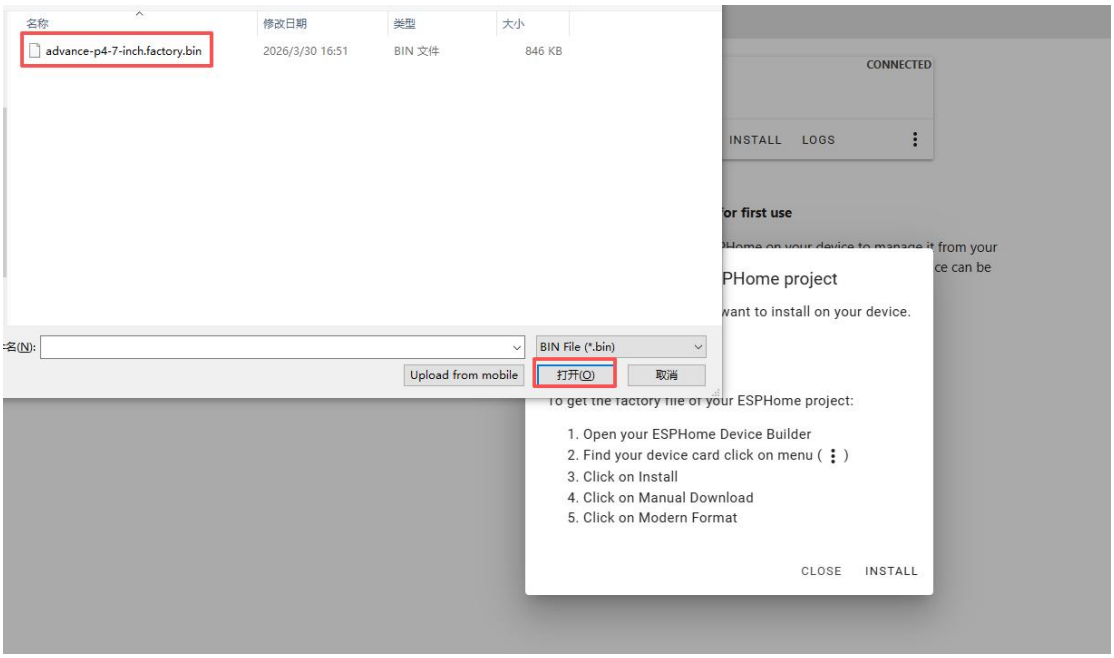


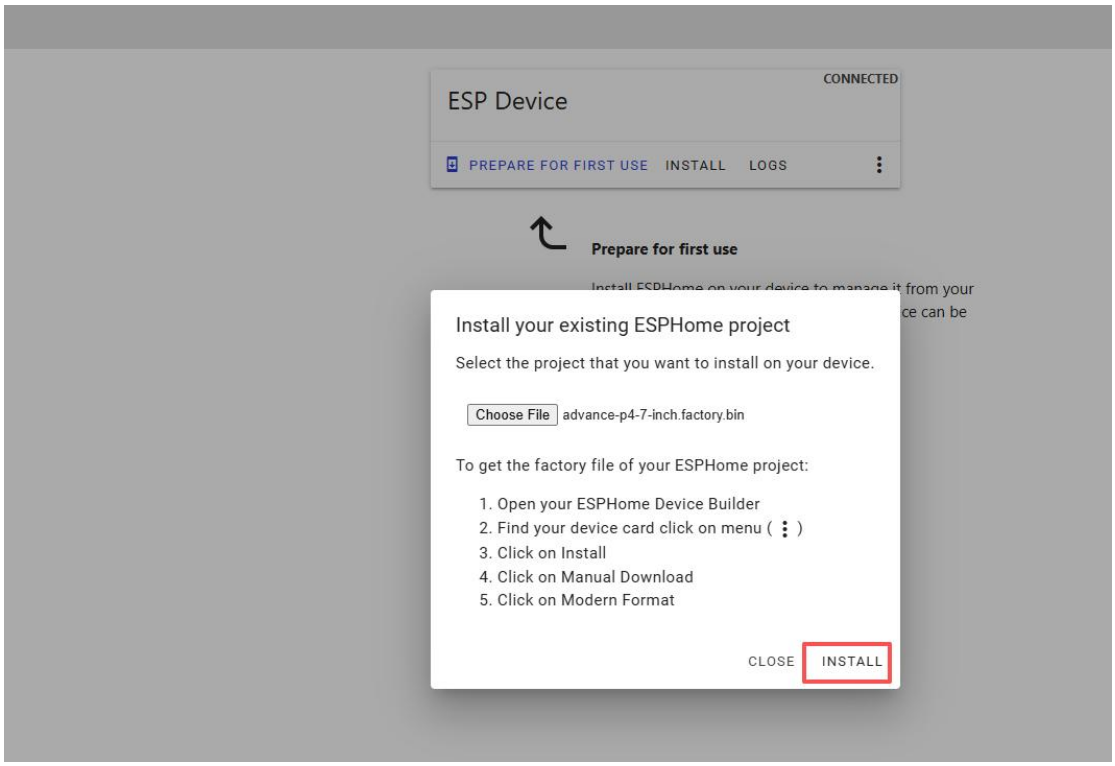
After connecting the CrowPanel Advanced ESP32-P4 HMI AI Display（7inch / 9inch / 10.1inch）, click "Install".



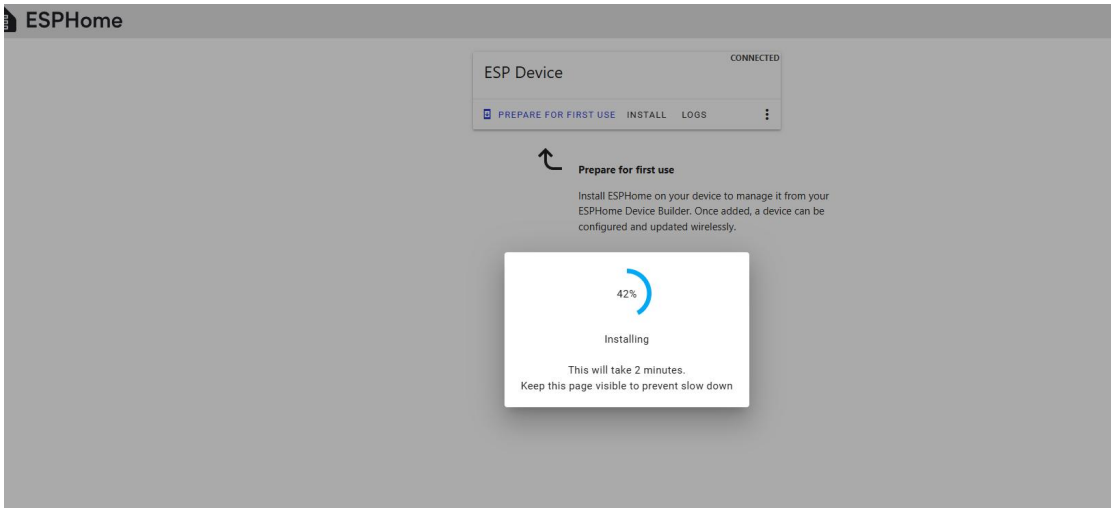
Add the .bin file you just downloaded, then click "Install".



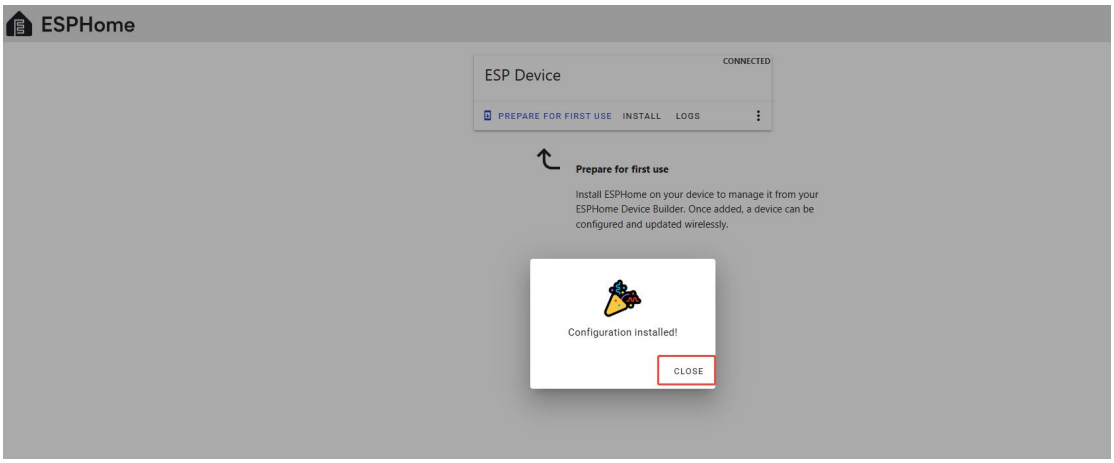




Wait for a few minutes.

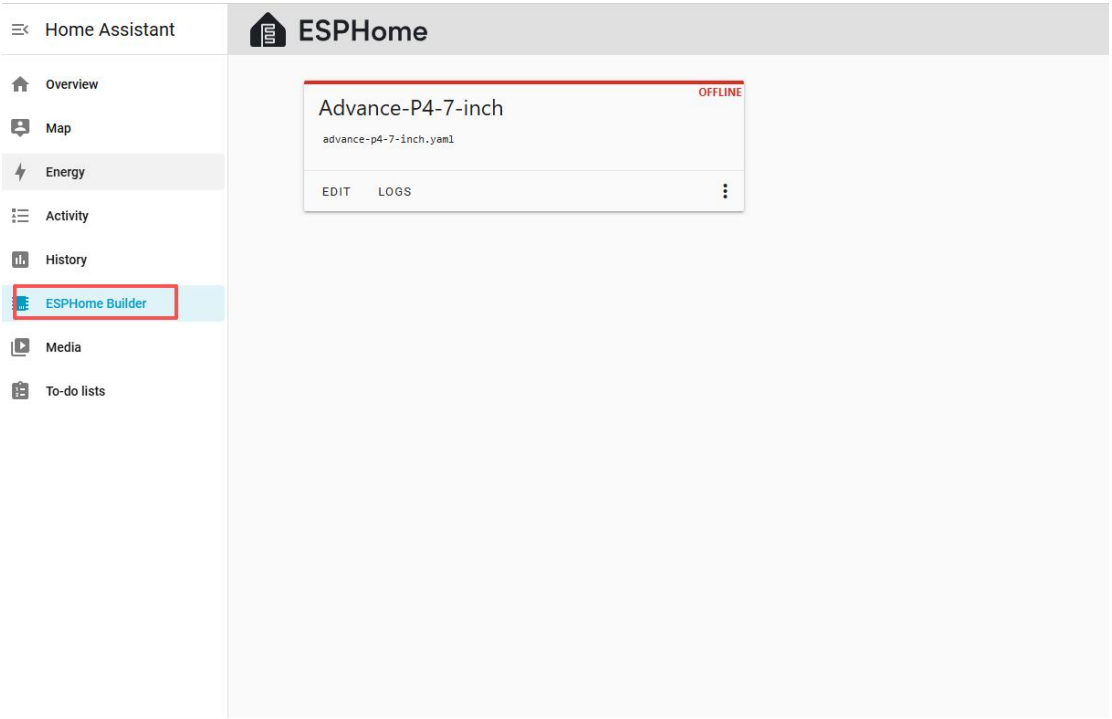


After the installation is complete, click "Close".

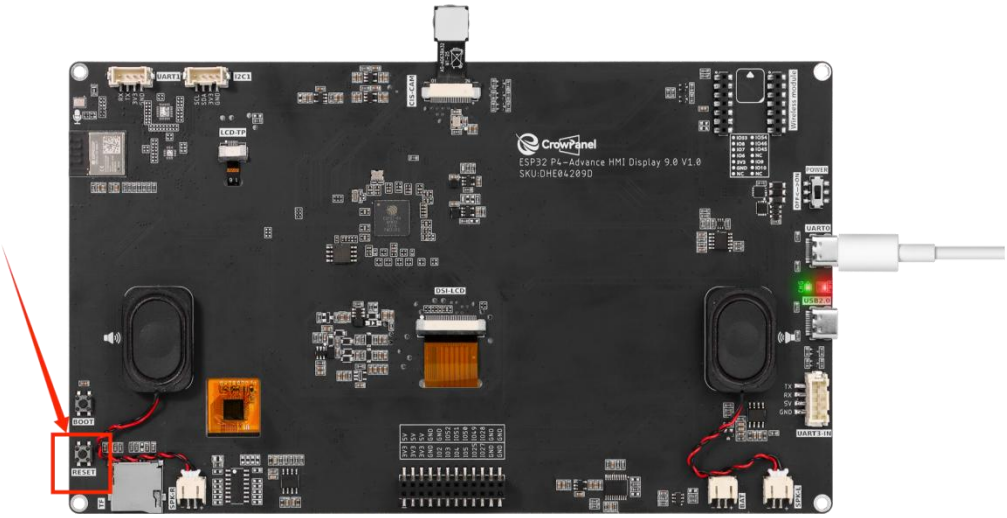


9. Observe the Wi-Fi connection status of the equipment

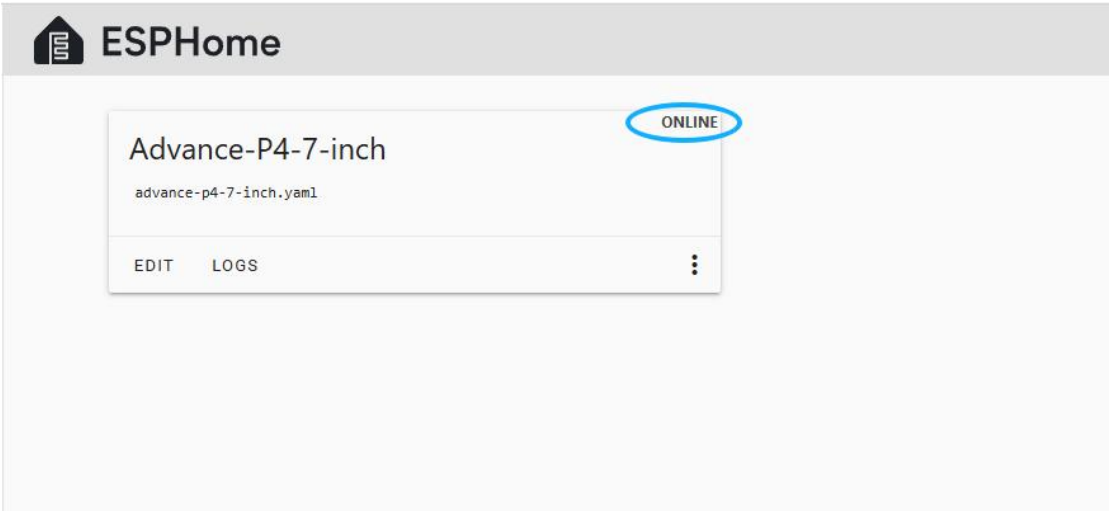
After successfully flashing the .bin file, return to the ESPHome page in Home Assistant.



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 ESP32-P4 HMI AI Display (7inch / 9inch / 10.1inch), 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

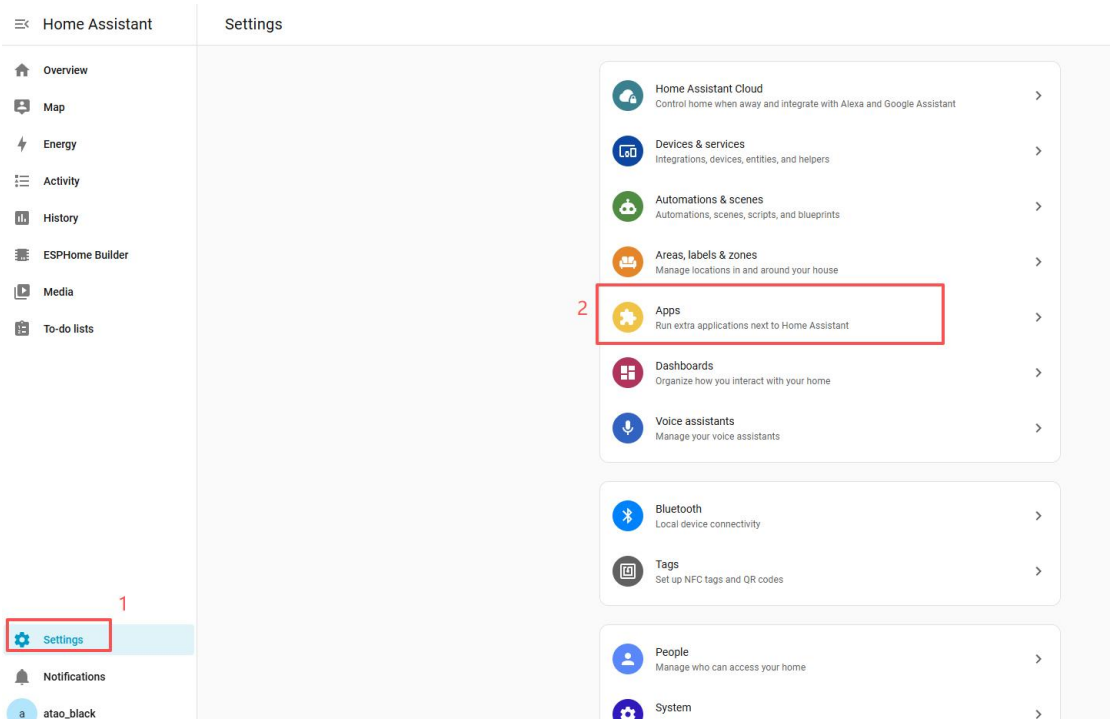
You can see the materials on the screen, which we need to use on the ESPHome platform, so they need to be uploaded to the ESPHome platform.

Click here to download the materials shown on our screen.

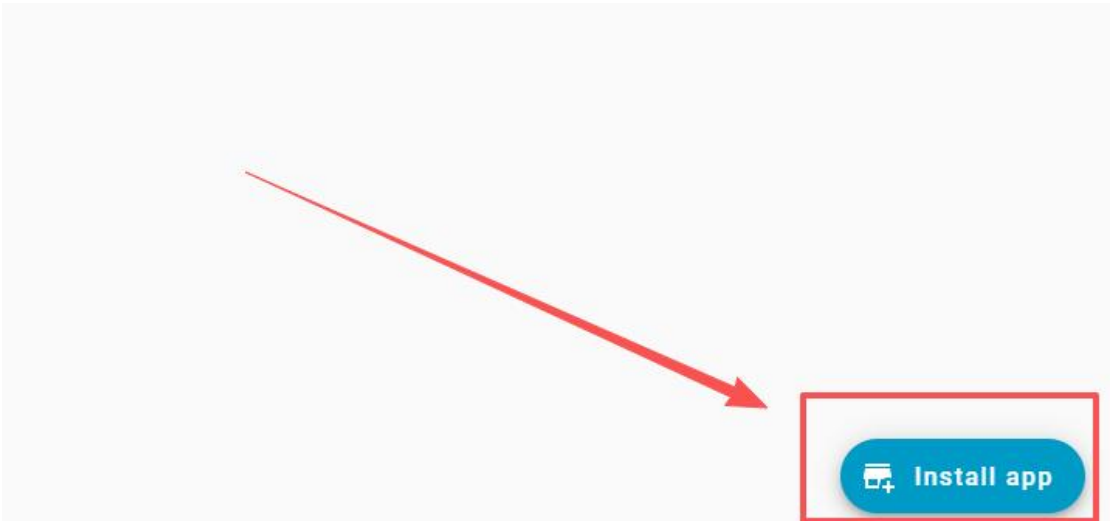
https://github.com/Elecrow-RD/CrowPanel-Advanced-7inch-ESP32-P4-HMI-AI-Display-1024x600-IPS-Touch-Screen/tree/master/example/V1.0/ESPHome/Materials

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

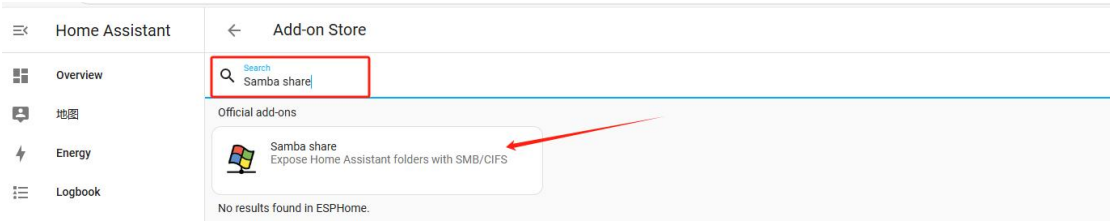
Click Settings and select Apps



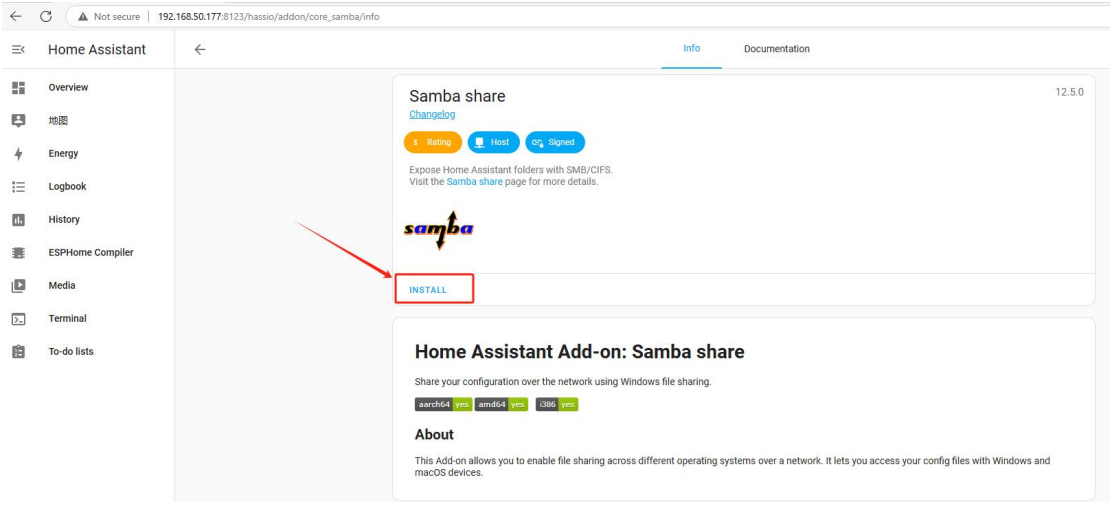
Then, click the “Install app” at the bottom right corner.



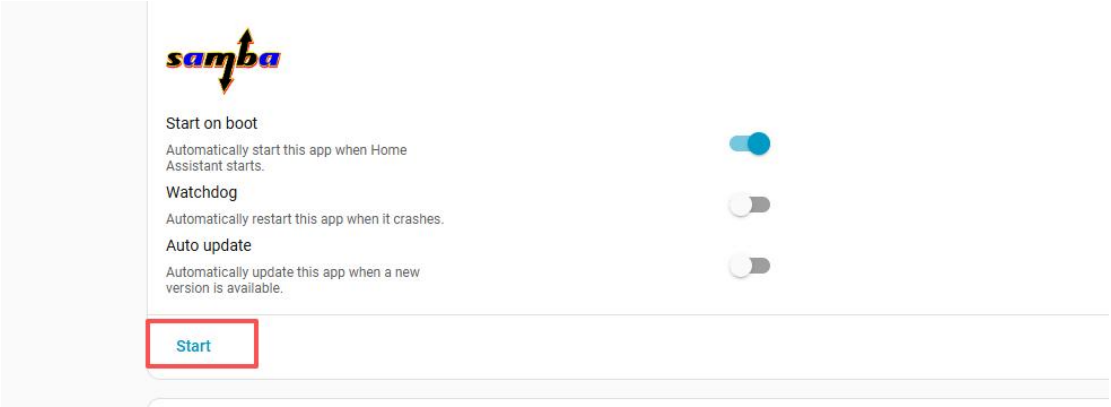
Search for Samba share at the top



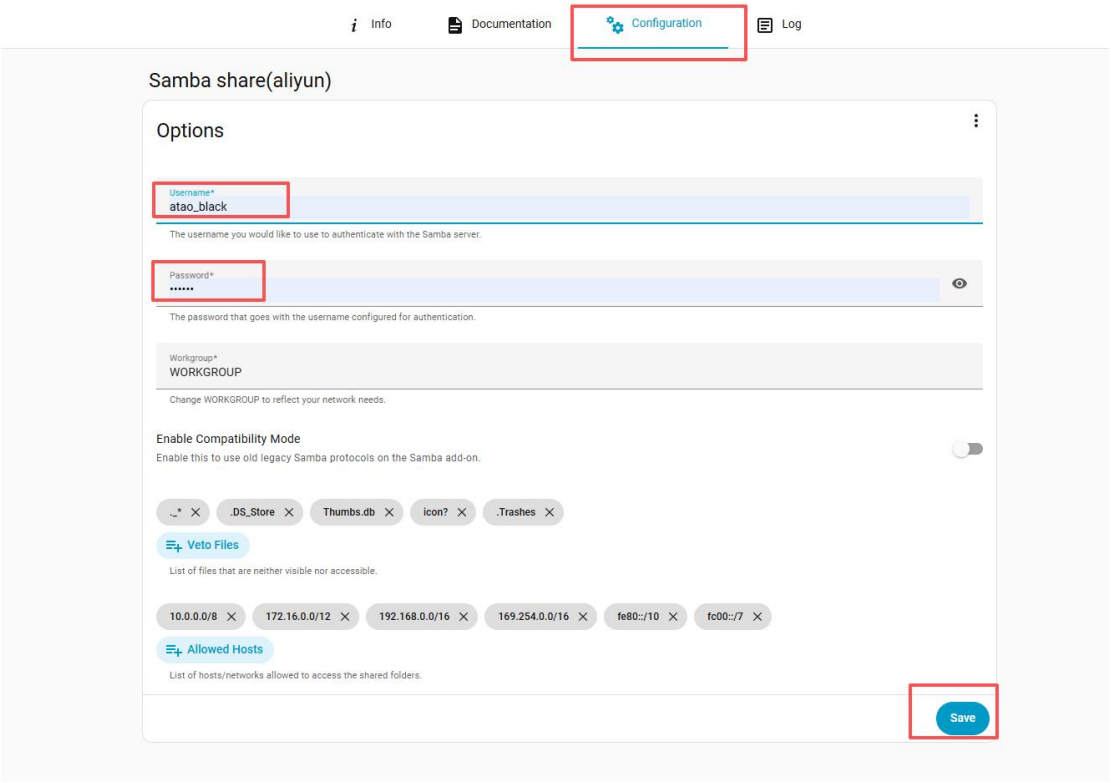
Click INSTALL to install it



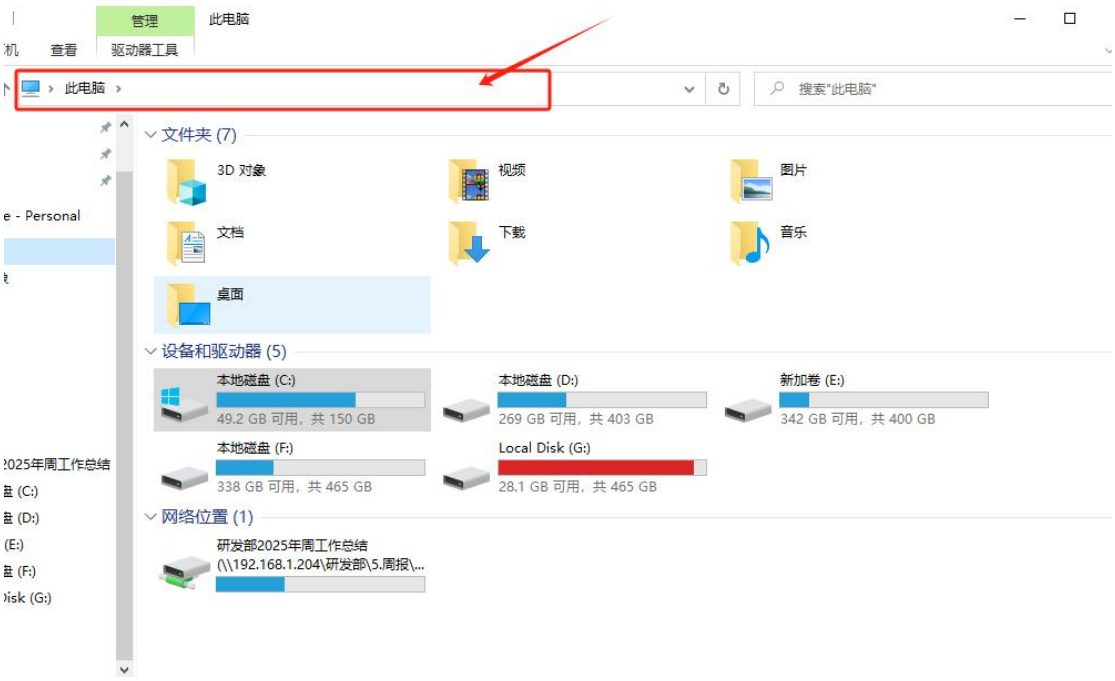
After the installation is completed, remember to start it.



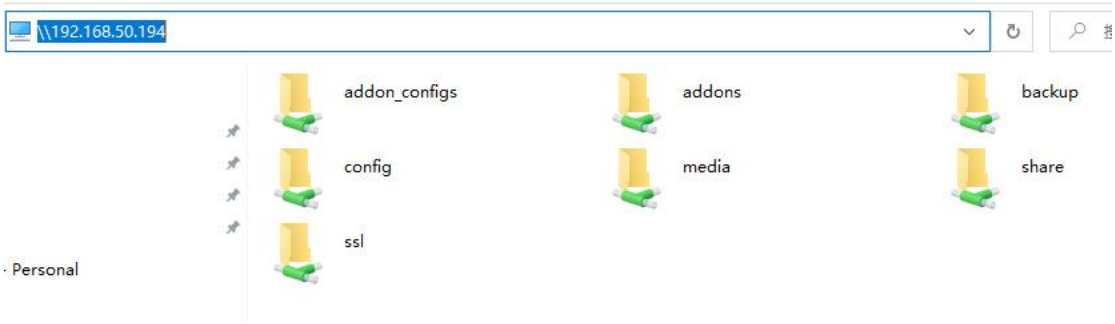
After installation, configure your Samba share with your own username and password—please remember them!



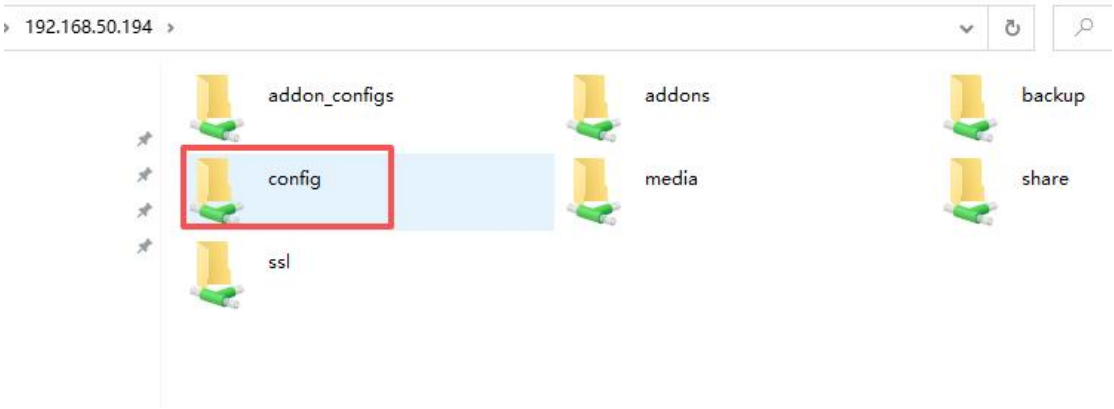
After saving, go to the File Explorer on your computer



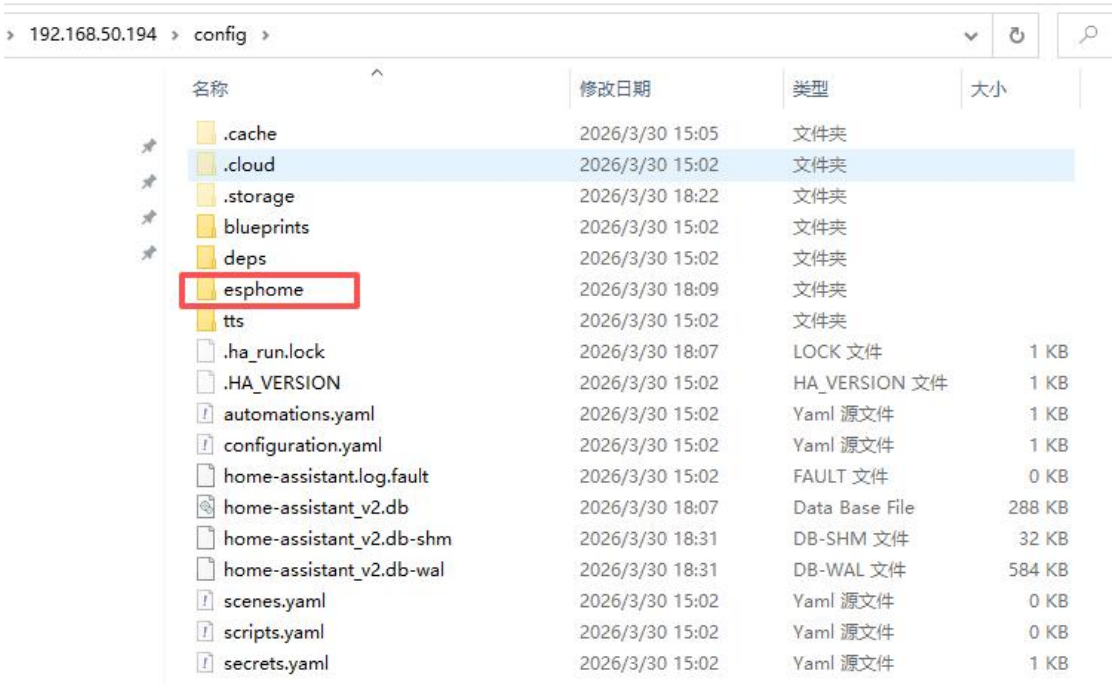
Enter \\ + your Home Assistant IP address



Go to config



Then choose esphome



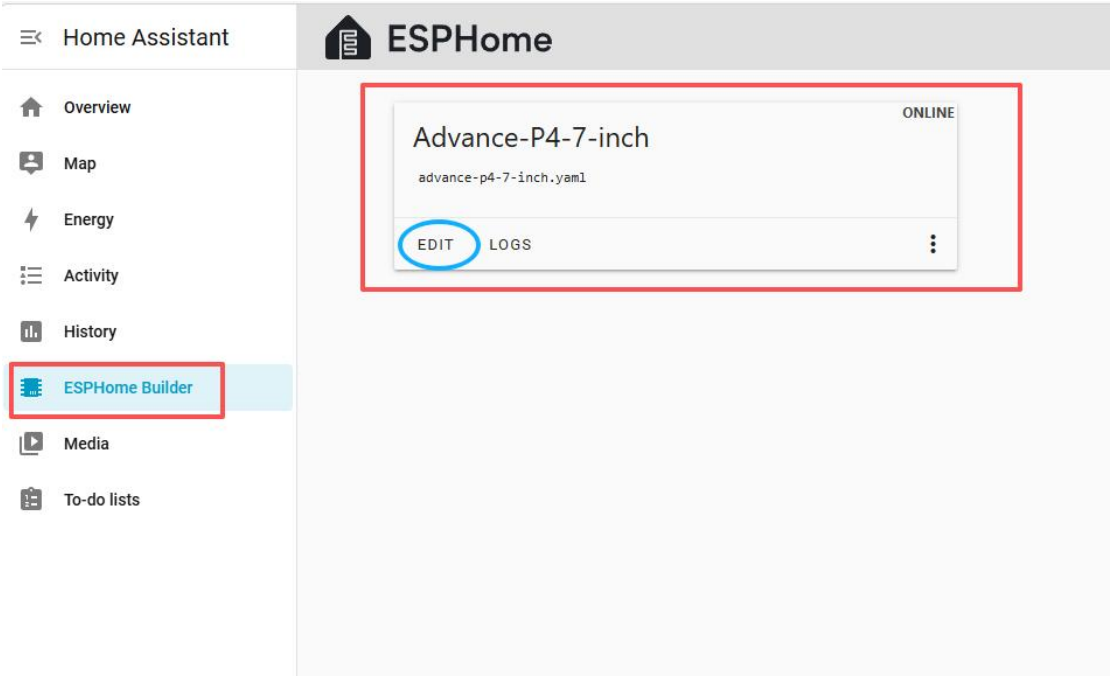
Put the materials you just downloaded into the esphome folder

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

The function of this code is to automatically perform a series of initialization operations when the device "starts up after power-on", ensuring that the screen and backlight are in a correct working state.

× advance-p4-7-inch.yaml

```yaml
esphome:
  name: advance-p4-7-inch
  friendly_name: Advance-P4-7-inch
  on_boot:
    priority: 800
    then:
      - logger.log: "Backlight ON"
      - output.turn_on: backlight_pwm
      - delay: 200ms
      - output.turn_off: power_light
      - delay: 500ms

      - light.turn_on:
          id: back_light
          brightness: 1
```

Firstly, the "esphome" part is the basic configuration of the entire device, equivalent to giving the device a name (advance-p4-7-inch), which is convenient for you to identify it in the system or Home Assistant.

The most crucial part is "on_boot", which means "the actions to be executed when the device starts up", and priority: 800 indicates that these actions will be executed at an earlier stage of system startup (the higher the priority, the earlier it is executed). This ensures that the screen is lit up before the system is fully operational.

Then the program starts to execute step by step: first, it prints a log "Backlight ON" using logger.log, which is convenient for you to see what the device is doing during debugging;

then, it uses output.turn_on: backlight_pwm to turn on the PWM output of the backlight, that is, to make the screen start to emit light; then, it delays for 200 milliseconds (giving the hardware some reaction time), and then executes output.turn_off: power_light to turn off an indicator light or power light (usually the LED on the board); then, it delays for 500 milliseconds to ensure that all circuits are stable; finally, it uses light.turn_on to turn on the backlight "light" and set the brightness to the maximum (brightness: 1), so that the screen will display in the brightest state.

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 7-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 + ldo + 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 provide normal power supply, be able to output debugging information, and also be able to connect to the network for control and remote upgrade".

```yaml
psram:
  speed: 200MHz

esp_ldo:
  - channel: 3
    voltage: 2.5V
  - channel: 4
    voltage: 2.5V

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

# Enable Home Assistant API
api:

  encryption:
    key: "Dp4Wgl1jj5F++6mnzSma7lHvgHCGfghlE/ohT96S+jw="

ota:
  - platform: esphome
    password: "bf1c7324cdcd0f41044c7722cbf5a048"
```

Firstly, PSRAM: Speed: 200MHz refers to configuring external memory (PSRAM). You can think of it as "extra large memory". Since this project has a 7-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.

Next is esp_ldo. This part might be difficult for you to understand. In fact, it is setting the internal power supply voltage regulator (LDO) of the chip. To put it simply: it provides a stable voltage for certain modules of the chip. Here, two channels (channel 3 and 4) are configured, both outputting 2.5V. The main purpose is to ensure that voltage stability for components like PSRAM or high-speed peripherals during high-frequency operation is maintained. Otherwise, there might be system crashes, restarts, or strange errors. It can be understood as "establishing a power foundation for the high-speed system".

Then comes the logger, which is the logging system:

```yaml
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:

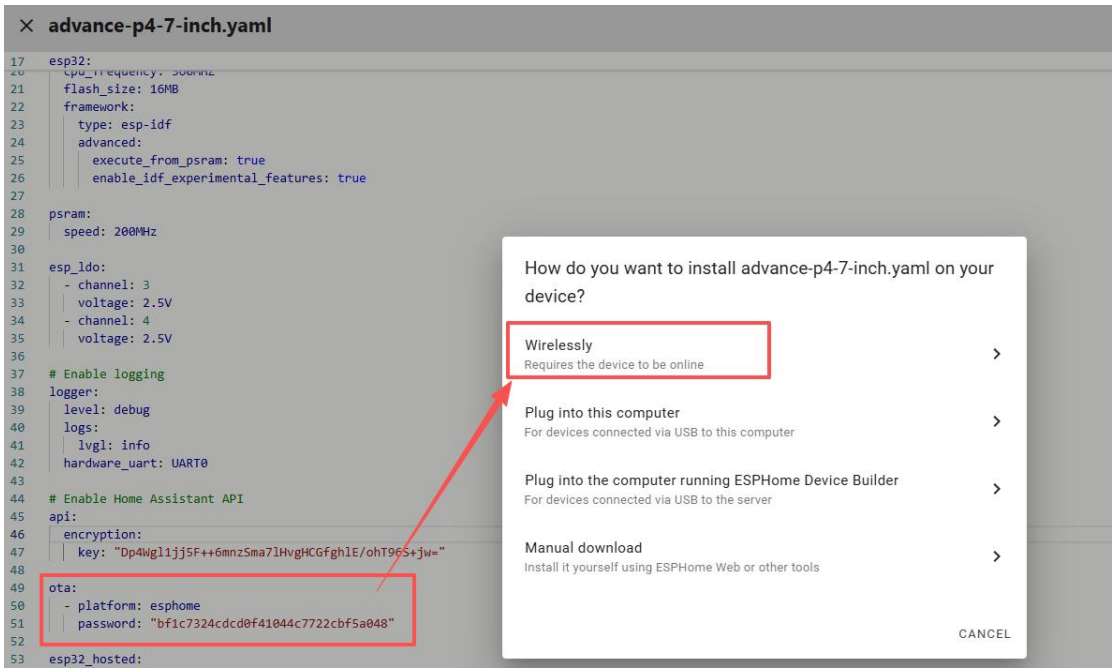
```
44  # Enable Home Assistant API
45  api:
46    encryption:
47      key: "Dp4Wgl1jj5F++6mnzSma7lHvgHCGfghlE/ohT96S+jw="
```

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:

```
49  ota:
50    - platform: esphome
51      password: "bf1c7324cdcd0f41044c7722cbf5a048"
```

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

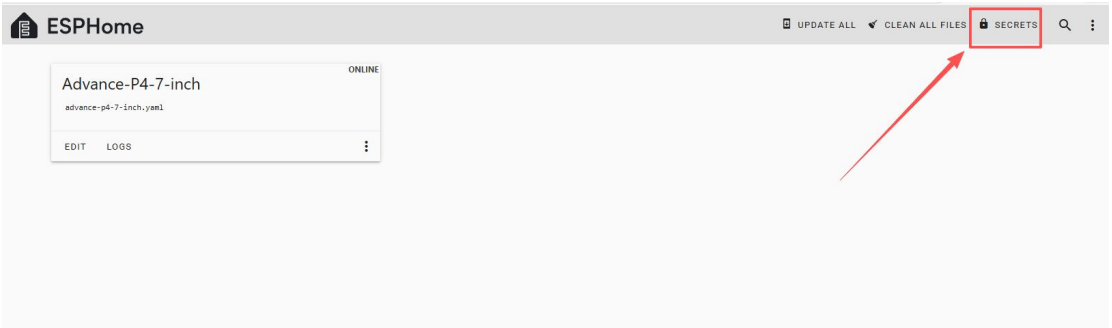
This piece of code is actually accomplishing a very crucial task: enabling the WiFi-less ESP32-P4 to "borrow" the networking capabilities of ESP32-C6, and configuring the wireless network connection and backup hotspot.

```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-P4-7-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

```
# 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.

```
touchscreen:
  - platform: gt911
    id: my_touchscreen
    i2c_id: bus_a
    reset_pin: 40
    interrupt_pin: 42
    update_interval: 50ms
    transform:
      swap_xy: false
      mirror_x: false
      mirror_y: false
    # Enabled Touch Logger here
    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: 40 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.

8.output

```yaml
output:
  - platform: ledc
    pin: GPIO31
    id: backlight_pwm
  - platform: gpio
    id: power_light
    pin: GPIO29
    inverted: true
```

This output configuration in ESPHome defines two hardware output interfaces: The first one is backlight_pwm, which uses the LEDC platform to output PWM signals on GPIO31, for smoothly adjusting the LCD backlight brightness. It can achieve a gradual brightening or dimming effect by controlling with percentages, providing precise brightness management.

The second one is power_light. It uses the ordinary GPIO platform to output digital levels on GPIO29 to control the power supply or switch devices. It also enables the parameter "inverted: true", meaning that a high level corresponds to the device being turned off and a low level corresponds to it being turned on, to adapt to the hardware logic triggered by low levels. Both of these outputs are defined with an internal ID, which can be called in light, switch, or automation logic to achieve the linkage between hardware control and software logic, forming a complete control system that can adjust brightness and provide switch control.

9.display

This display configuration in ESPHome defines a MIPI DSI interface LCD display with the ID of "my_display" and the model of "WAVESHARE-ESP32-P4-WIFI6-TOUCH-LCD-7B", which is used to display content when connected to the ESP32-P4 control board.

```yaml
display:
  - platform: mipi_dsi
    id: my_display
    model: WAVESHARE-ESP32-P4-WIFI6-TOUCH-LCD-7B
    reset_pin:
      number: 41
    update_interval: never
    auto_clear_enabled: false
    dimensions:
      width: 1024
      height: 600
    color_order: RGB
    color_depth: 16
```

reset_pin: 41 specifies the hardware reset pin, which is used for initializing the screen.
update_interval: never indicates that the screen content will not be refreshed automatically, but will be updated manually through the program or LVGL controls. auto_clear_enabled: false retains the screen content without automatically clearing it, facilitating the continuous display of interface elements.
The screen resolution is 1024x600, using RGB color sequence, with a color depth of 16 bits,

capable of displaying high-quality color images and icons.
This configuration provides a hardware foundation for the subsequent display of LVGL interface controls, images, and text, and allows for precise control of refresh and content management, achieving high-performance, customizable touchscreen display.

10.image

This image configuration in ESPHome defines a set of image resources for the touchscreen interface. Each image specifies the file path, internal ID, display size, pixel format and byte order, making it convenient to call in the LVGL control.

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

The images here are the ones we previously added using the Samba Share tool. We are now initializing these images.

Specifically, small_temp.png and small_hum.png are used to display small icons for temperature and humidity, with a uniform size of 200x200 pixels. They use the 16-bit RGB565 pixel format and little-endian byte order to ensure correct color display on the ESP32 and low memory usage;
small_logo.png is the interface or brand LOGO, with a size of 200x100. It also uses the RGB565 format and little-endian storage to ensure clear display;

light.png and no_light.png correspond to the on and off states of the light, with a size of 200x200 pixels. They support the alpha_channel transparent channel, allowing the images to be overlaid with the interface background or other controls, achieving a visual overlay effect and aesthetic state transition, while ensuring hardware compatibility and display efficiency.

11.lvgl

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

```yaml
lvgl:
  buffer_size: 50%
  byte_order: little_endian
  color_depth: 16
  log_level: INFO
  bg_color: 0xFFFFFF
  text_font: montserrat_26
  widgets:
    - image:
        id: lvgl_logo
        src: small_logo
        x: 400
        y: 20
    - image:
        id: lvgl_temp
        src: small_temp
        x: 250
        y: 250
    - image:
        id: lvgl_hum
        src: small_hum
        x: 500
        y: 250
    - image:
        id: lvgl_light
        src: light_off    # Default 'Turn off lights' image
        x: 750            # You can adjust the position by yourself
        y: 250
    - label:
        id: label_temp
        x: 190
        y: 350
        text: "Temp: --°C"
    - label:
        id: label_hum
        x: 450
        y: 350
        text: "Humi: --%"
    - label:
        id: label_light_state
        x: 750
        y: 350
        text: "OFF"
```

First, by setting buffer_size: 50% , LVGL uses half of the screen's video memory as the drawing buffer to balance the refresh speed and memory usage. byte_order: little_endian and color_depth: 16 ensure that the 16-bit RGB565 color data is stored and displayed in little-endian

format, making the interface colors accurate and compatible with the ESP32-P4 display hardware. bg_color: 0xFFFFFF sets the default background of the entire screen to white. text_font: montserrat_26 defines the global text font to ensure clear and beautiful text display. log_level: INFO enables the LVGL internal status log for easier debugging and monitoring of the interface rendering process.

In the widgets section, various controls have been configured:
The lvgl_logo displays the top LOGO image, located at screen coordinates (400,20);
The lvgl_temp and lvgl_hum display the temperature and humidity icons respectively, at coordinates (250,250) and (500,250), which are the main information area in the middle of the interface;

The lvgl_light is used to display the status diagram of the light, initially set as light_off, at coordinates (750,250), corresponding to the physical LED or switch status;

At the same time, three text labels label_temp, label_hum and label_light_state are located at (190,350), (450,350) and (750,350) respectively, with initial text "Temp: --℃", "Humi: --%" and "OFF" to display the real-time temperature and humidity data and the switch status of the light.
The entire configuration achieves layout management of interface images and text, dynamic data update, status synchronization, and visual aesthetics. Through the control IDs and coordinate system of LVGL, the interface can be refreshed instantly in the program or touch interaction, realizing the visual display of sensor data and physical switch status, while ensuring rendering efficiency, color accuracy, and clarity of control layout. It is a complete foundation for the development of touch screen interactive interface.

12.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

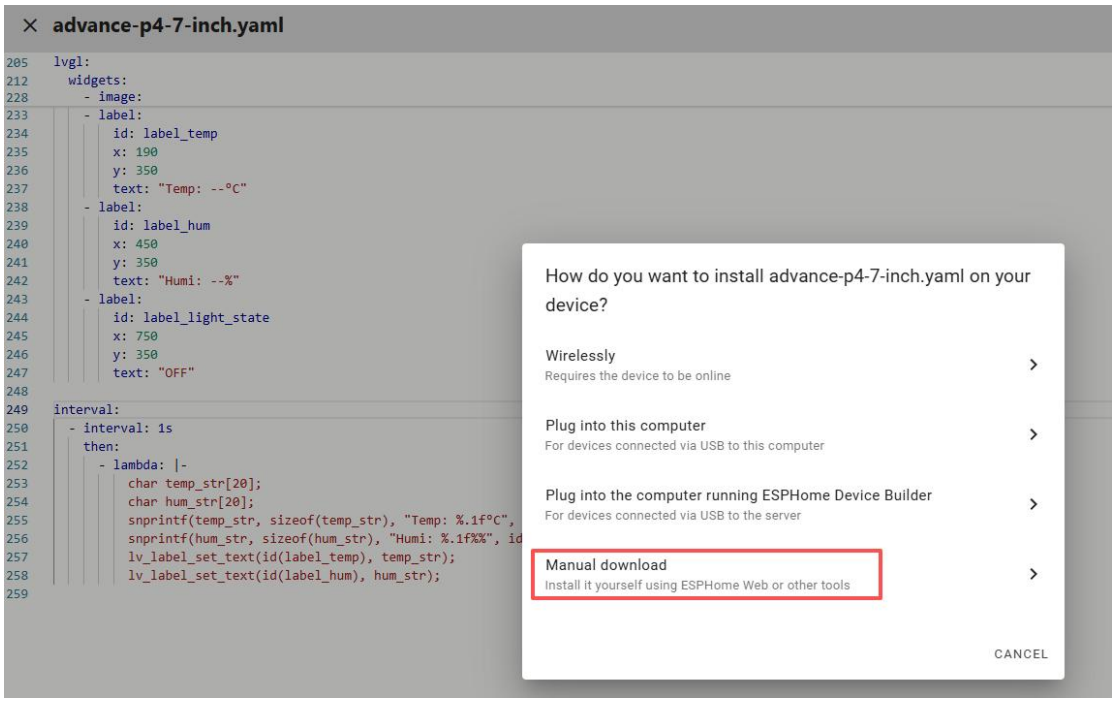
and label_hum control widgets to the latest formatted temperature and humidity strings, thereby reflecting the current environmental data in real time on the touch screen interface.

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

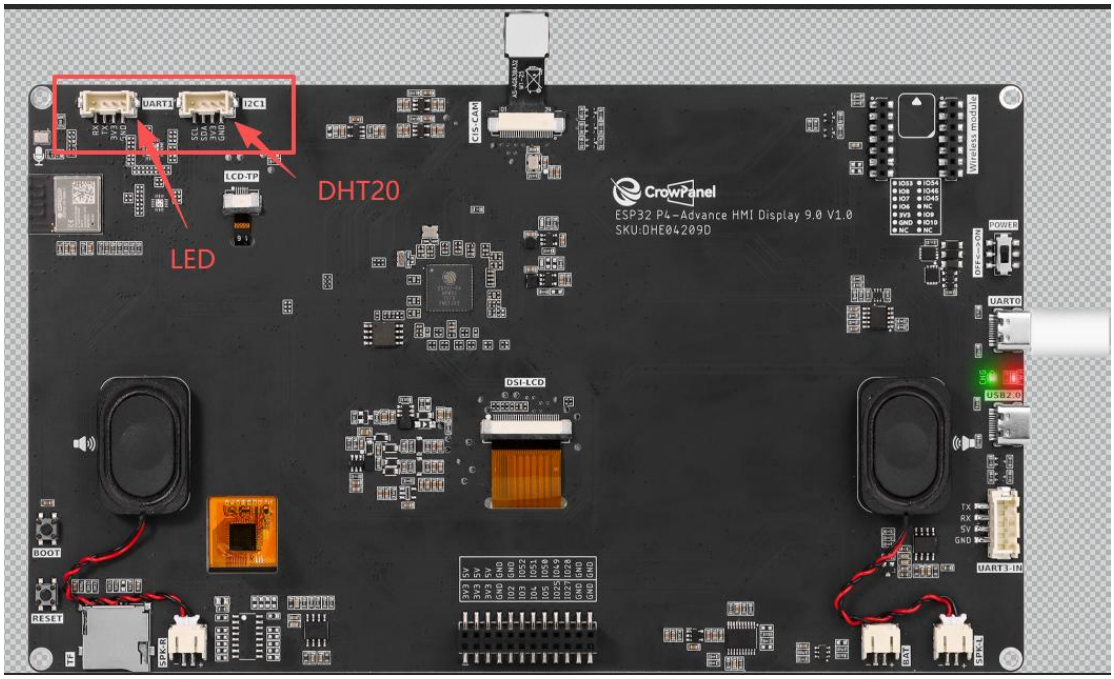
Now, all the codes have been prepared. Next, we will upload the codes.

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.

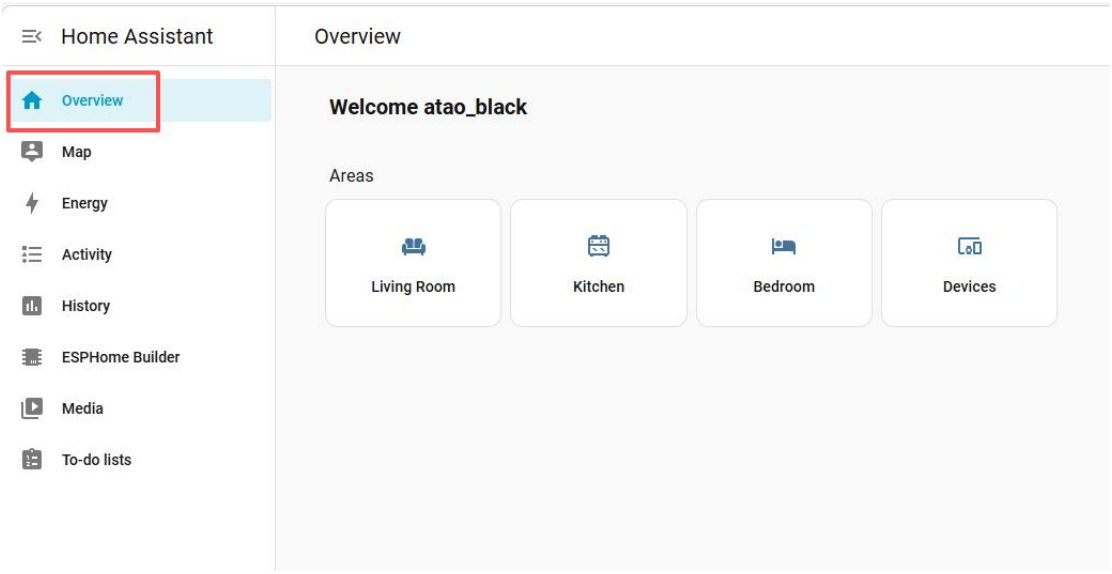


Then you will be able to see the real-time display of temperature and humidity data on the screen.

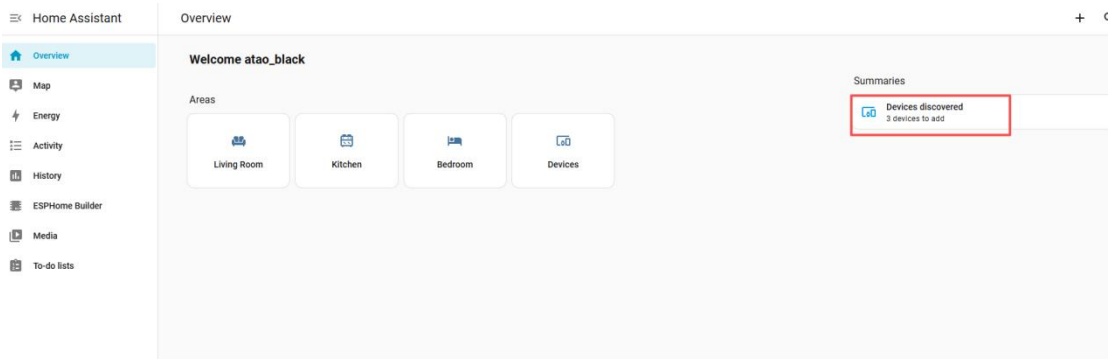
13.Remote observation and control

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.

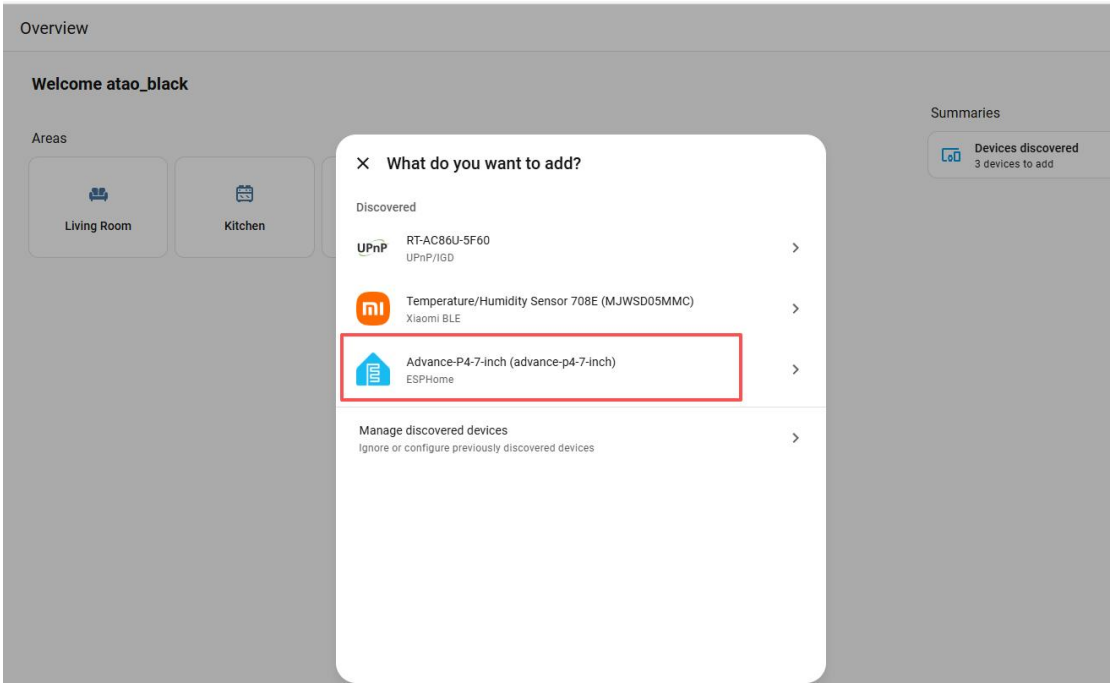
Let's return to the initial interface.



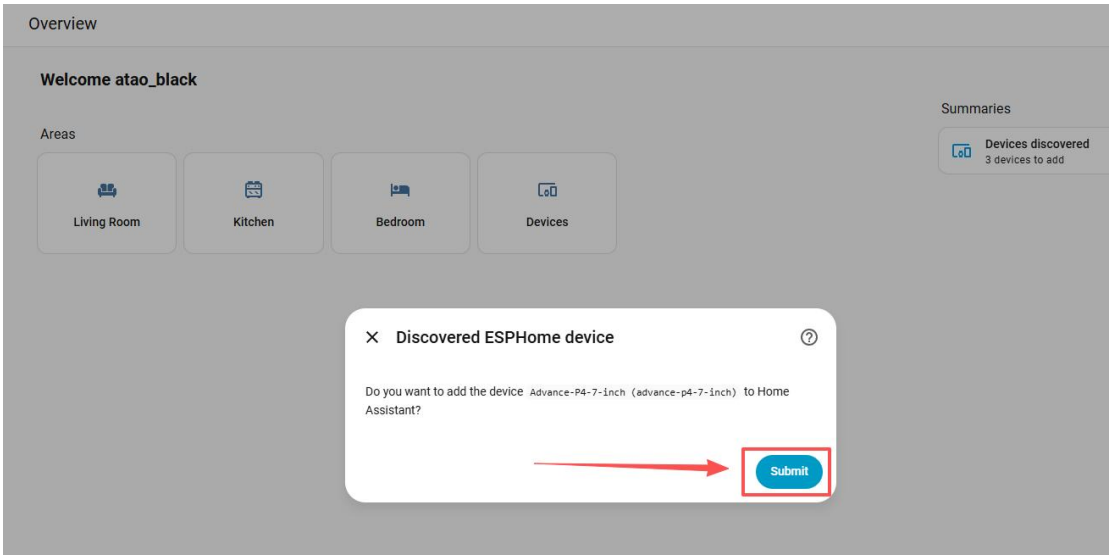
Once your code is uploaded successfully, you will be able to see your device information here.



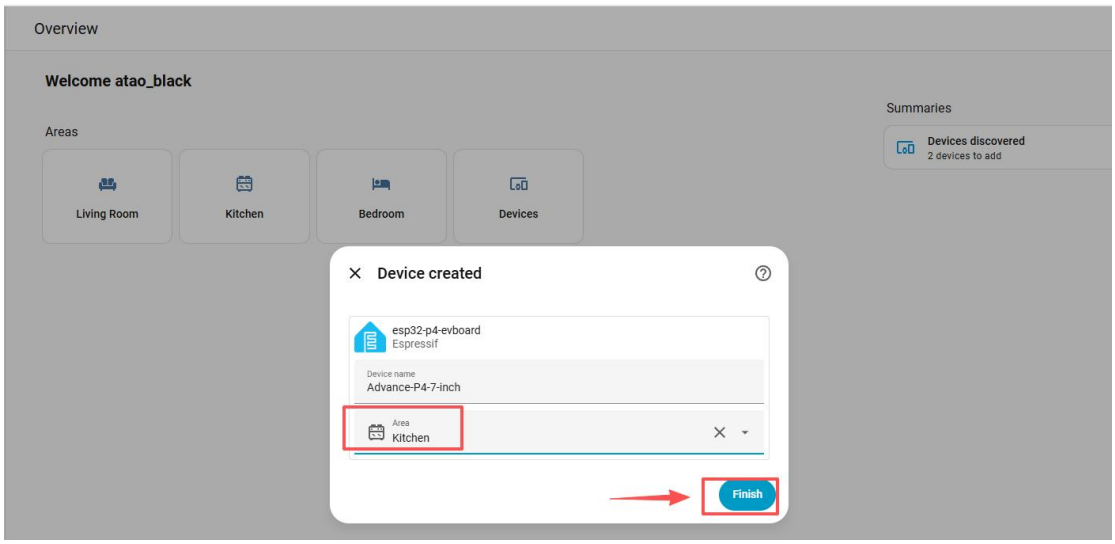
This is the system's scan of the devices nearby you.



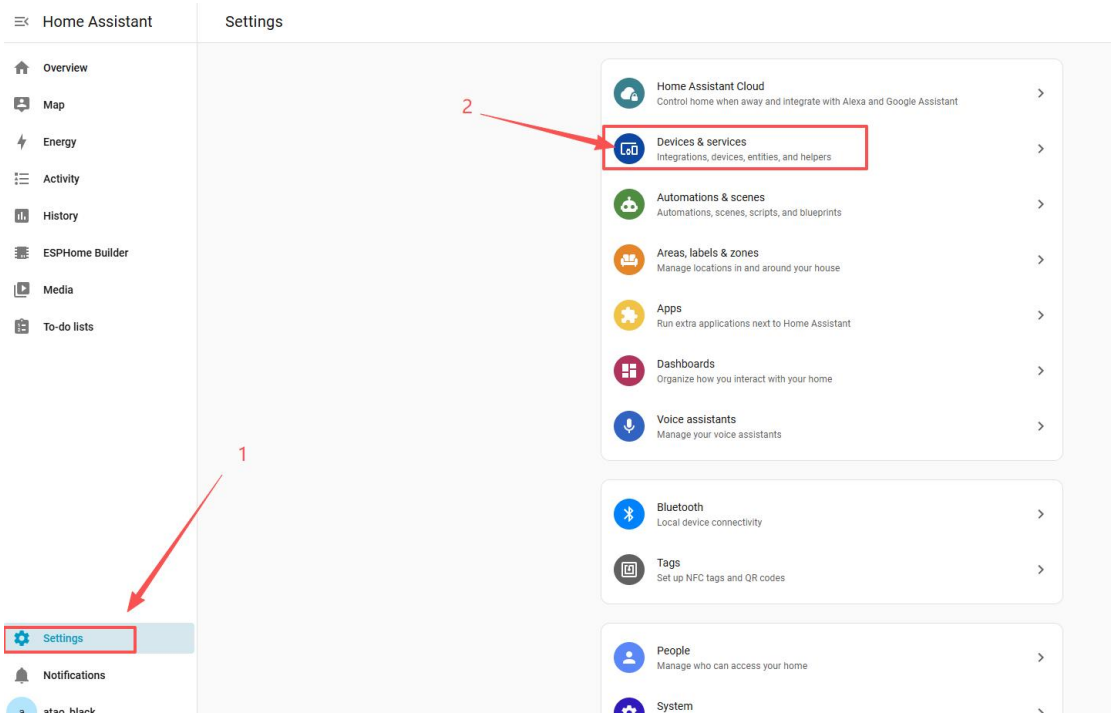
Click on our device and confirm the addition.



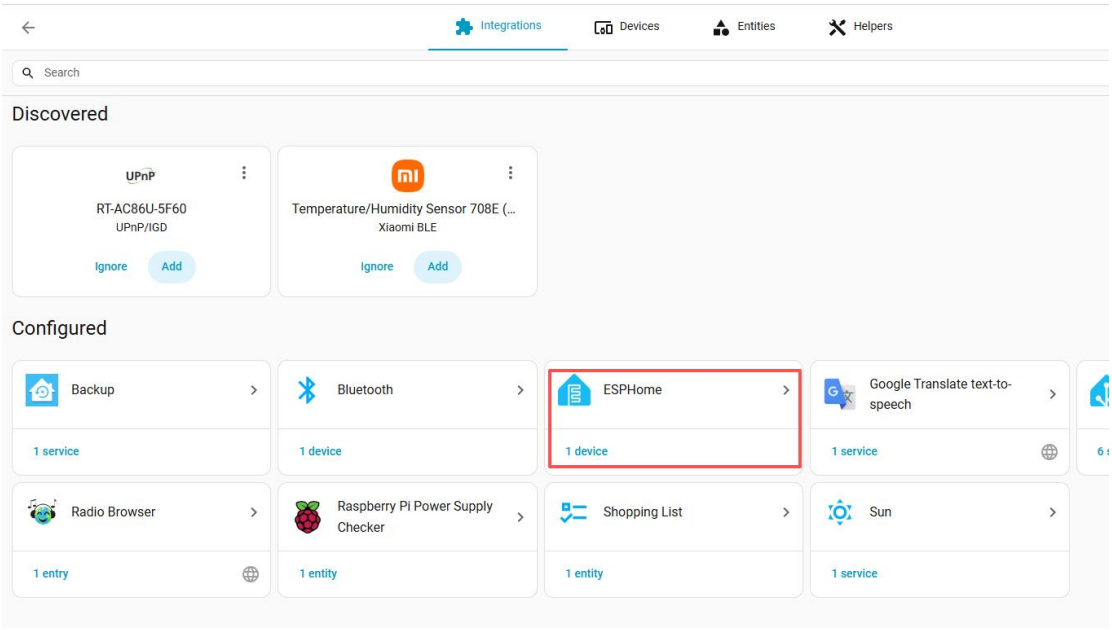
You can also create a zone for it. Here, I choose to place it in "kitchen".



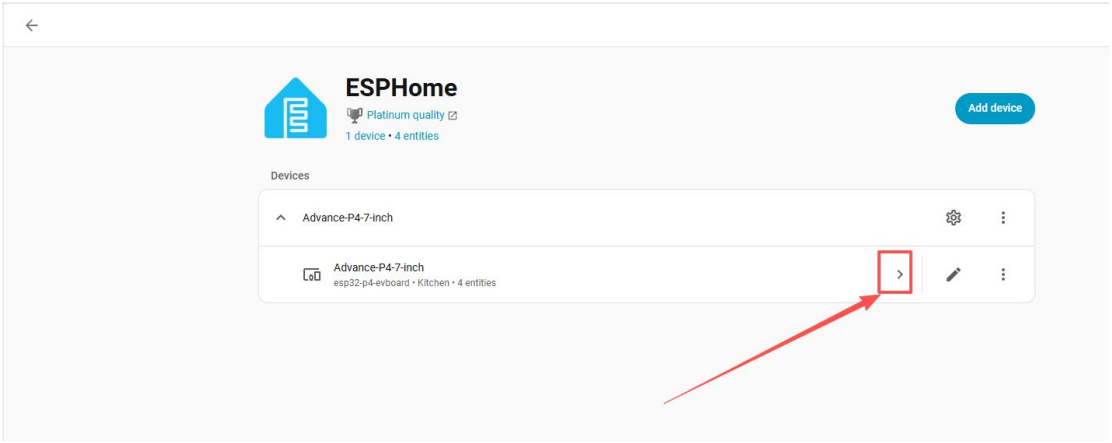
After completion, go to settings and select devices

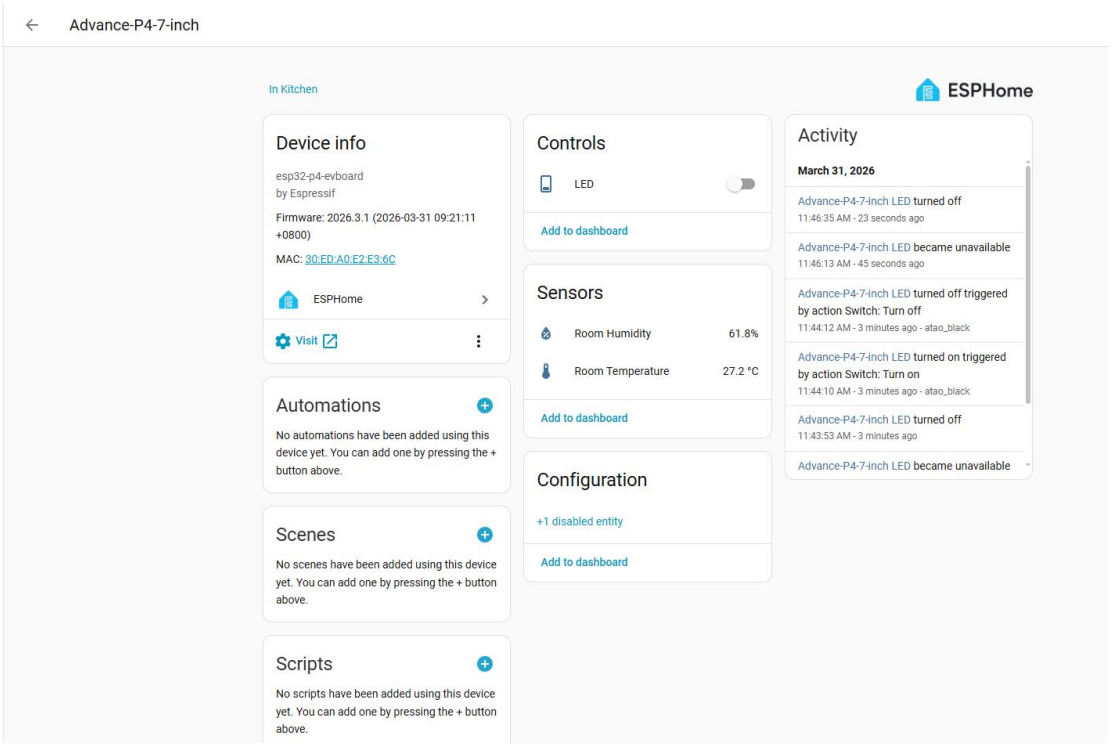


Choose ESPHome

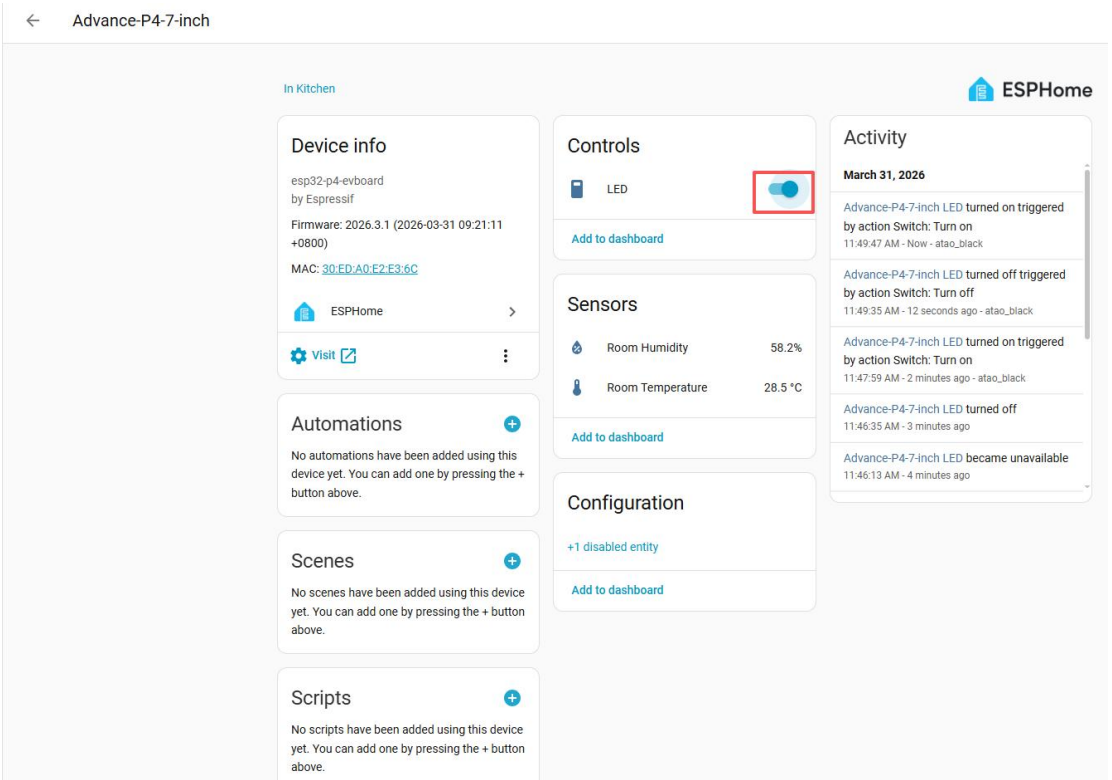


Here you can see the specific details of the equipment we just added.



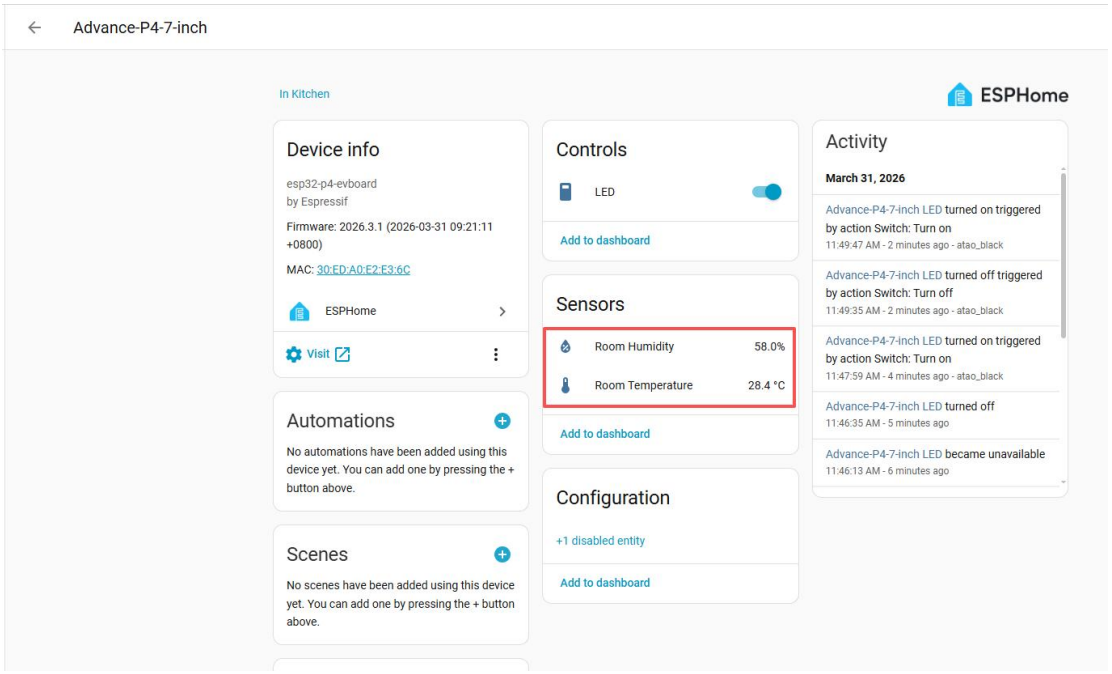


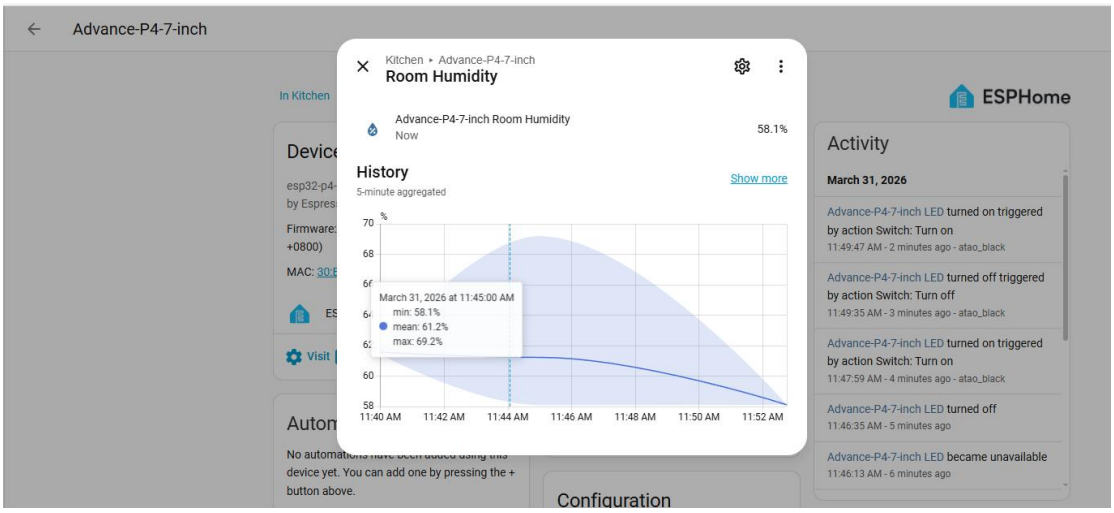
Next, when you click the switch of the LED, you will be able to see that the indicator light on the screen turns on, and the feedback from the LED is also quite obvious.

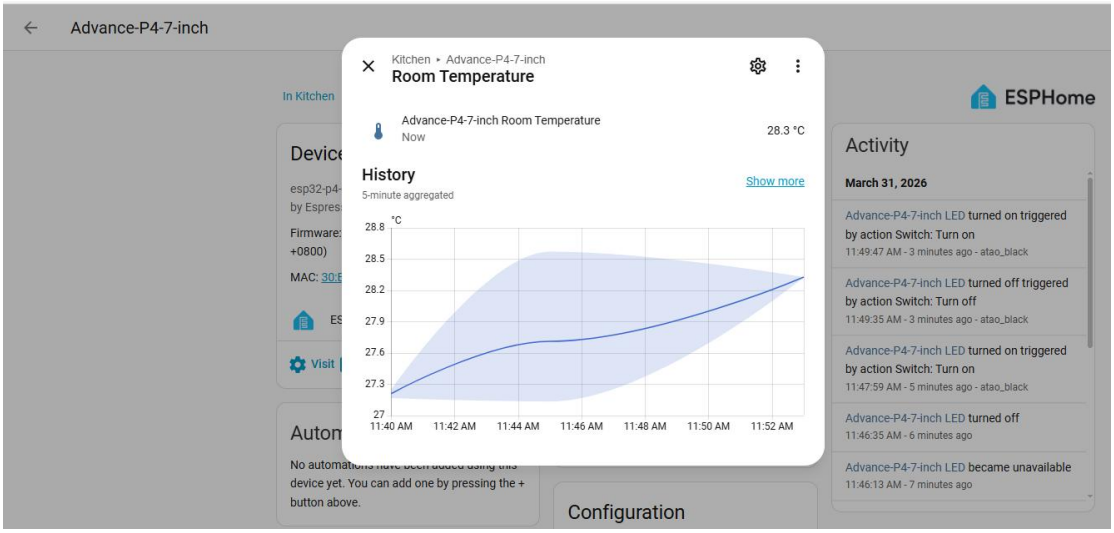


Next, when you click the switch of the LED, you will be able to see that the display light on the screen turns off, and the feedback from the LED is also quite obvious.

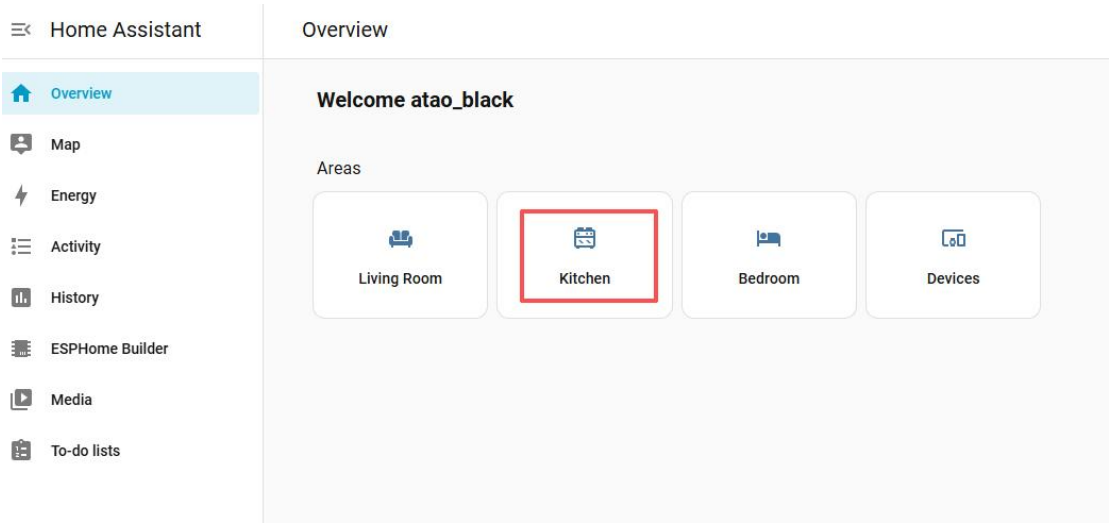
And you can also see the historical data of temperature and humidity collection from here.







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



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