

CrowPanel Advanced 7inch/9inch/10.1inch ESP32-P4 HMI

--- ESPHome Tutorial (V1.2)

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.

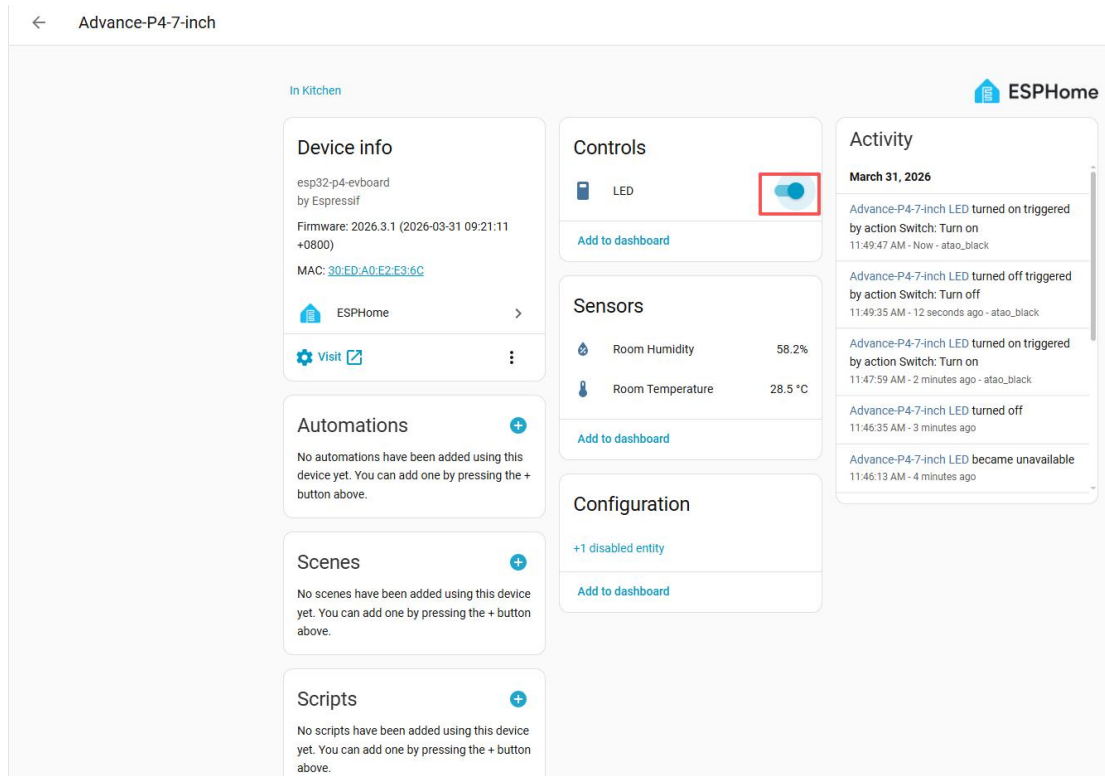
So let's move on to learn how to install the ESPHome environment, allowing you to edit code on it to achieve reading temperature and humidity data and remotely control the on and off of LEDs.

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.

← Advance-P4-7-inch

In Kitchen

Device info

esp32-p4-evboard
by Espressif

Firmware: 2026.3.1 (2026-03-31 09:21:11 +0800)

MAC: 30:ED:A0:F2:F3:6C

ESPHome

Visit

Controls

LED ☒

Add to dashboard

Sensors

Room Humidity	58.0%
Room Temperature	28.4 °C

Add to dashboard

Configuration

+1 disabled entity

Add to dashboard

Activity

March 31, 2026

Advance-P4-7-inch LED turned on triggered by action Switch: Turn on
11:49:47 AM - 2 minutes ago - atao_black

Advance-P4-7-inch LED turned off triggered by action Switch: Turn off
11:49:35 AM - 2 minutes ago - atao_black

Advance-P4-7-inch LED turned on triggered by action Switch: Turn on
11:47:59 AM - 4 minutes ago - atao_black

Advance-P4-7-inch LED turned off
11:46:35 AM - 5 minutes ago

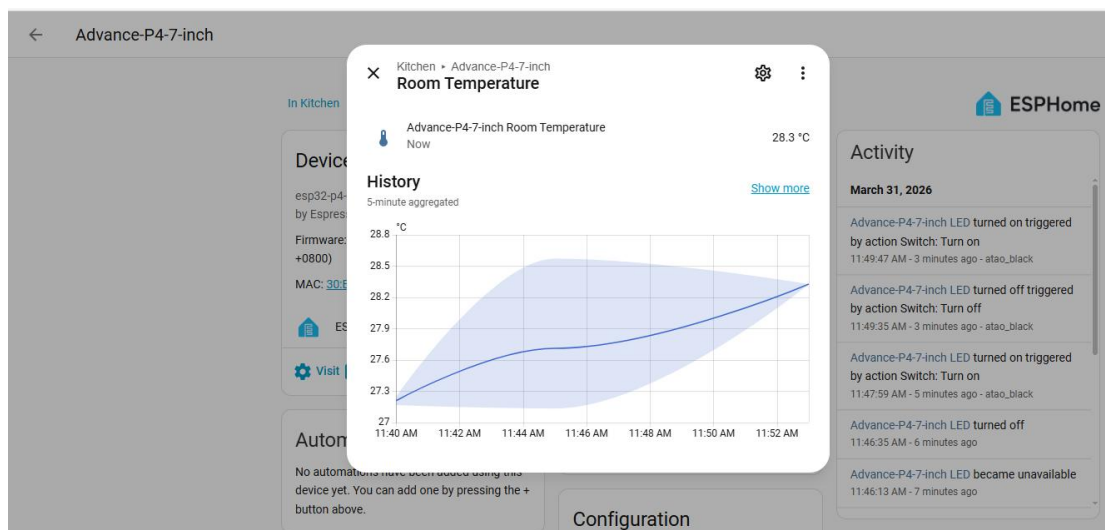
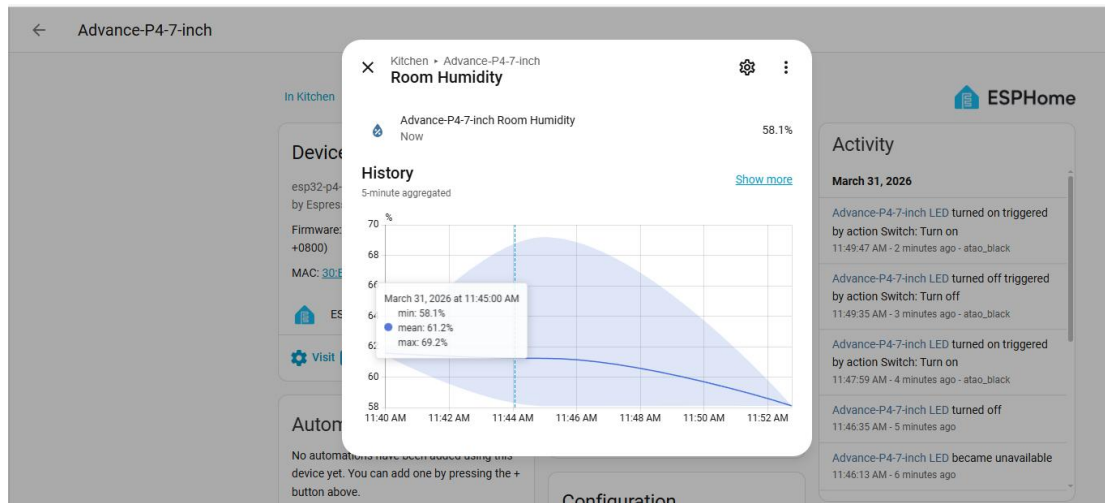
Advance-P4-7-inch LED became unavailable
11:46:13 AM - 6 minutes ago

Automations

No automations have been added using this device yet. You can add one by pressing the + button above.

Scenes

No scenes have been added using this device yet. You can add one by pressing the + button above.



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; (**Must use Raspberry Pi 5**)

a 64GB SD card;

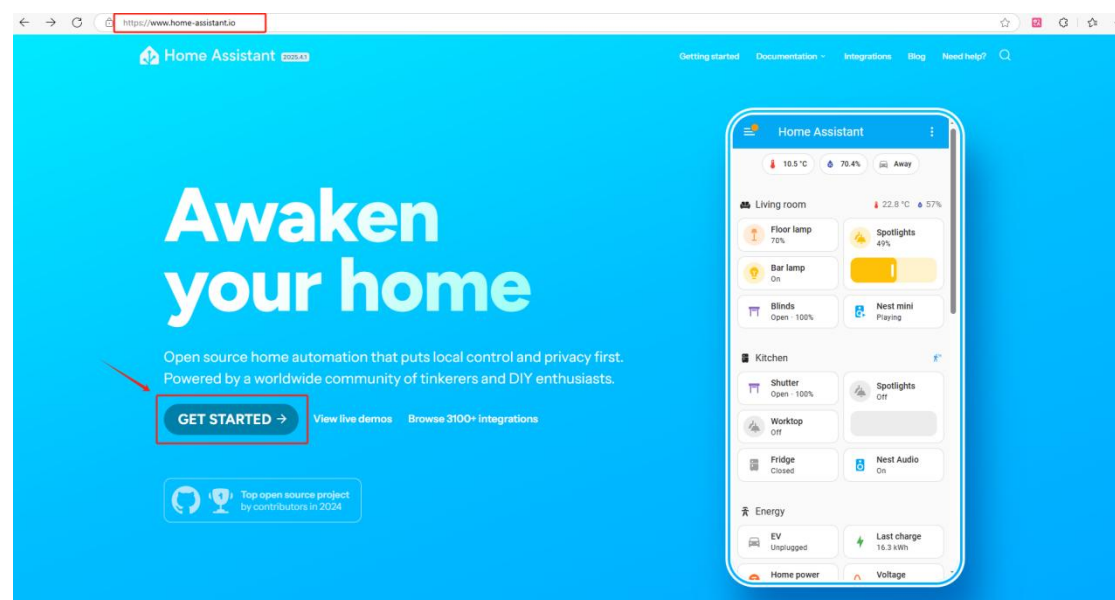
a CrowPanel Advanced ESP32-P4 HMI AI Display (7inch / 9inch / 10.1inch) ;

A humidity and temperature sensor;

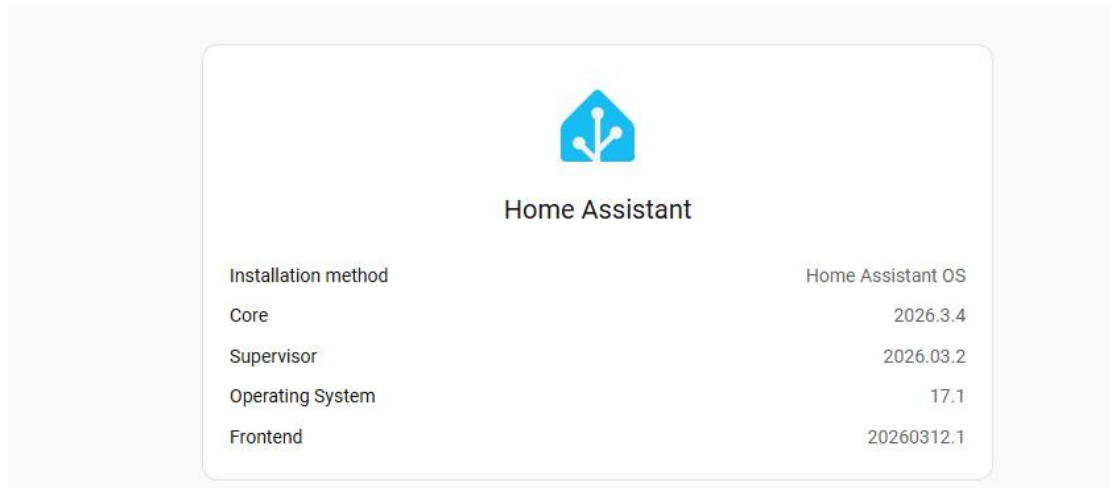
An LED light;

Download HomeAssistant

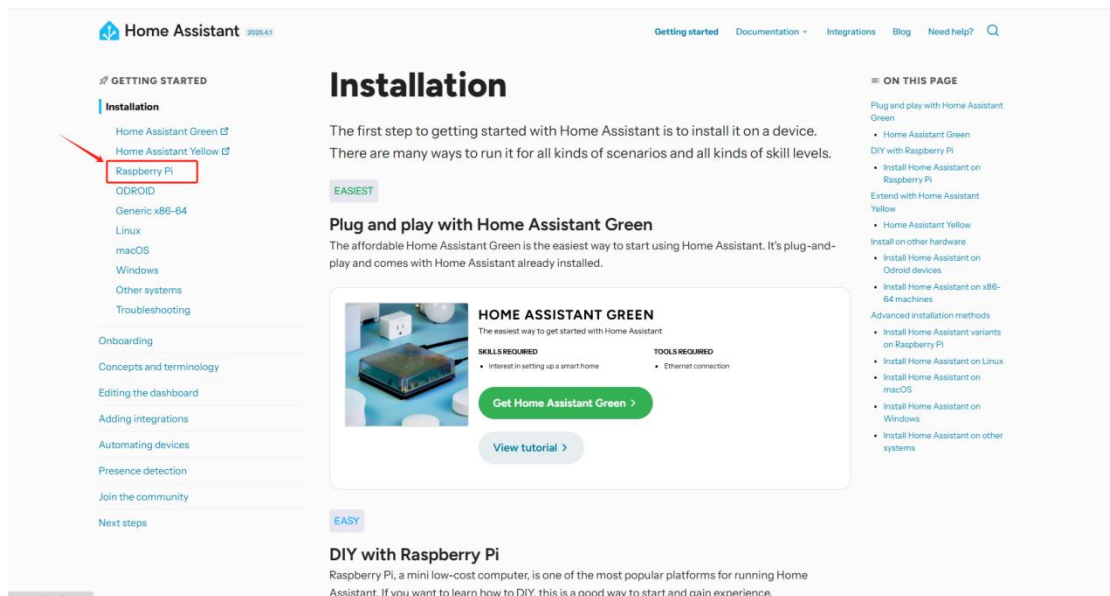
Open the official website of HomeAssistant: <https://www.home-assistant.io/>



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.)

```
Waiting for the Home Assistant CLI to be ready...

      _ _ _ _ _
     / /   / /
    / /   / /
   / /   / /
  / /   / /
 / /   / /
/ /   / /

Welcome to the Home Assistant command line.

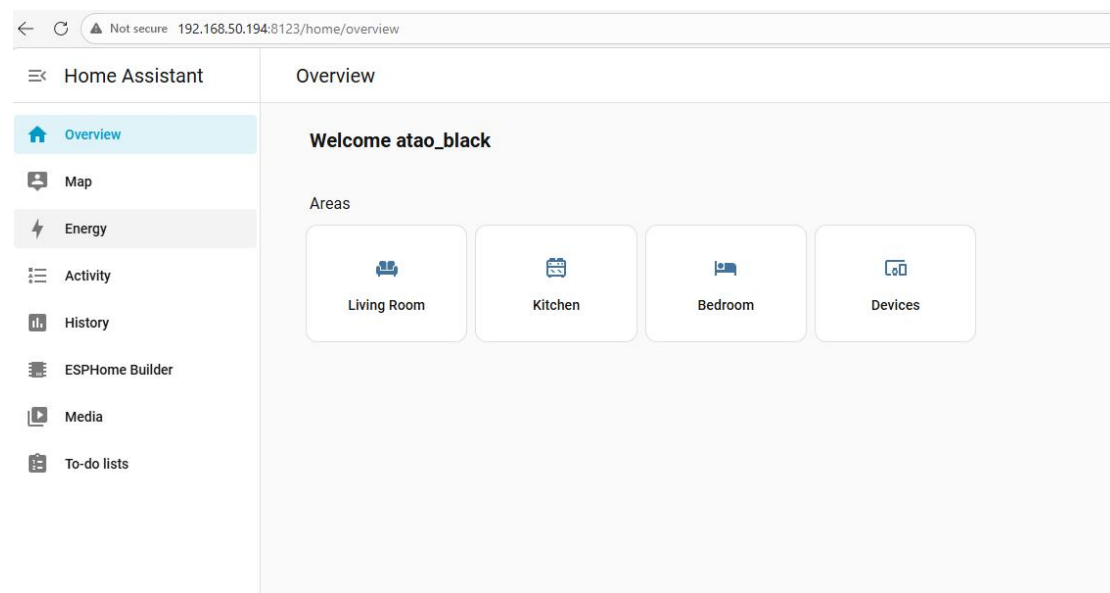
Waiting for Supervisor to startup...
System information
IPv4 addresses for enp1s0: 192.168.2.169, 24
IPv6 addresses for enp1s0: fe80::...

OS Version:      Home Assistant OS
Home Assistant Core:
Home Assistant URL:  http://baker.local:8123
Observer URL:      http://baker.local:4357
```

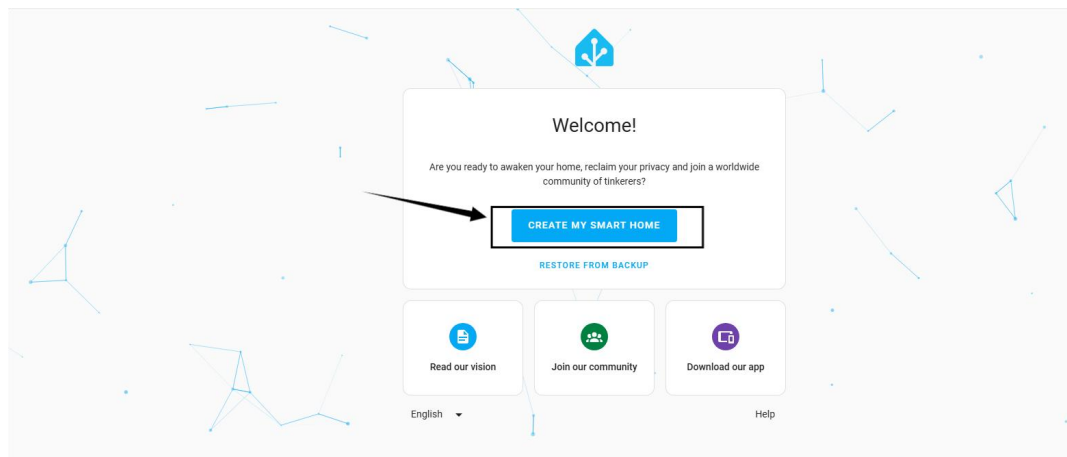
Remember the IP address and the Home Assistant URL.

In my case, the IP address is: 192.168.50.194

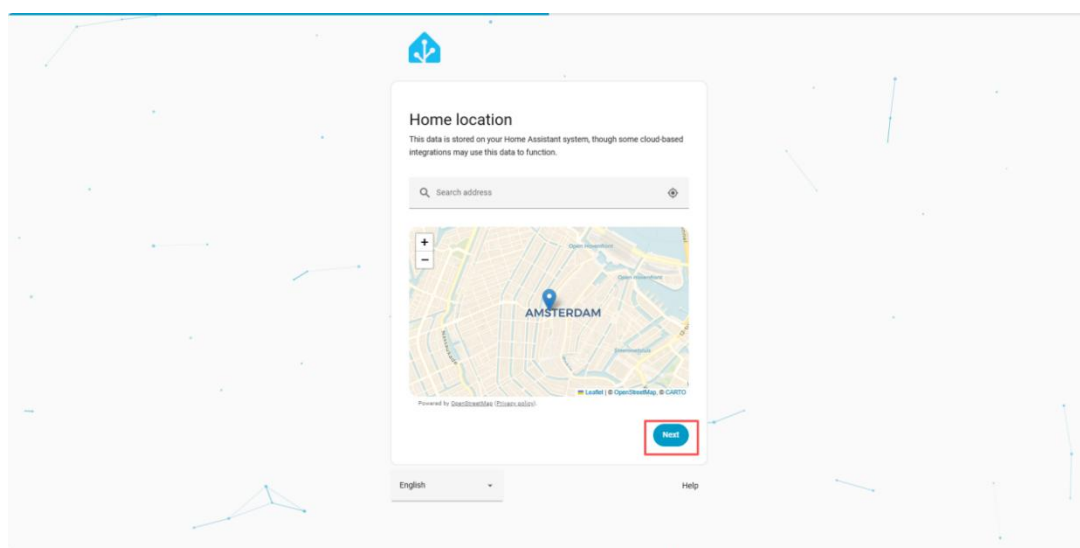
So I enter 192.168.50.194:8123 in the browser.



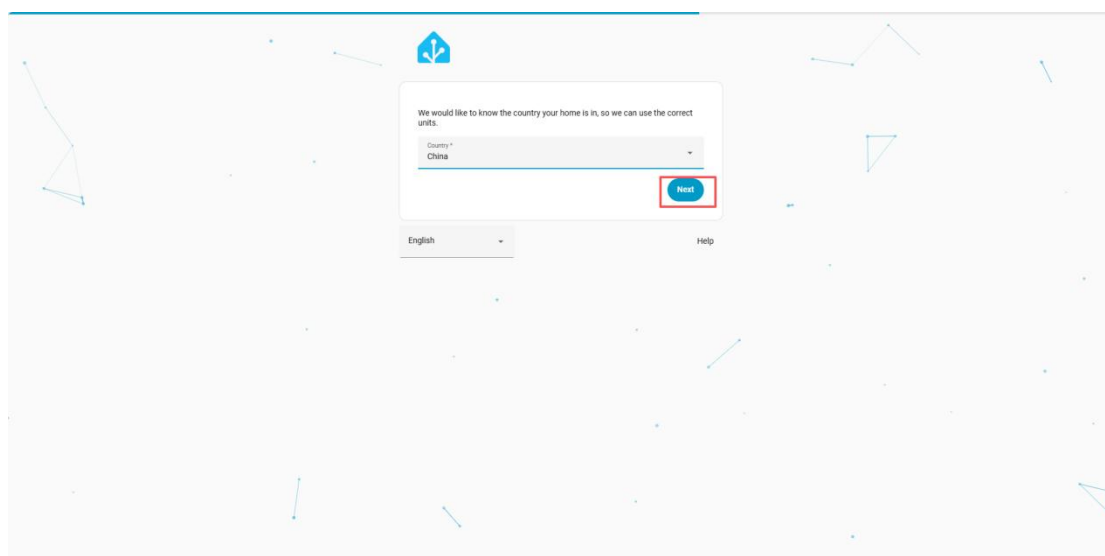
Next, start configuring the Home Assistant system.



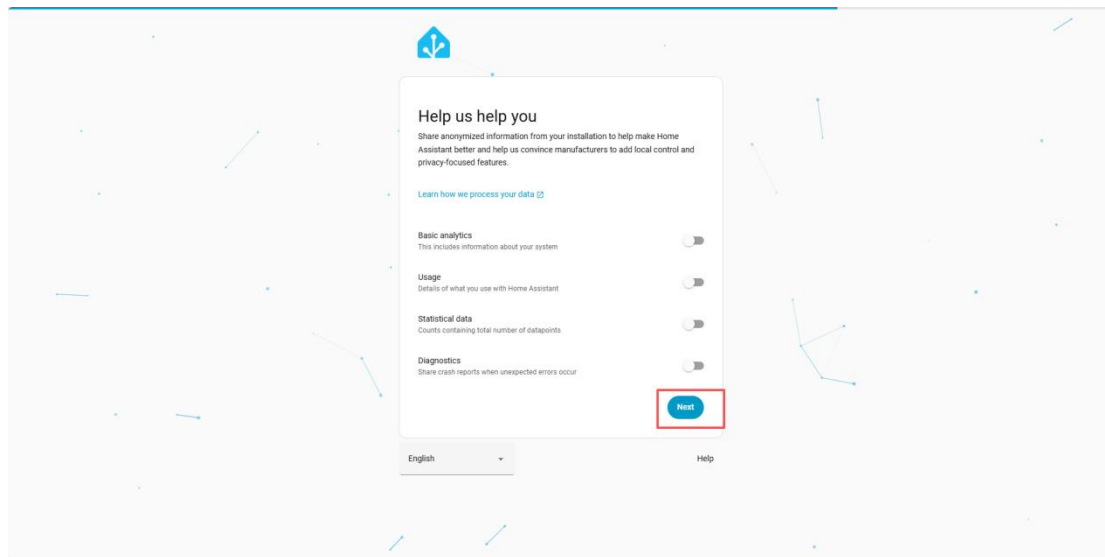
You can either manually set your location or allow it to be detected automatically.



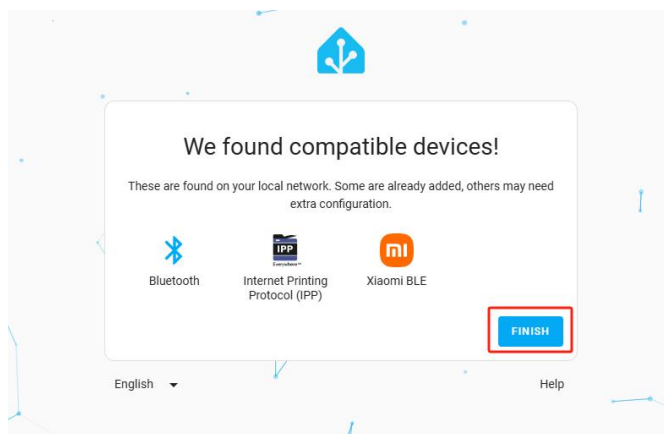
Set the country where you are located.



Keep it as default here.

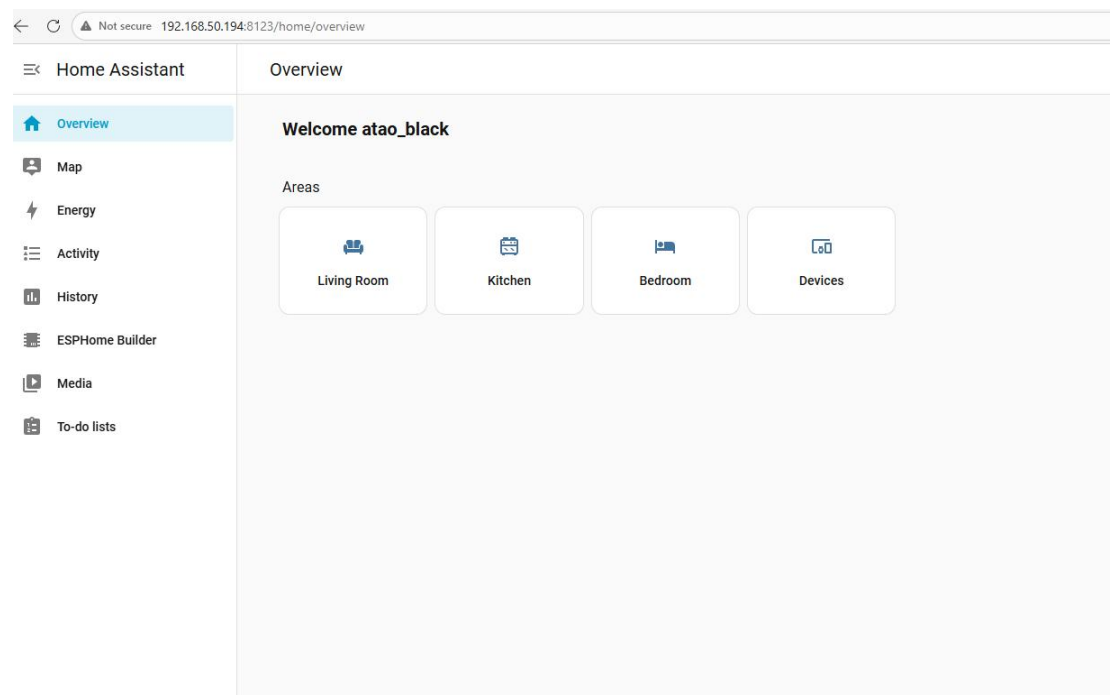


You can either add smart devices now or click **Finish** to add them later.
Here, we will add the devices later.

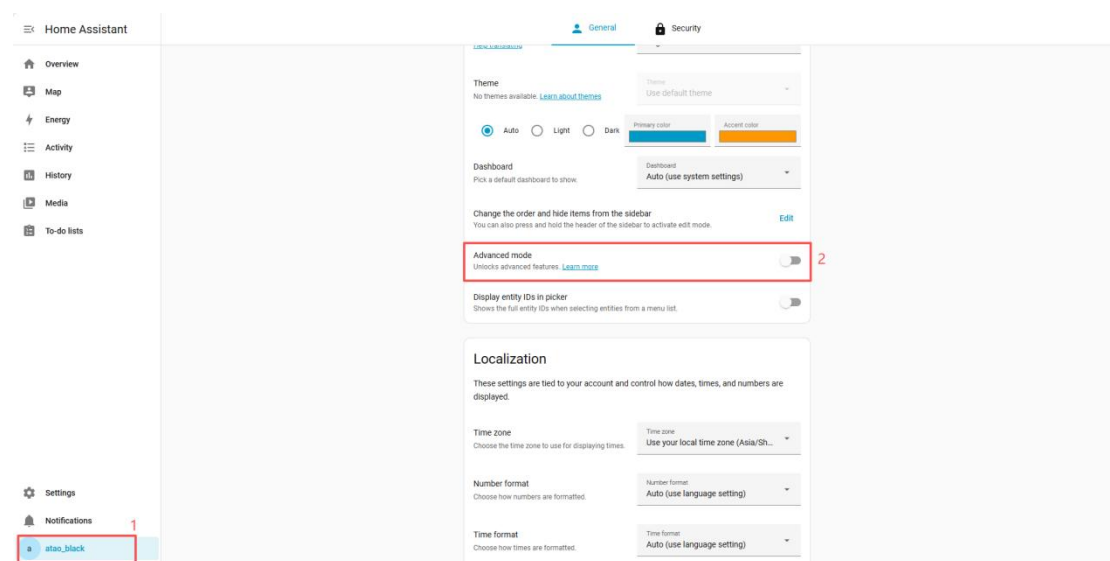


Add ESPHome

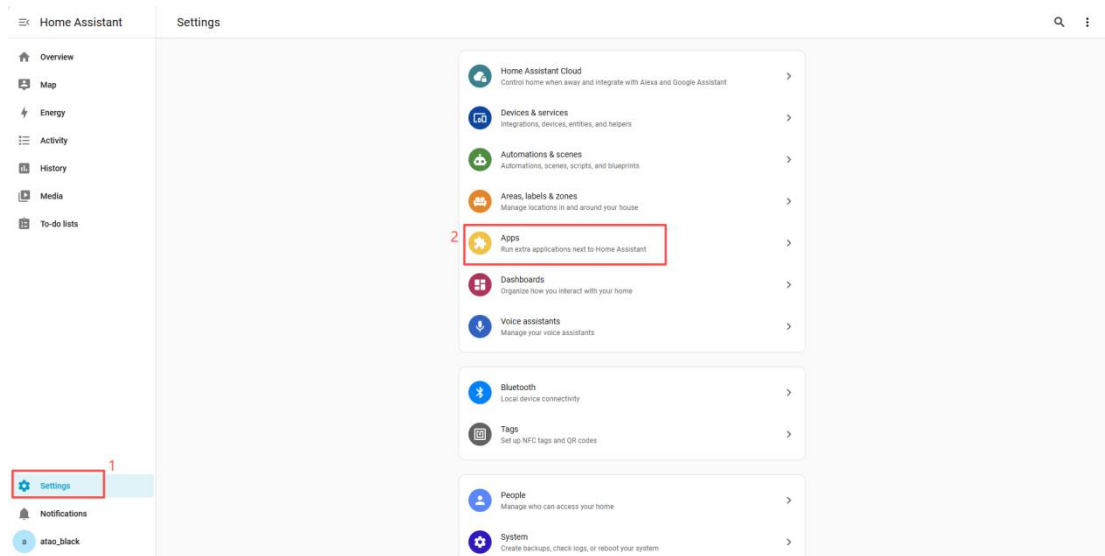
This will bring you to the main interface of Home Assistant.



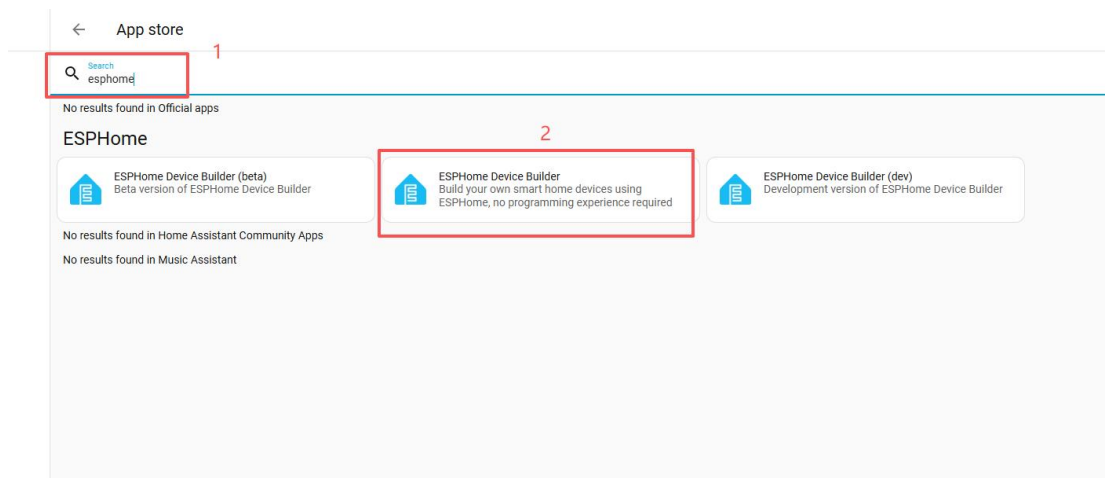
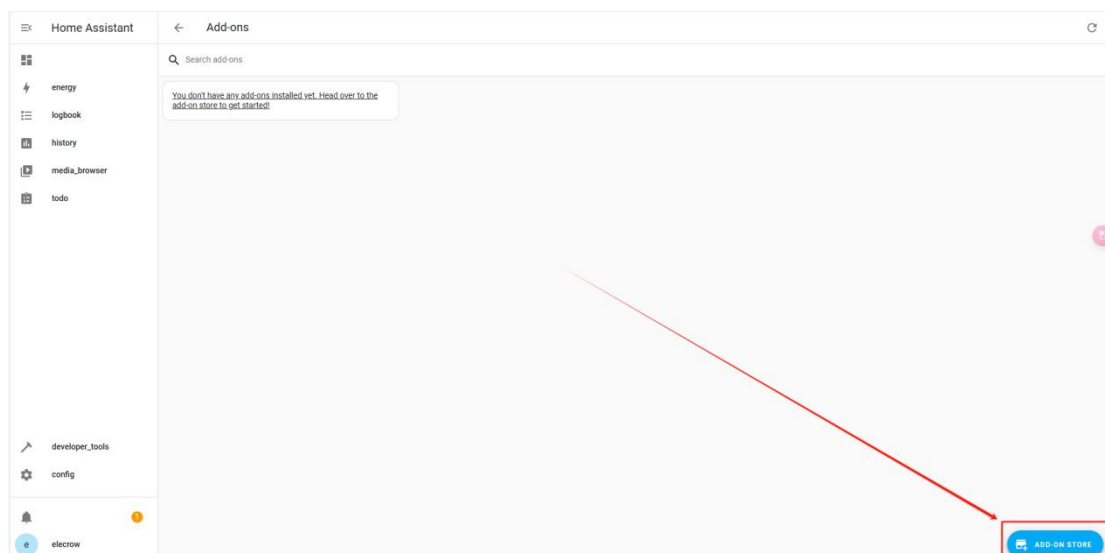
Click on the username and enable "Advanced Mode."



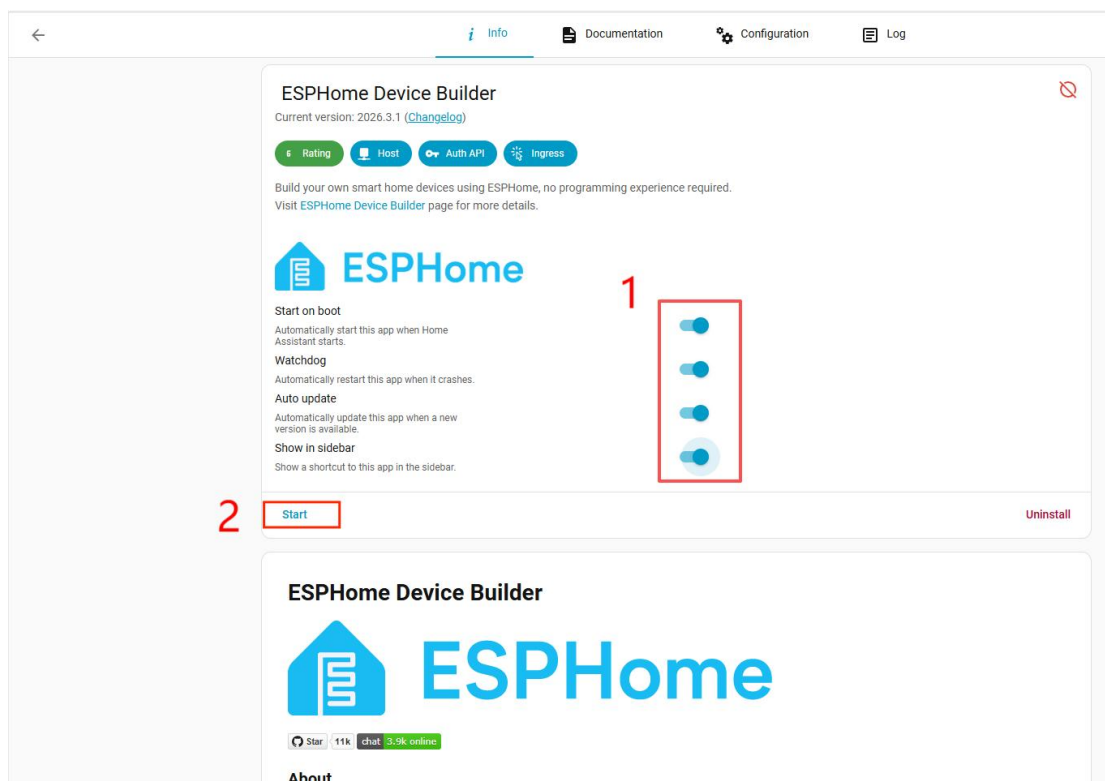
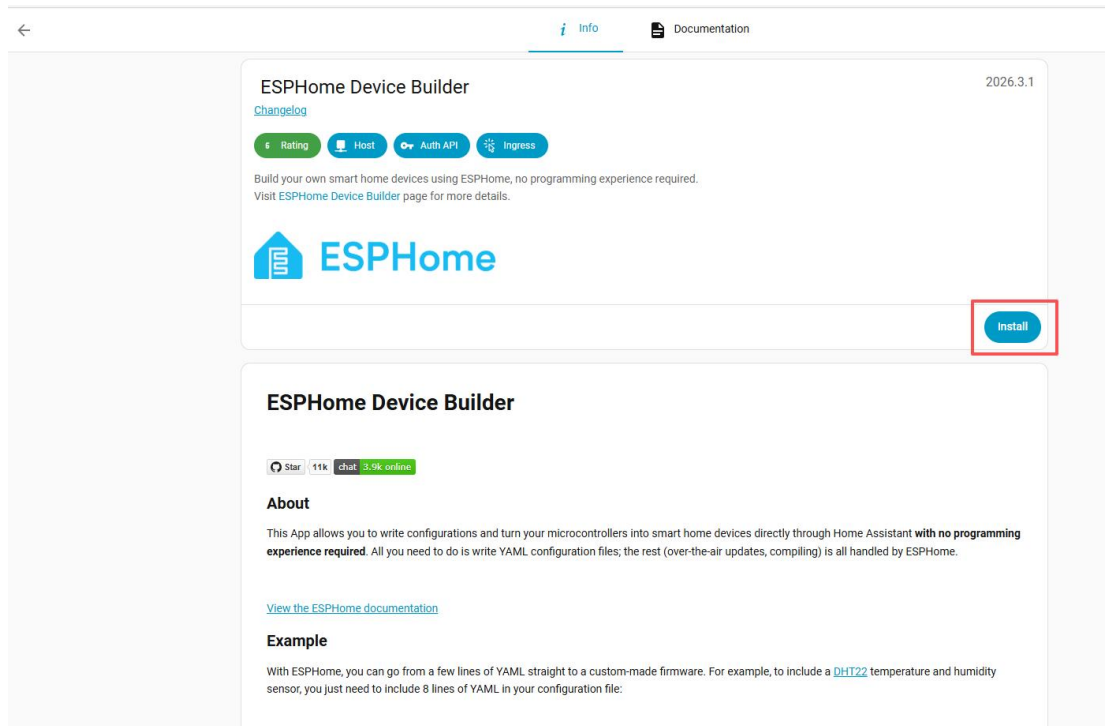
Click on "Settings" and then "Apps."



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



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

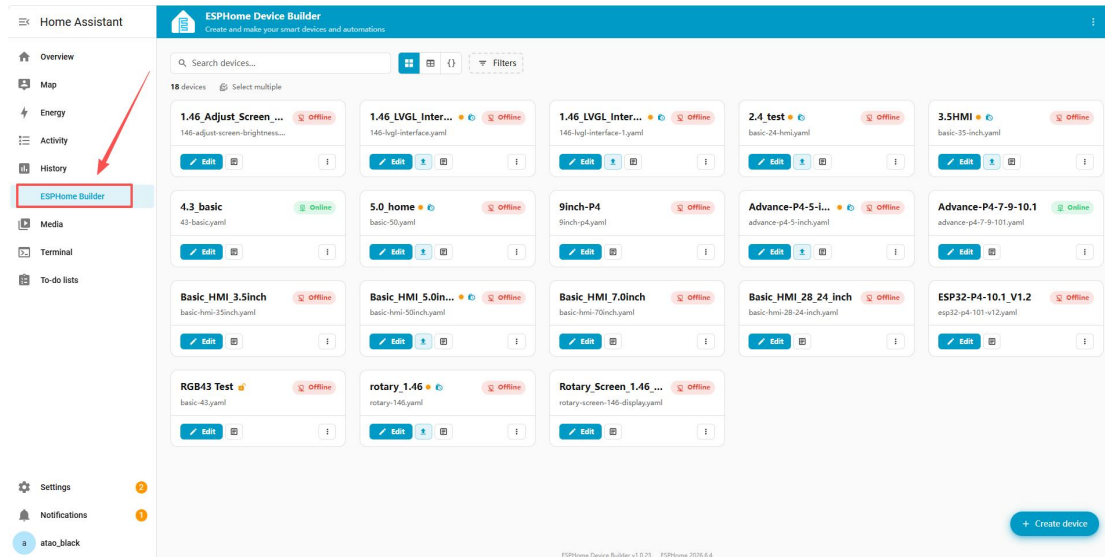


So we have successfully installed ESPHome. From now on, we can use ESPHome to edit the code and achieve the functions we want.

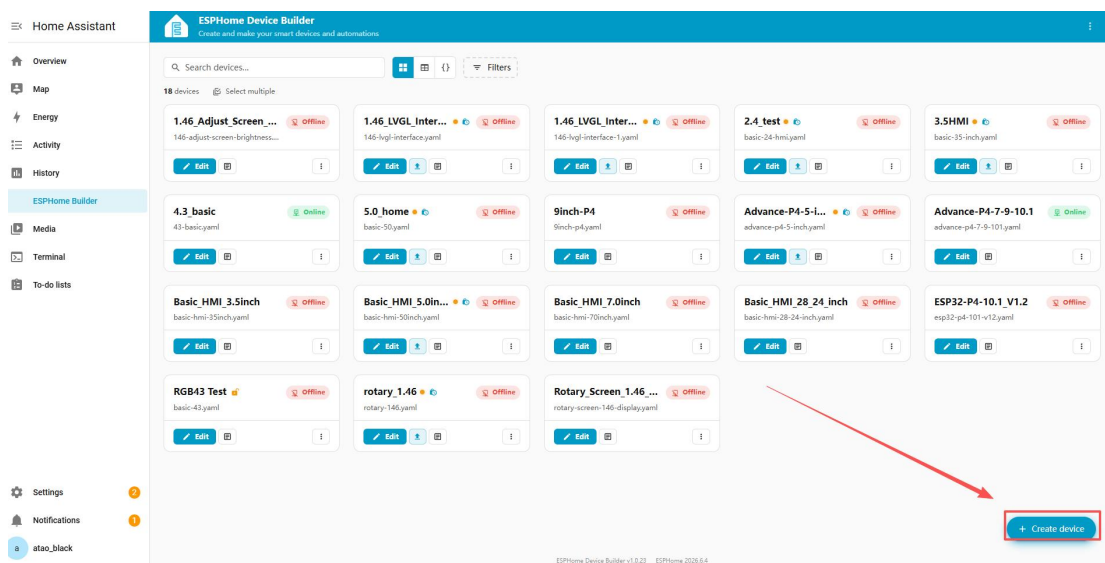
6. New Project

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

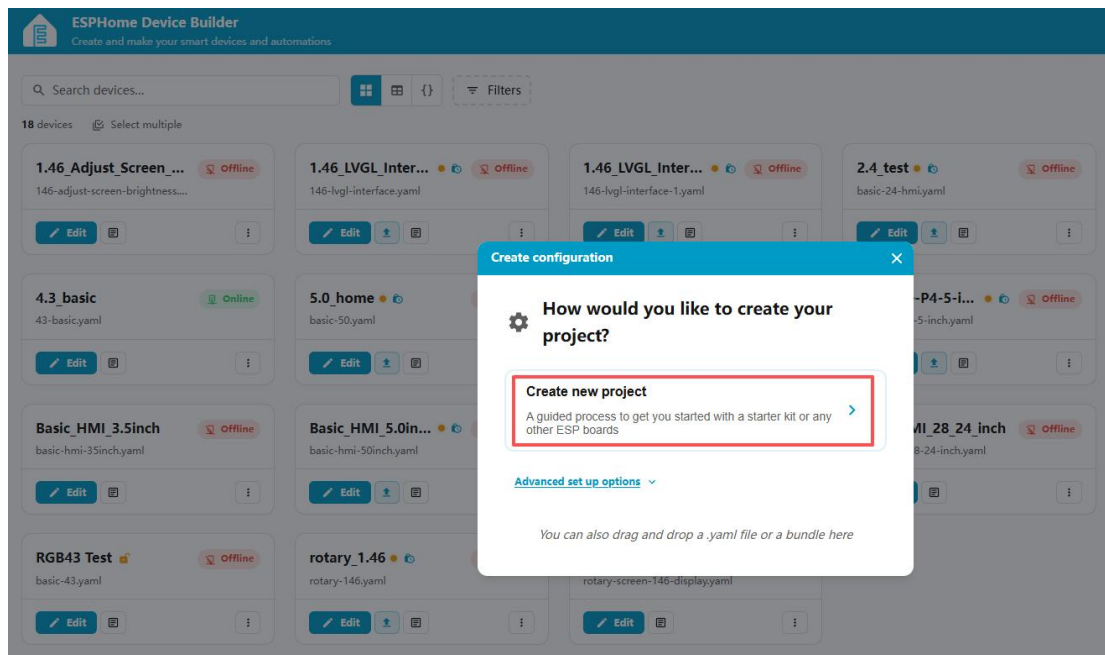
Come to ESPHome Builder



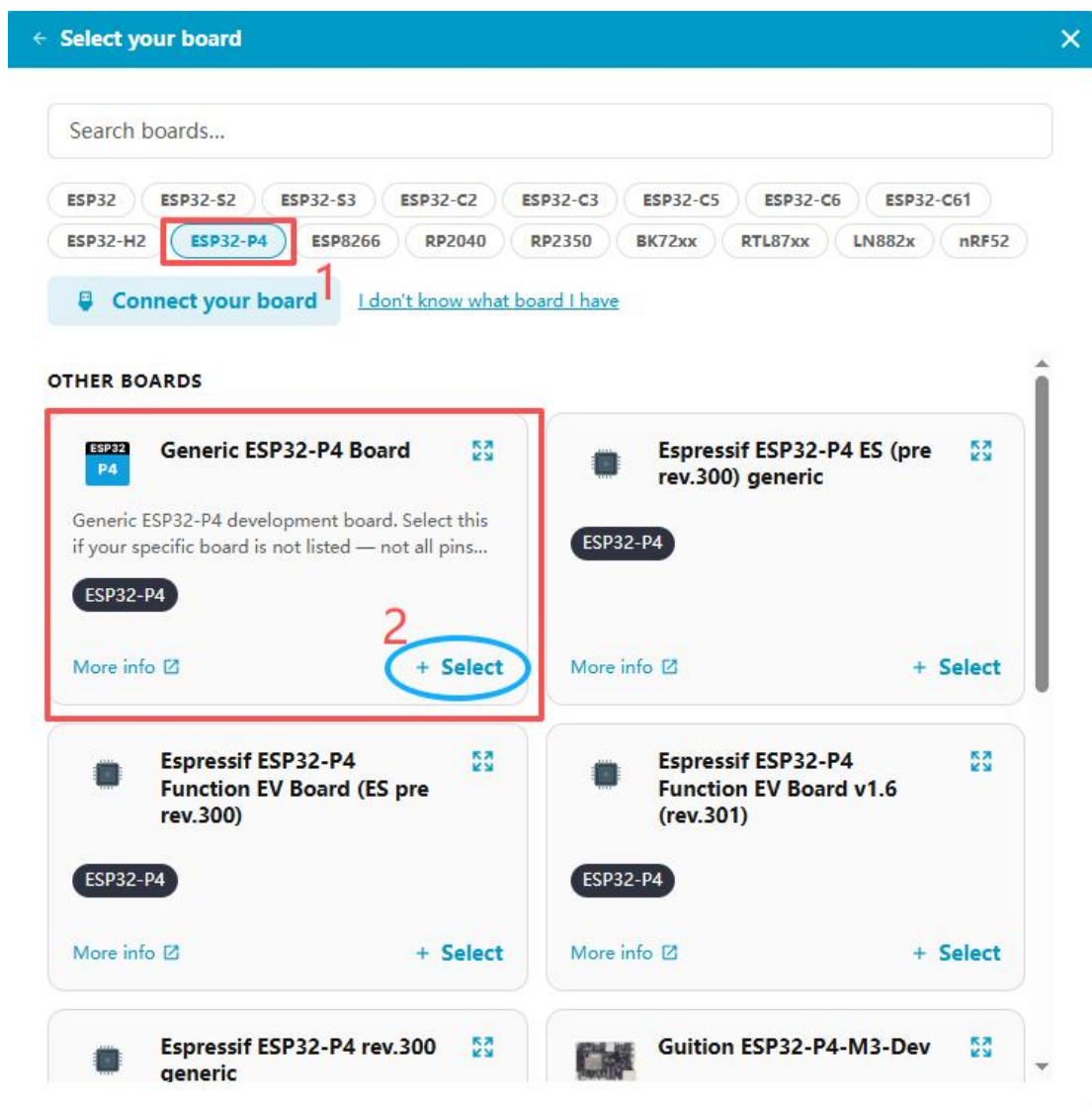
Click to add a new device



Click to create



Choose the **ESP32-P4** chip used in our project



Set a name for this project.

← Set up your ESPHome board

×

Generic ESP32-P4 Board

ESP32

P4

Name your device

Naming your project will show up in the ESPHome dashboard. Choose a recognisable, descriptive and memorable name. You can use component names or relevant words. You can edit this after you finish this project.

Device name

1

Advance-P4-7inch

Back

Finish setup

After the configuration is completed, the system will present a graphical configuration interface. This textbook will mainly guide you to learn how to write the programs in the right code column. In the subsequent operations, you will find that the final effect achieved by manually writing the code and operating through the graphical interface is completely the same.

ESPHome Device Builder

Create and make your smart devices and automations.

Device navigator

Navigate, add and remove options to define your ESP device and its functionality.

Core configuration

Components

Automations

Editing Advance-P4-7inch advance-p4-7inch.yaml

Generic ESP32-P4 Board

ESP32

P4

Generic ESP32-P4 development board. Select this if your specific board is not listed — not all pins may be physically accessible on your board. Note that the ESP32-P4 has no built-in Wi-Fi or Bluetooth and typically requires a co-processor for wireless connectivity.

Congratulations, you just configured your Generic ESP32-P4 Board!

A minimal default configuration has been applied. From here you can refine the core settings, add components like sensors or lights, and set up automations. Use the Device navigator on the left or the panels below to get started.

1. Core configuration

The core configuration components are the base building blocks that let your board run ESPHome. They cover settings such as the device name, Wi-Fi or Ethernet network, OTA updates, the API connection to Home Assistant, and optional add-ons like MQTT. Use the Device navigator on the left to view, edit or add core components.

Show core configuration

2. Components

Components are the building blocks that bring your device to life — sensors, switches, lights, displays, buses, and more. Each one represents a piece of hardware on your board or a virtual entity exposed to Home Assistant. Use the Device navigator on the left to view, edit or add components.

Show components

3. Automations

Automations let your device react to events on its own — turning on a light when a button is pressed, sending a notification when a sensor changes, scheduling tasks, and so on. They run locally on the device. Use the Device navigator on the left to view, edit or add automations.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

465

466

467

468

469

470

471

472

473

474

475

476

477

478

479

480

481

482

483

484

485

486

487

488

489

490

491

492

493

494

495

496

497

498

499

500

501

502

503

504

505

506

507

508

509

510

511

512

513

514

515

516

517

518

519

520

521

522

523

524

525

526

527

528

529

530

531

532

533

534

535

536

537

538

539

540

541

542

543

544

545

546

547

548

549

550

551

552

553

554

555

556

557

558

559

560

561

562

563

564

565

566

567

568

569

570

571

572

573

574

575

576

577

578

579

580

581

582

583

584

585

586

587

588

589

590

591

592

593

594

595

596

597

598

599

600

601

602

603

604

605

606

607

608

609

610

611

612

613

614

615

616

617

618

619

620

621

622

623

624

625

626

627

628

629

630

631

632

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648

649

650

651

652

653

654

655

656

657

658

659

660

661

662

663

664

665

666

667

668

669

670

671

672

673

674

675

676

677

678

679

680

681

682

683

684

685

686

687

688

689

690

691

692

693

694

695

696

697

698

699

700

701

702

703

704

705

706

707

708

709

710

711

712

713

714

715

716

717

718

719

720

721

722

723

724

725

726

727

728

729

730

731

732

733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

810

811

812

813

814

815

816

817

818

819

820

821

822

823

824

825

826

827

828

829

830

831

832

833

834

835

836

837

838

839

840

841

842

843

844

845

846

847

848

849

850

851

852

853

854

855

856

857

858

859

860

861

862

863

864

865

866

867

868

869

870

871

872

873

874

875

876

877

878

879

880

881

882

883

884

885

886

887

888

889

890

891

892

893

894

895

896

897

898

899

900

901

902

903

904

905

906

907

908

909

910

911

912

913

914

915

916

917

918

919

920

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972

973

974

975

976

977

978

979

980

981

982

983

984

985

986

987

988

989

990

991

992

993

994

995

996

997

998

999

1000

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

299

300

301

302

303

304

305

306

307

308

309

310

311

312

313

314

315

316

317

318

319

320

321

322

323

324

325

326

327

328

329

330

331

332

333

334

335

336

337

338

339

340

341

342

343

344

345

346

347

348

349

350

351

352

353

354

355

356

357

358

359

360

361

362

363

364

365

366

367

368

369

370

371

372

373

374

375

376

377

378

379

380

381

382

383

384

385

386

387

388

389

390

391

392

393

394

395

396

397

398

399

400

401

402

403

404

405

406

407

408

409

410

411

412

413

414

415

416

417

418

419

420

421

422

423

424

425

426

427

428

429

430

431

432

433

434

435

436

437

438

439

440

441

442

443

444

445

446

447

448

449

450

451

452

453

454

455

456

457

458

459

460

461

462

463

464

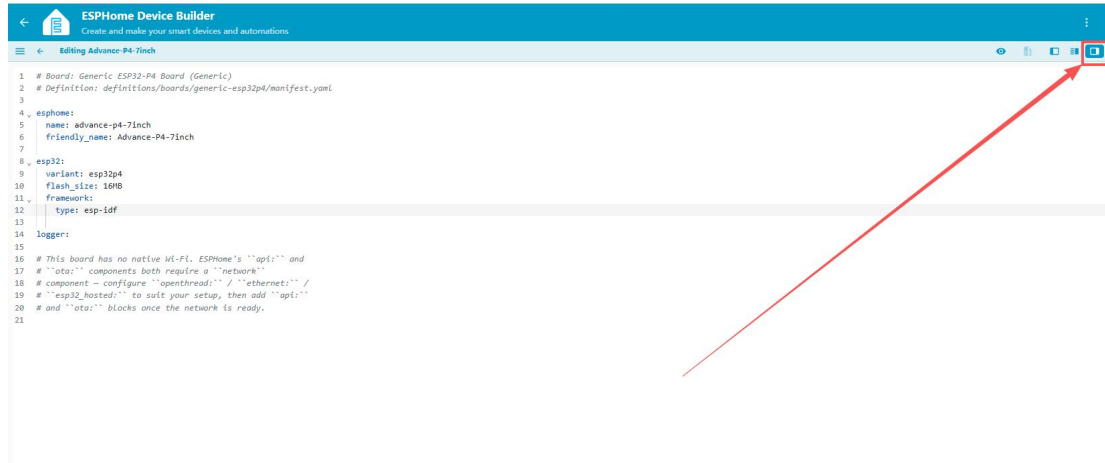
465

466

467

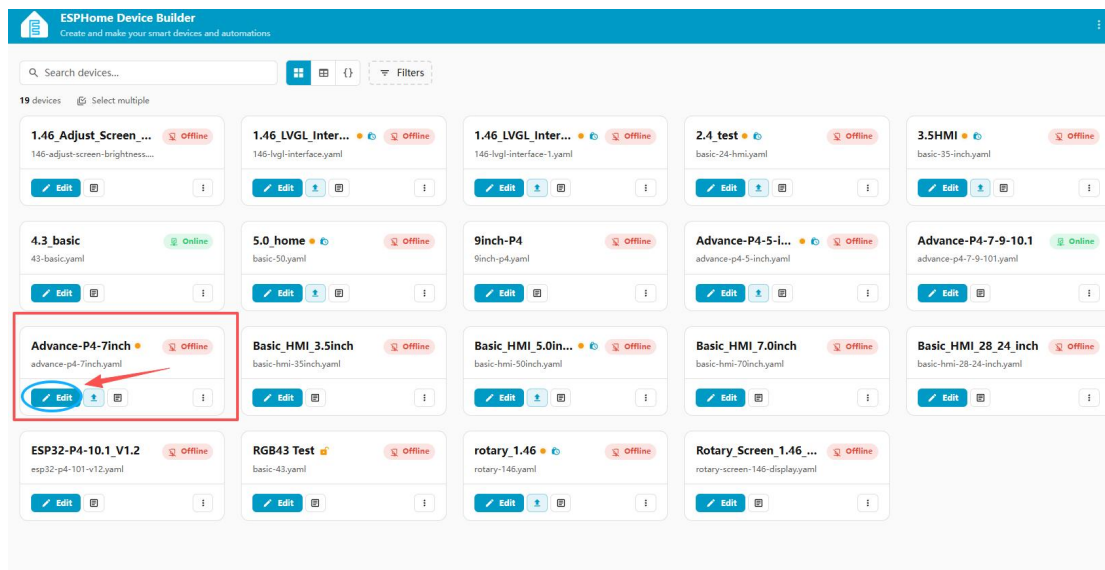
468

4



7. Configuration Code

Then, return to the main interface, find the [Advance-P4-7inch](#) 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.



ESPHome Device Builder
Create and make your smart devices and automations

Editing Advance-P4-7inch

```
1 # Board: Generic ESP32-P4 Board (Generic)
2 # Definition: definitions/boards/generic-esp32p4/manifest.yaml
3
4 esphome:
5   name: advance-p4-7inch
6   friendly_name: Advance-P4-7inch
7
8 esp32:
9   variant: esp32p4
10  flash_size: 16MB
11  framework:
12    type: esp-idf
13
14 logger:
15
16 # This board has no native Wi-Fi. ESPHome's ``api`` and
17 # ``ota`` components both require a ``network``
18 # component – configure ``openthread`` / ``ethernet`` /
19 # ``esp32_hosted`` to suit your setup, then add ``api``
20 # and ``ota`` blocks once the network is ready.
21
```

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/V1.2/ESPHome/Code>

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



ESPHome Device Builder
Create and make your smart devices and automations

Editing Advance-P4-7inch

```
1 # Board: Generic ESP32-P4 Board (Generic)
2 # Definition: definitions/boards/generic-esp32p4/manifest.yaml
3
4 esphome:
5   name: advance-p4-7inch
6   friendly_name: Advance-P4-7inch
7
8 esp32:
9   variant: esp32p4
10  flash_size: 16MB
11  framework:
12    type: esp-idf
13
14 logger:
15
16 # This board has no native Wi-Fi. ESPHome's ``api`` and
17 # ``ota`` components both require a ``network``
18 # component – configure ``openthread`` / ``ethernet`` /
19 # ``esp32_hosted`` to suit your setup, then add ``api``
20 # and ``ota`` blocks once the network is ready.
21
```

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.

```
Editing Advance-P4-7inch

1 # Board: Generic ESP32-P4 Board (Generic)
2 # Definition: definitions/boards/generic-esp32p4/manifest.yaml
3
4 esphome:
5   name: advance-p4-7inch
6   friendly_name: Advance-P4-7inch
7
8 esp32:
9   variant: esp32p4
10  engineering_sample: true
11  cpu_frequency: 360MHZ
12  flash_size: 16MB
13  framework:
14    type: esp-idf
15    advanced:
16      execute_from_psram: true
17      enable_idf_experimental_features: true
18
19 psram:
20   speed: 200MHz
21
22 logger:
23
24 # This board has no native Wi-Fi. ESPHome's ``api`` and
25 # ``ota`` components both require a ``network``
26 # component - configure ``openthread`` / ``ethernet`` /
27 # ``esp32_hosted`` to suit your setup, then add ``api``
28 # and ``ota`` blocks once the network is ready.
29
```

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.

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

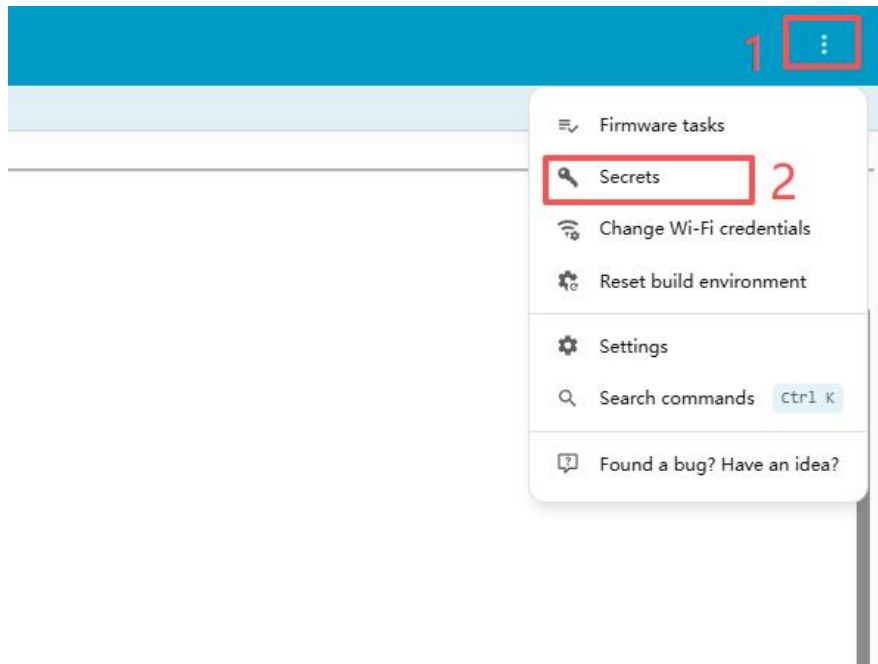
wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

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

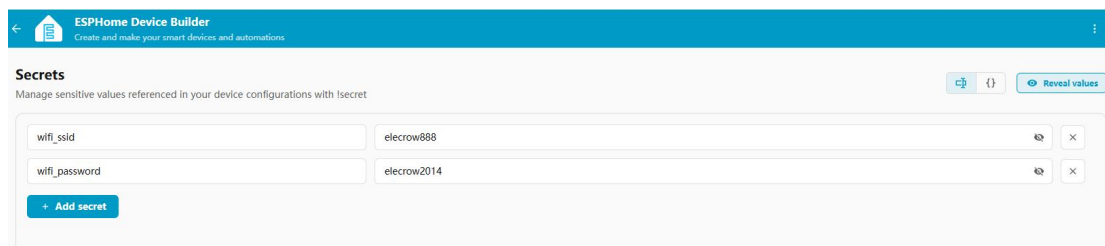
The WiFi account and password filled in here are all using the default settings.

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

Modifications can be made here.



Clicking on it will show this.



Just make sure that the network you set up is in the same local area network (LAN) as your computer and Raspberry Pi.

Further down is the API:

```
# This board has no native Wi-Fi. ESPHome's ``api:`` and  
# ``ota:`` components both require a ``network``  
# component – configure ``openthread:`` / ``ethernet:`` /  
# ``esp32_hosted:`` to suit your setup, then add ``api:``  
# and ``ota:`` blocks once the network is ready.  
# Enable Home Assistant API  
api:  
  encryption:  
    key: .....
```

"The **ESP32-P4** chip itself is a 'pure computing power monster,' and it doesn't integrate Wi-Fi or Bluetooth antennas at all. If you directly add `api:` and `ota:`, the compiler will report an error because it can't find the network driver. You must first configure the network card (such as Ethernet, OpenThread, or an external ESP32-C6 chip) to enable these two functions."

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.

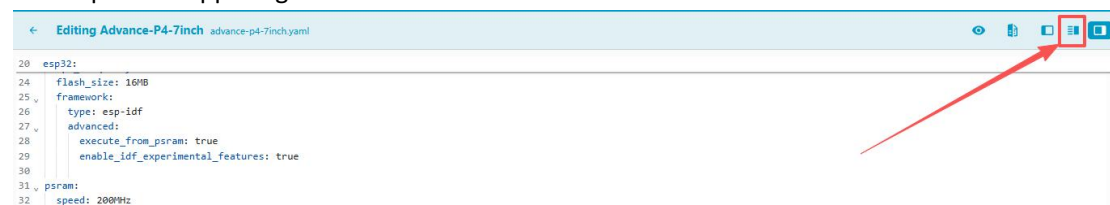
It can be understood as: a "remote control entry has been added to the device, and it is encrypted and secure".

This section is generated when creating the file and should not be modified.

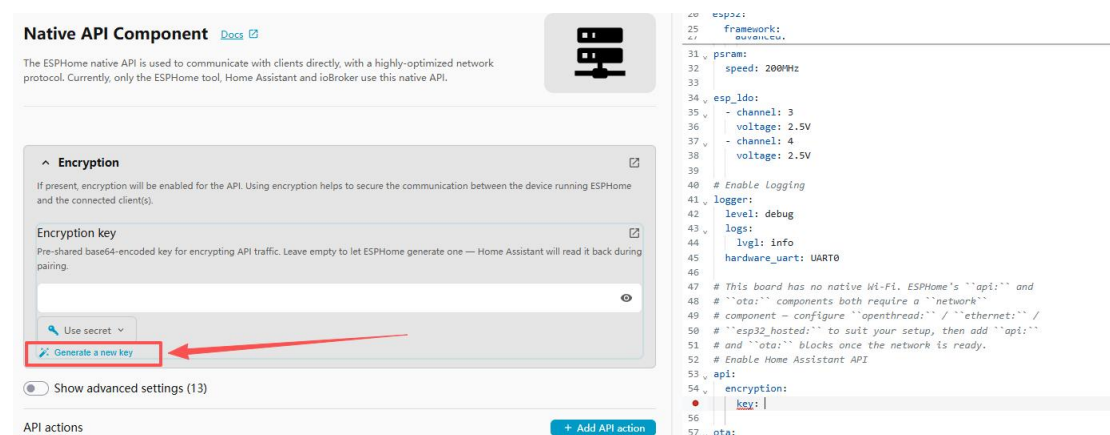
You can fill it in properly in the code.

```
53  api:
54  encryption:
    key:
```

Then open the upper right corner



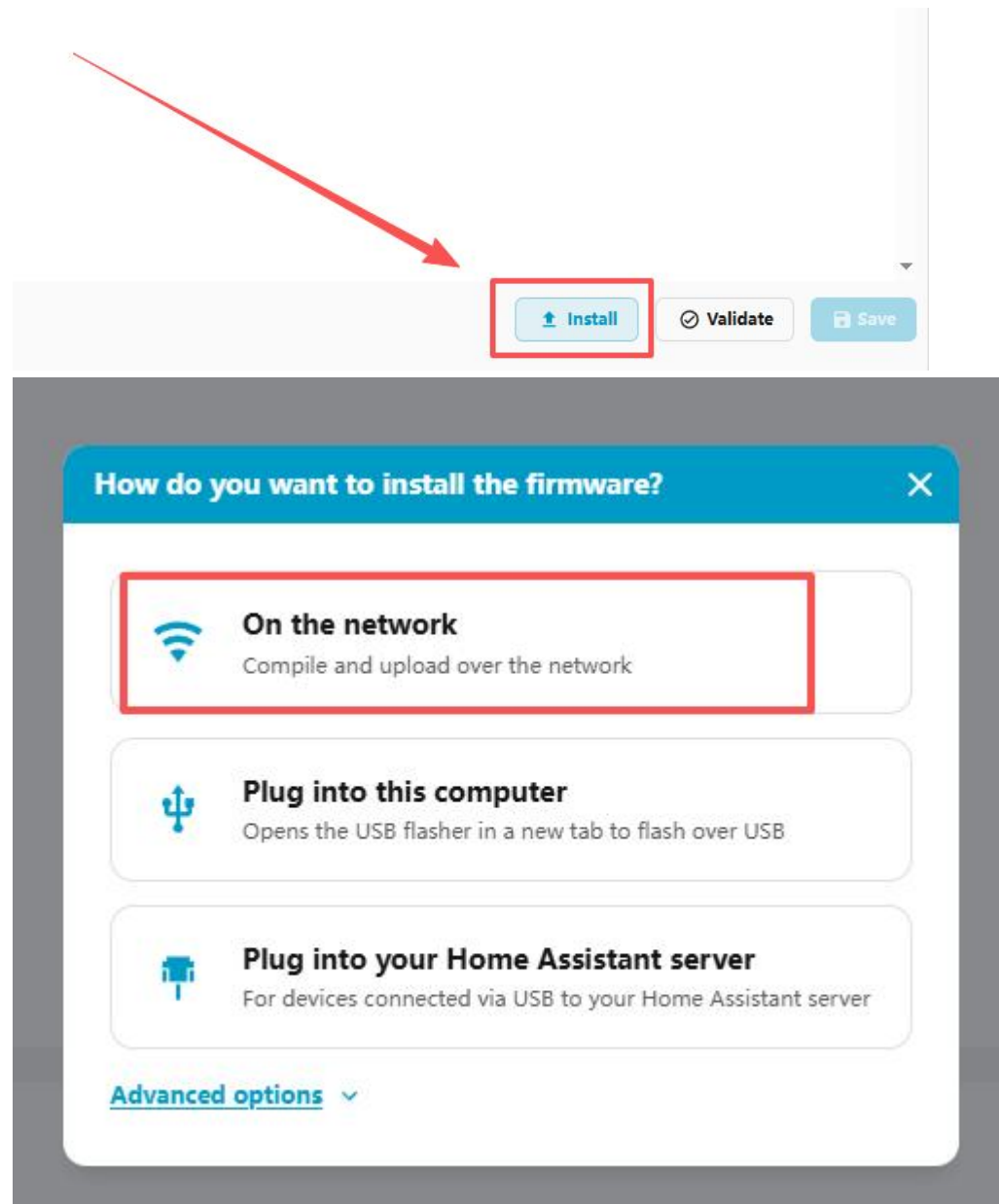
Generate a new key here again.



Finally, it's ota:

```
ota:
  - platform: esphome
    password: .....
```

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.)

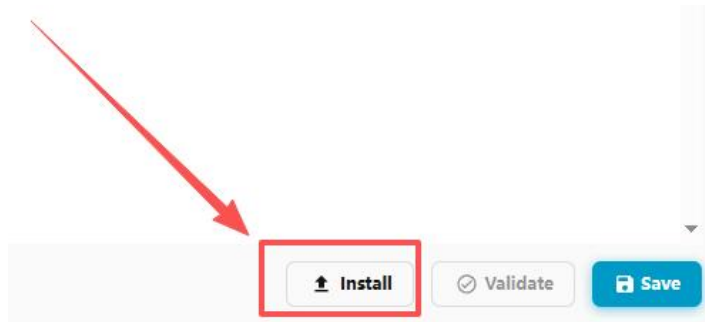
The OTA password of ESPHome is fully customizable. Essentially, it is just a plain text string that you can freely decide (it can be any combination of letters or numbers). The only rule is: when performing a wireless upgrade, the password filled in the new code must be exactly the same as the current running password of the device. Otherwise, the device will refuse to connect due to

"incorrect password".

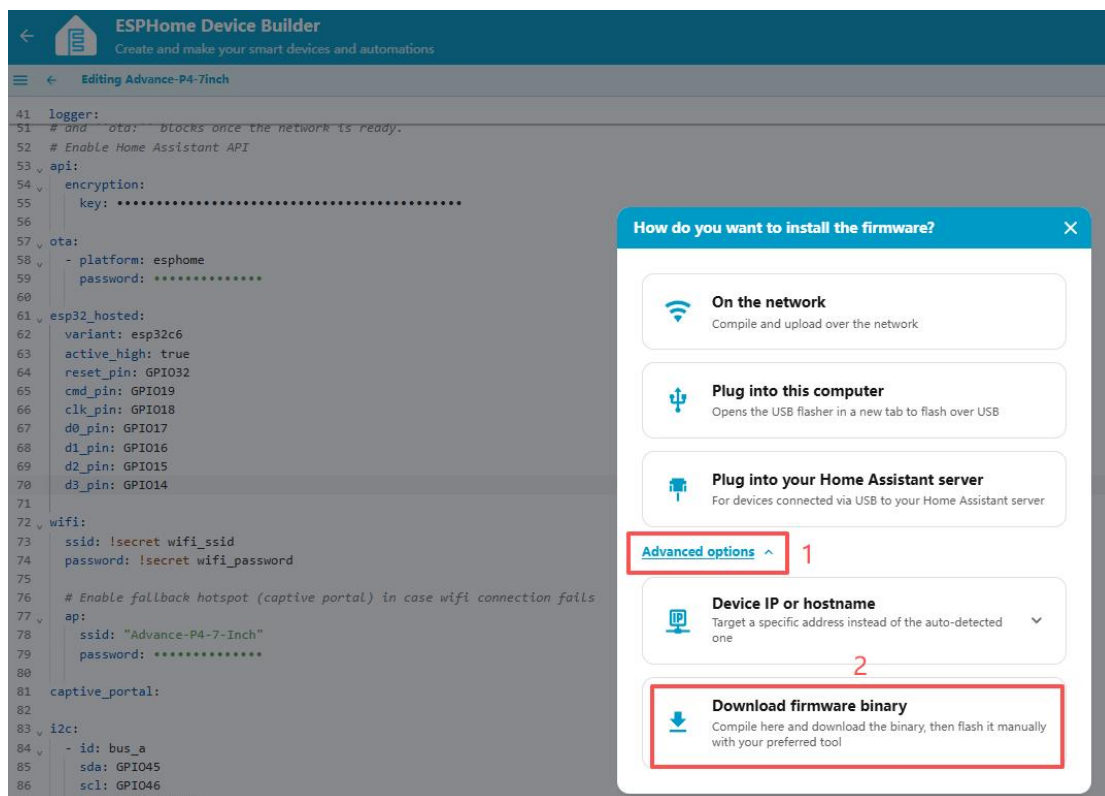
```
ota:  
- platform: esphome  
  password: "admin123456"
```

8. First upload of code

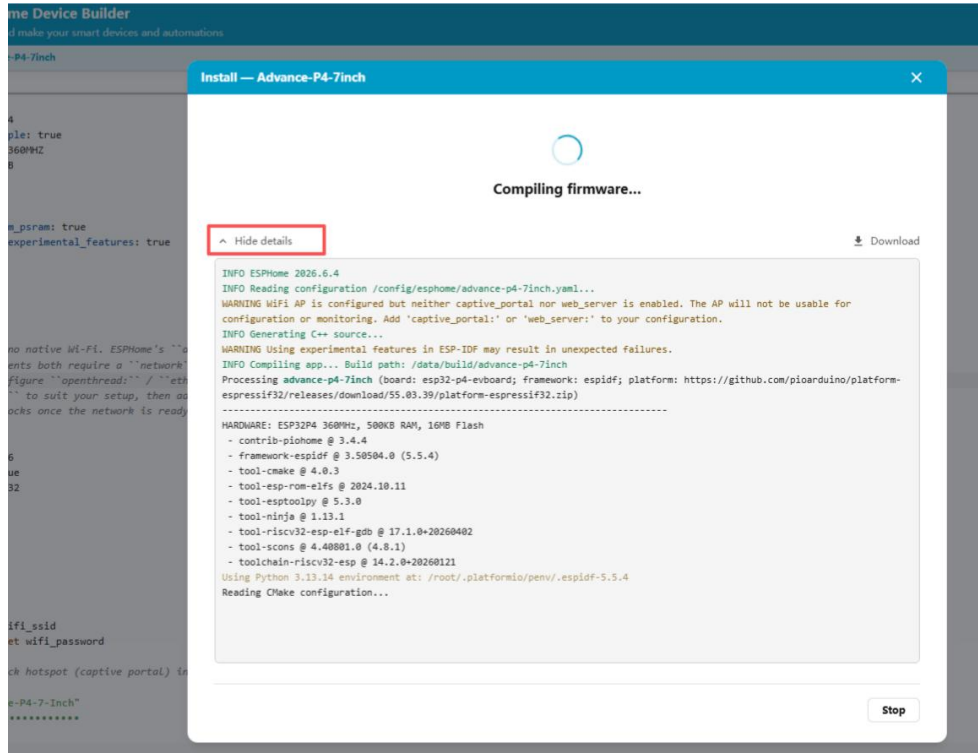
After the code replacement is completed, click the "Install" button at the bottom right corner.



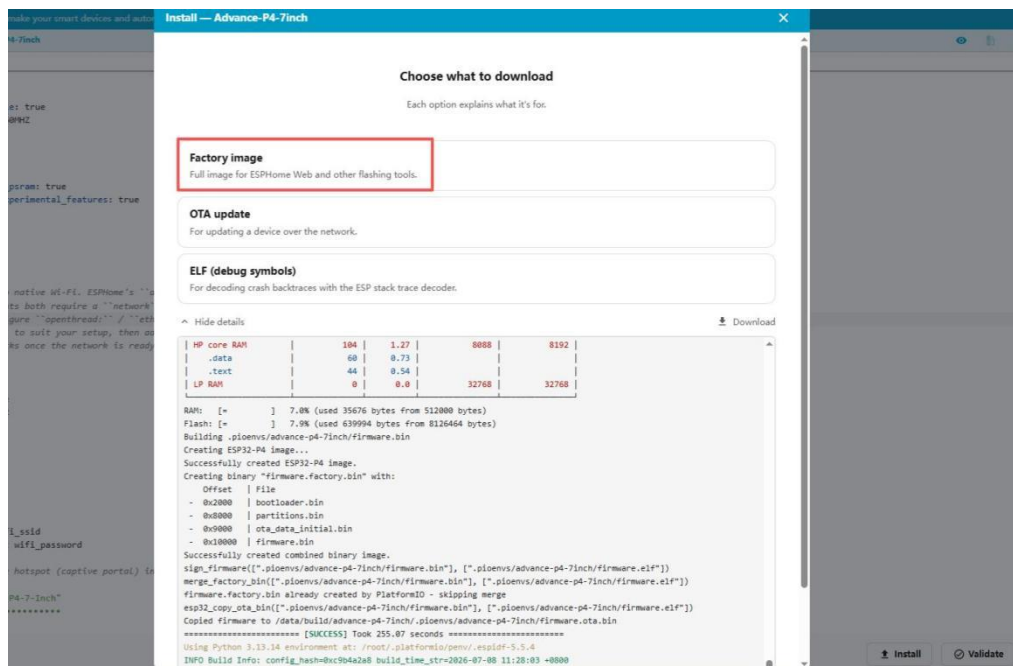
Subsequently, select the appropriate firmware burning method and carry out the first code upload.



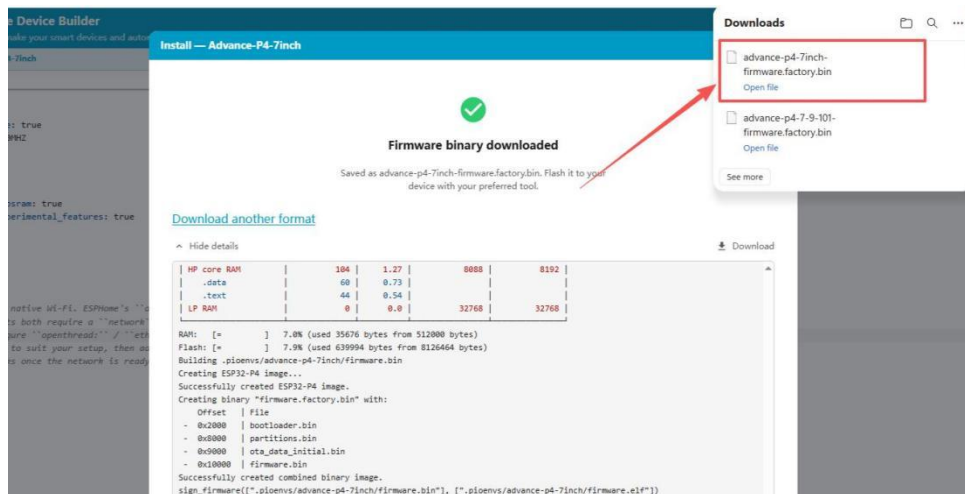
Wait for a few minutes until the installation is complete. The first installation and download process may take some time as it involves downloading necessary tools and libraries and compiling the code.



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



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



Remember the path of this .bin file.

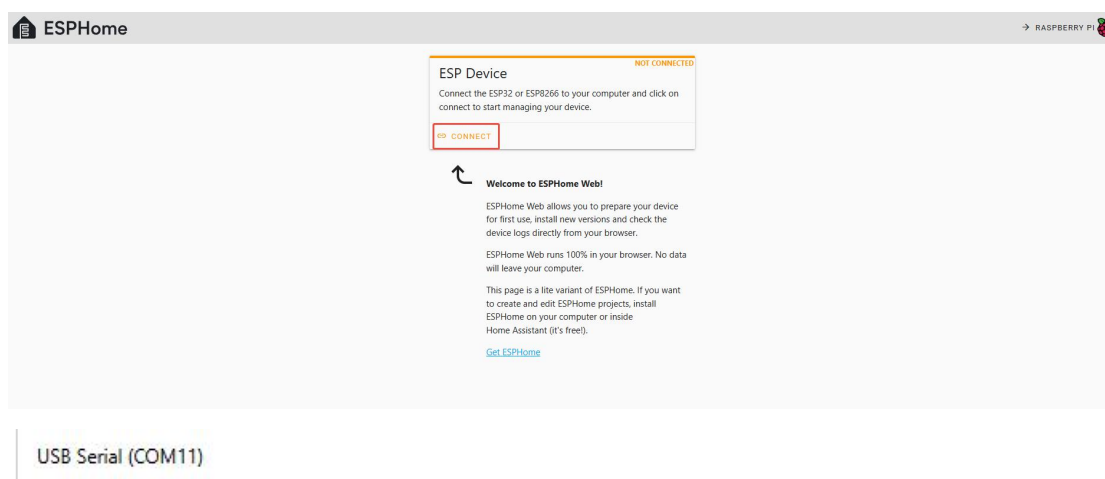
Open the following website:

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

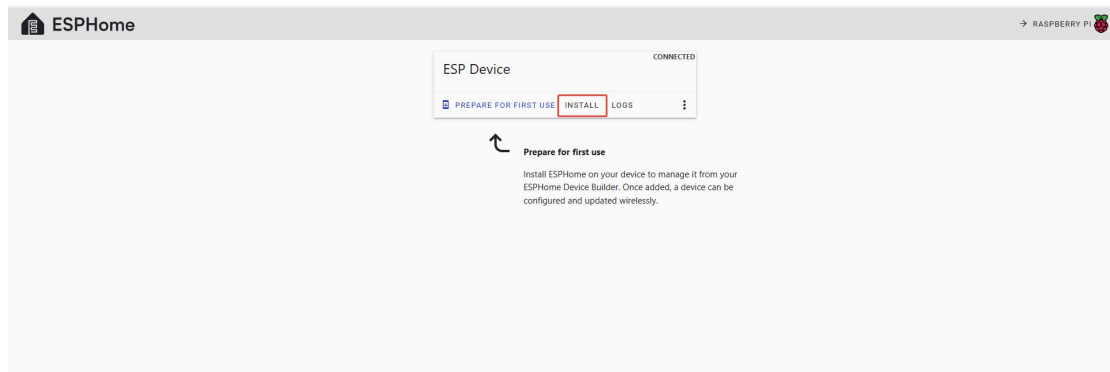
Next, we will flash this .bin file into the CrowPanel Advanced ESP32-P4 HMI AI Display (7inch / 9inch / 10.1inch) .

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

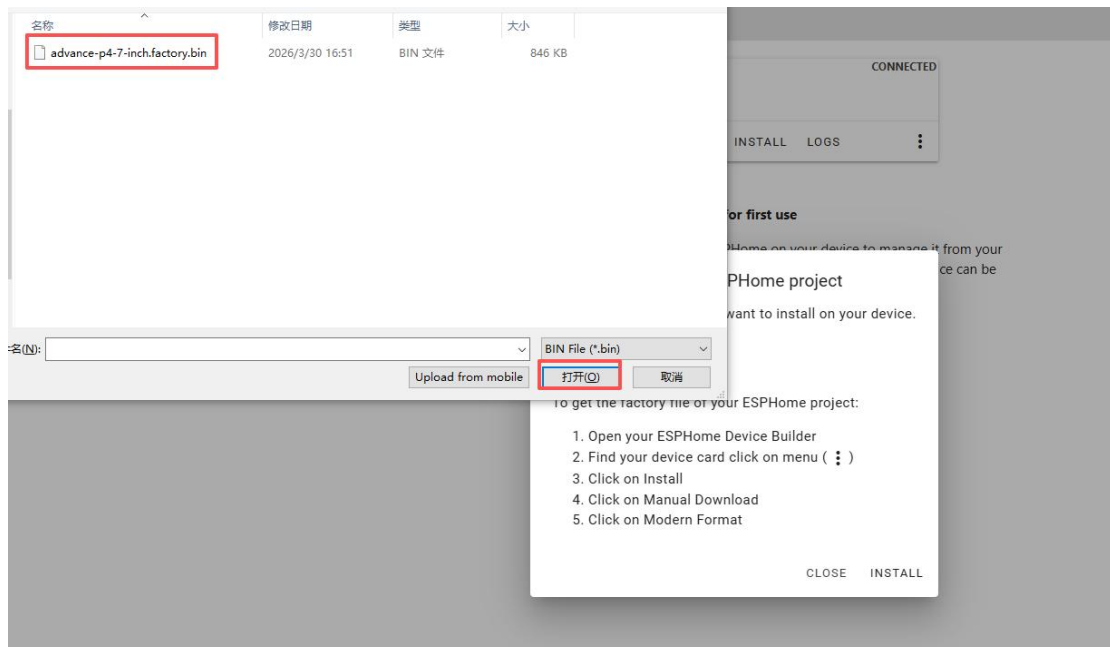
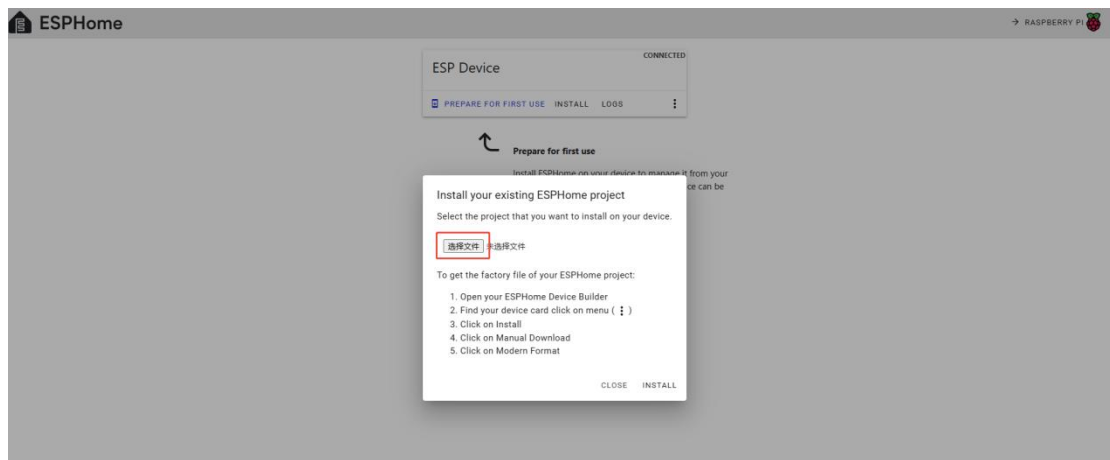
Click "Connect", select the COM port, and connect it.

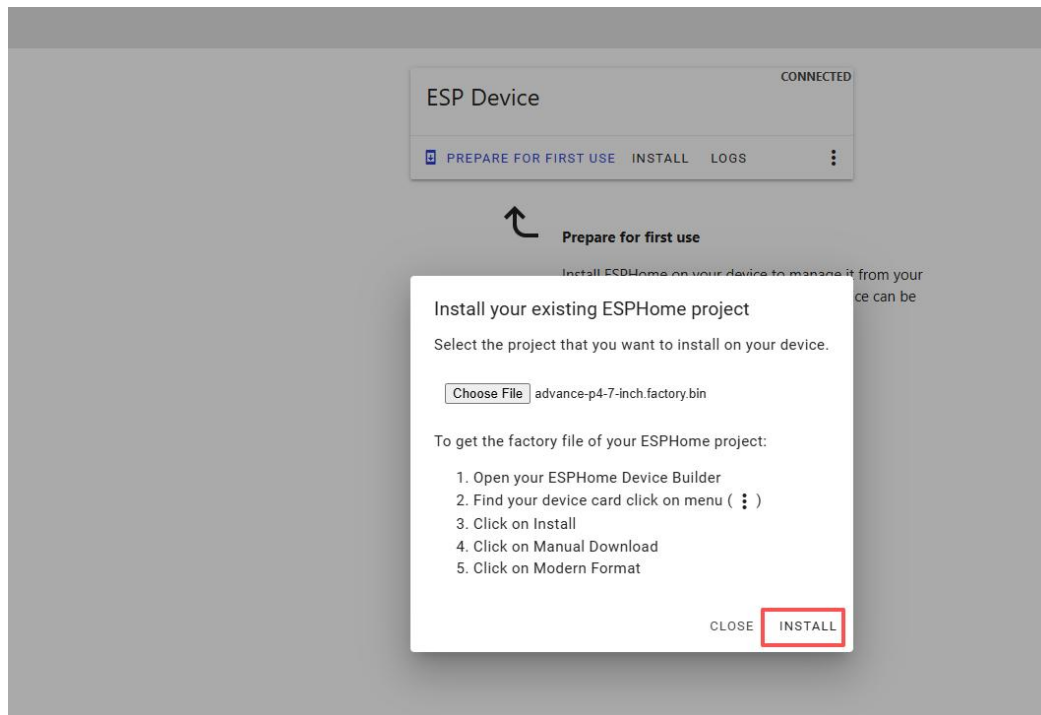


After connecting the CrowPanel Advanced ESP32-P4 HMI AI Display (7inch / 9inch / 10.1inch) , click "Install".

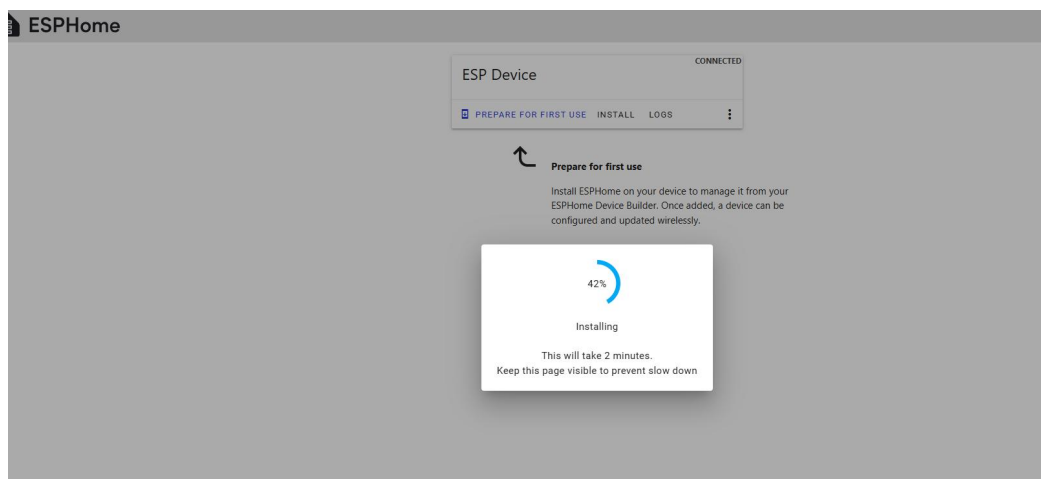


Add the .bin file you just downloaded, then click "Install".

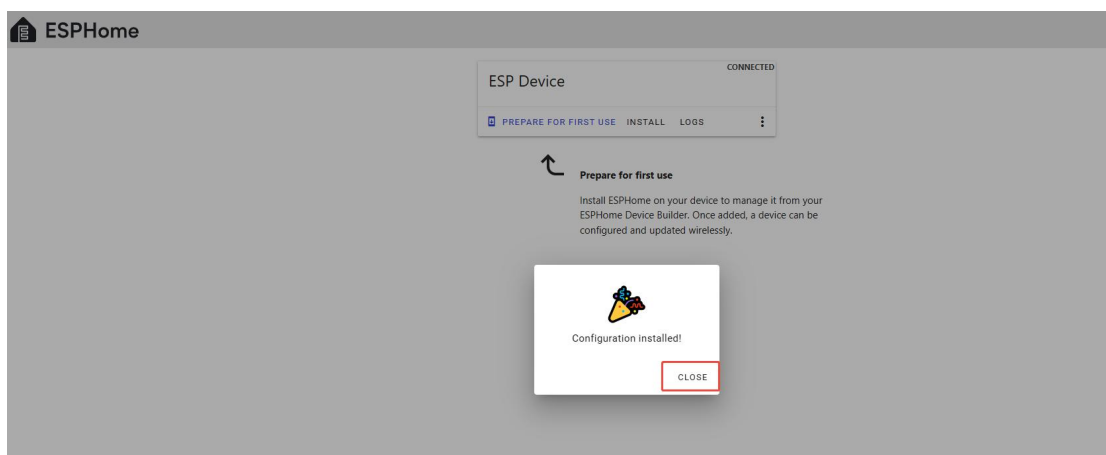




Wait for a few minutes.

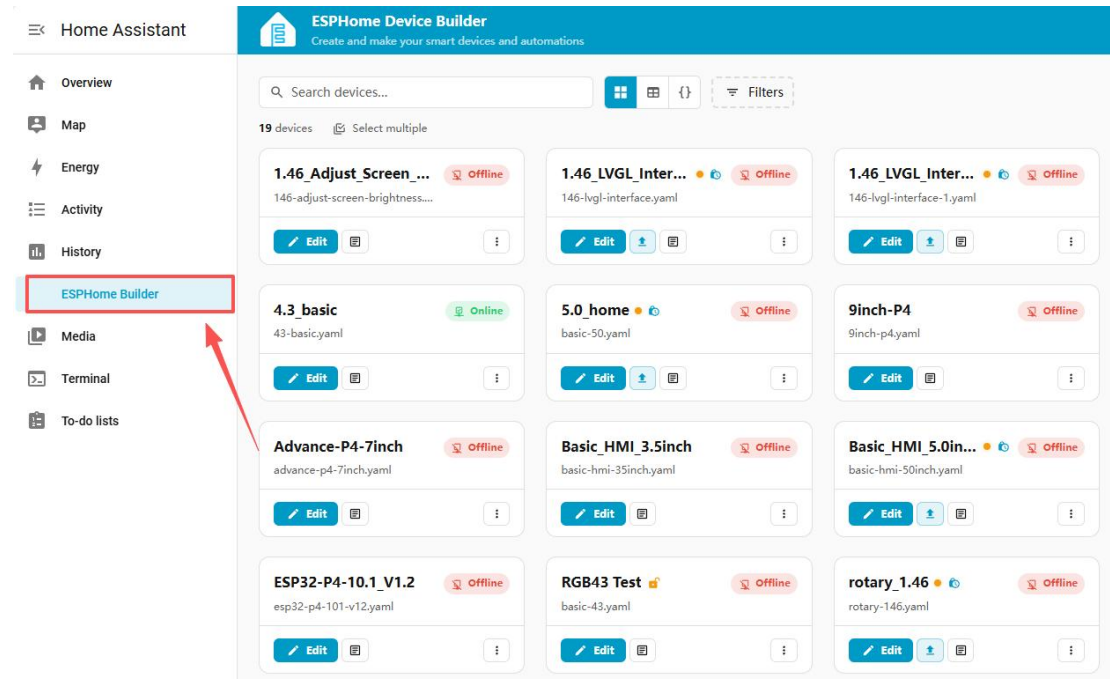


After the installation is complete, click "Close".

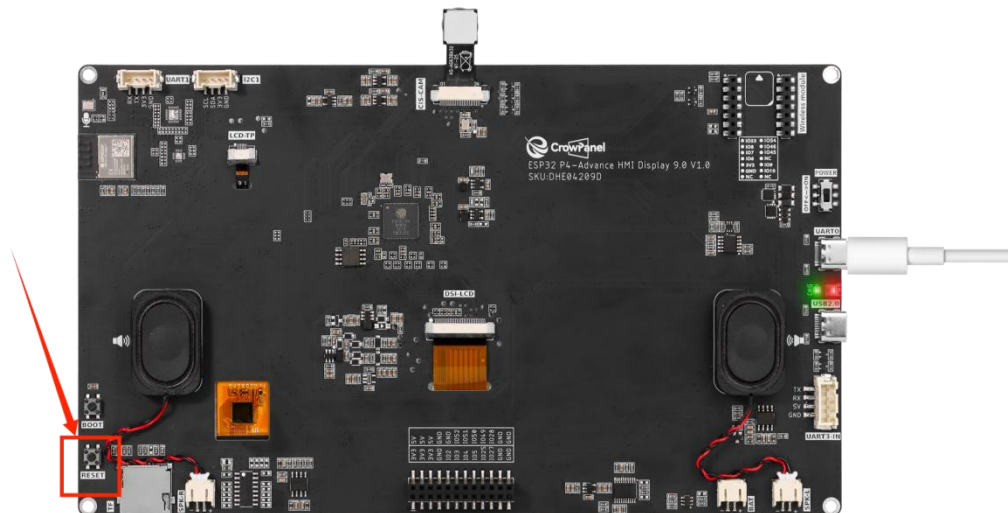


9. Observe the Wi-Fi connection status of the equipment

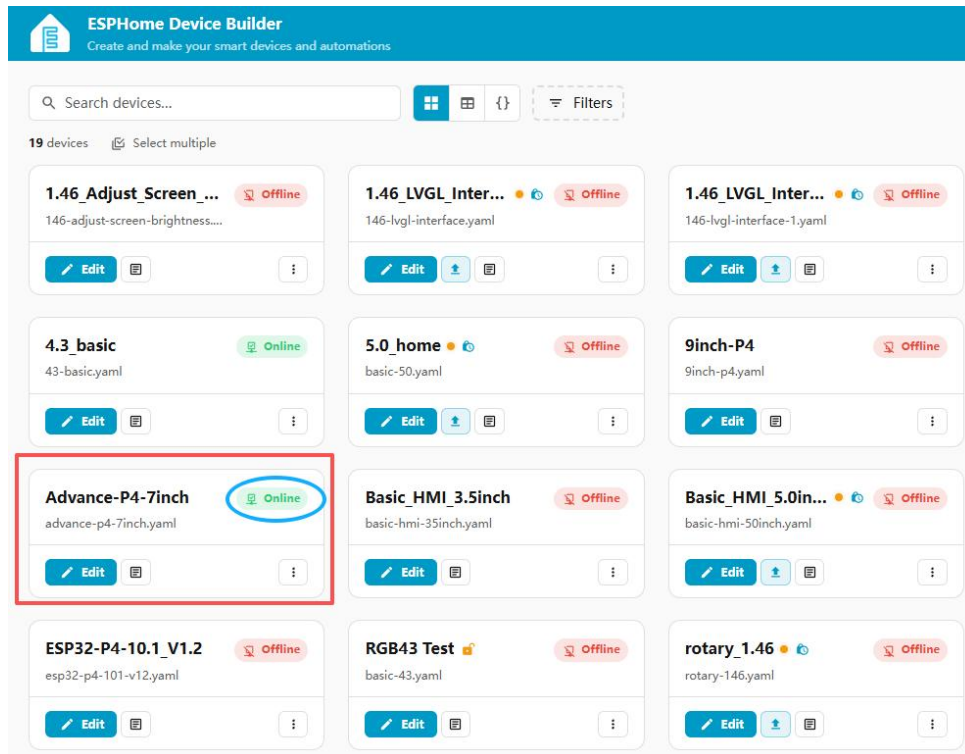
After successfully flashing the .bin file, return to the ESPHome page in Home Assistant.



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

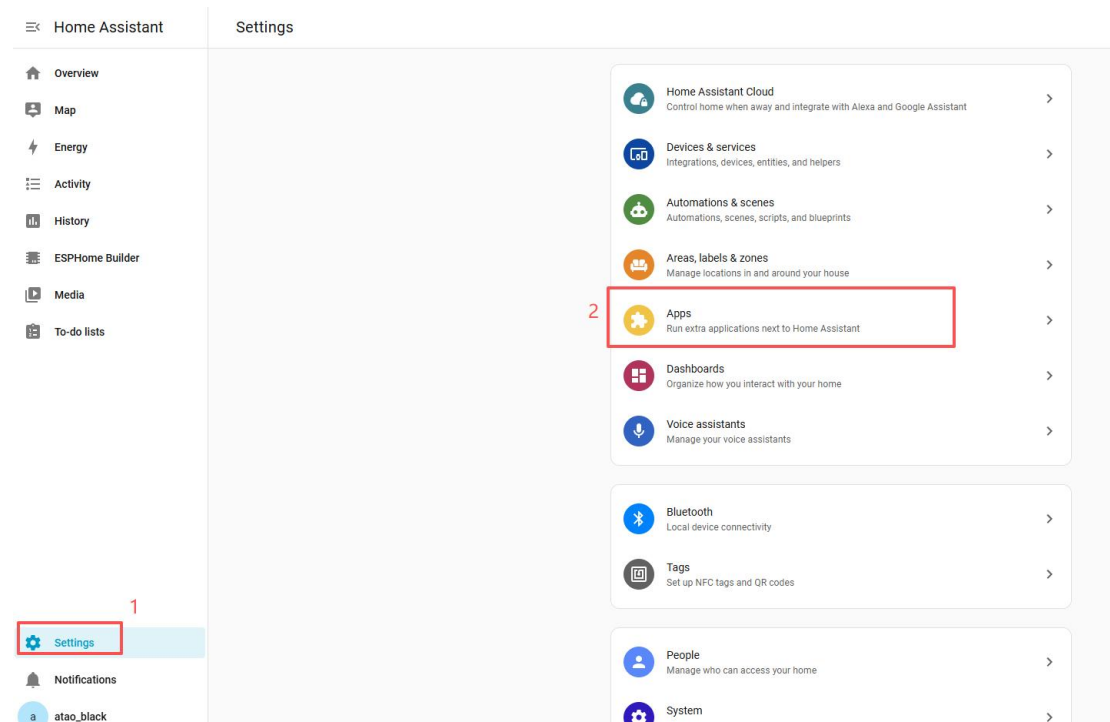
You can see the materials on the screen, which we need to use on the ESPHome platform, so they need to be uploaded to the ESPHome platform.

[Click here to download the materials shown on our screen.](#)

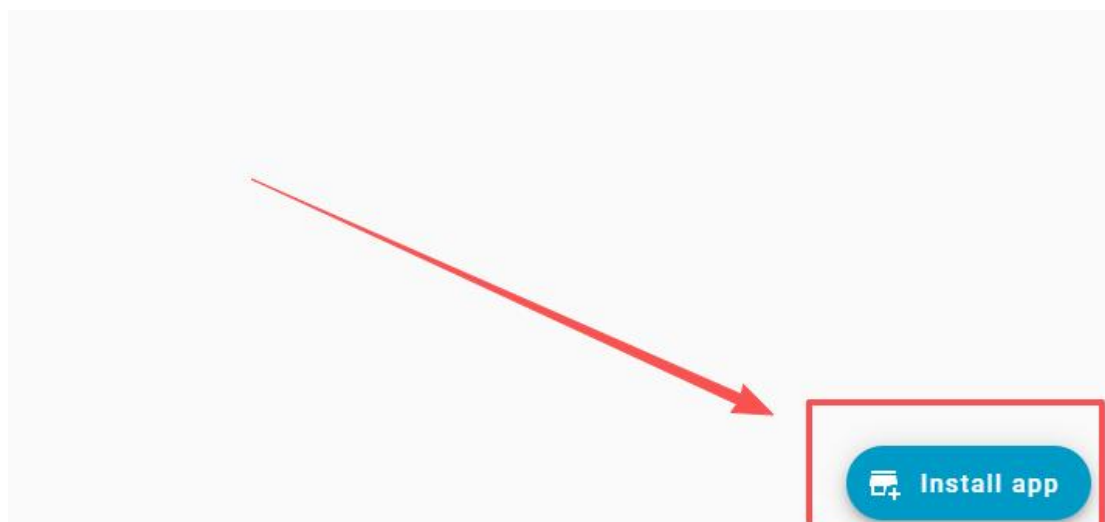
<https://github.com/Elecrow-RD/CrowPanel-Advanced-7inch-ESP32-P4-HMI-AI-Display-1024x600-IPS-Touch-Screen/tree/master/example/V1.2/ESPHome/Materials>

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

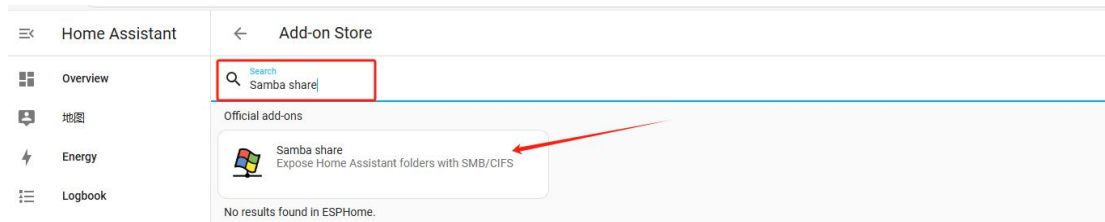
Click Settings and select Apps



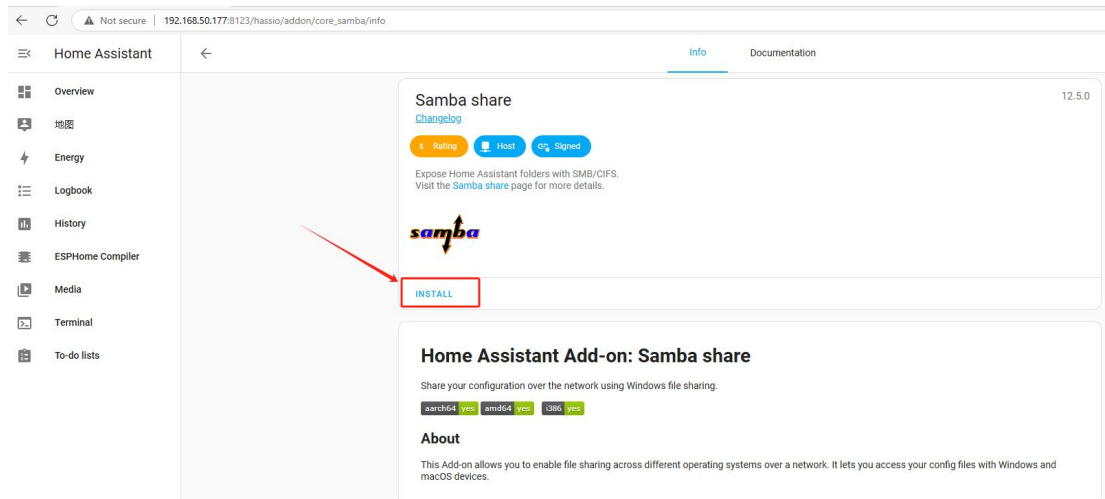
Then, click the “Install app” at the bottom right corner.



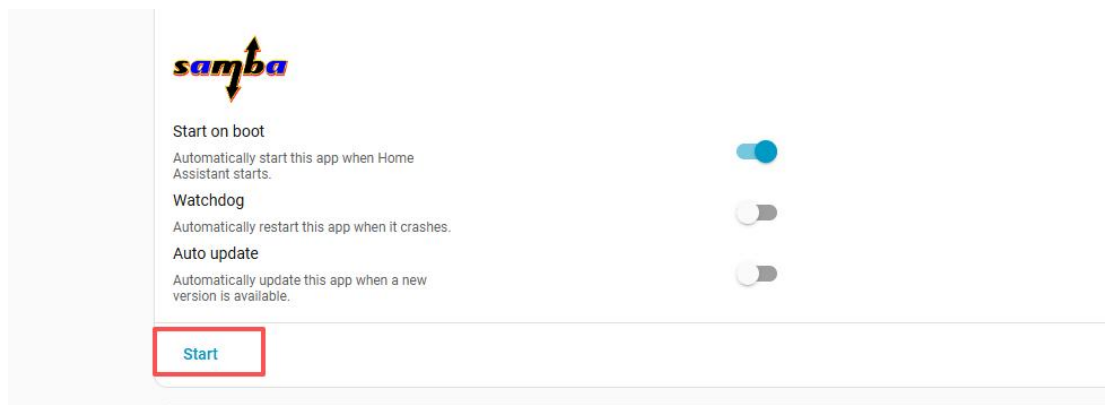
Search for Samba share at the top



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!

Configuration

Samba share(aliyun)

Options

Username*
atao_black

The username you would like to use to authenticate with the Samba server.

Password*

The password that goes with the username configured for authentication.

Workgroup*
WORKGROUP

Change WORKGROUP to reflect your network needs.

Enable Compatibility Mode
Enable this to use old legacy Samba protocols on the Samba add-on.

.* X .DS_Store X Thumbs.db X icon? X .Trashd X

Veto Files

List of files that are neither visible nor accessible.

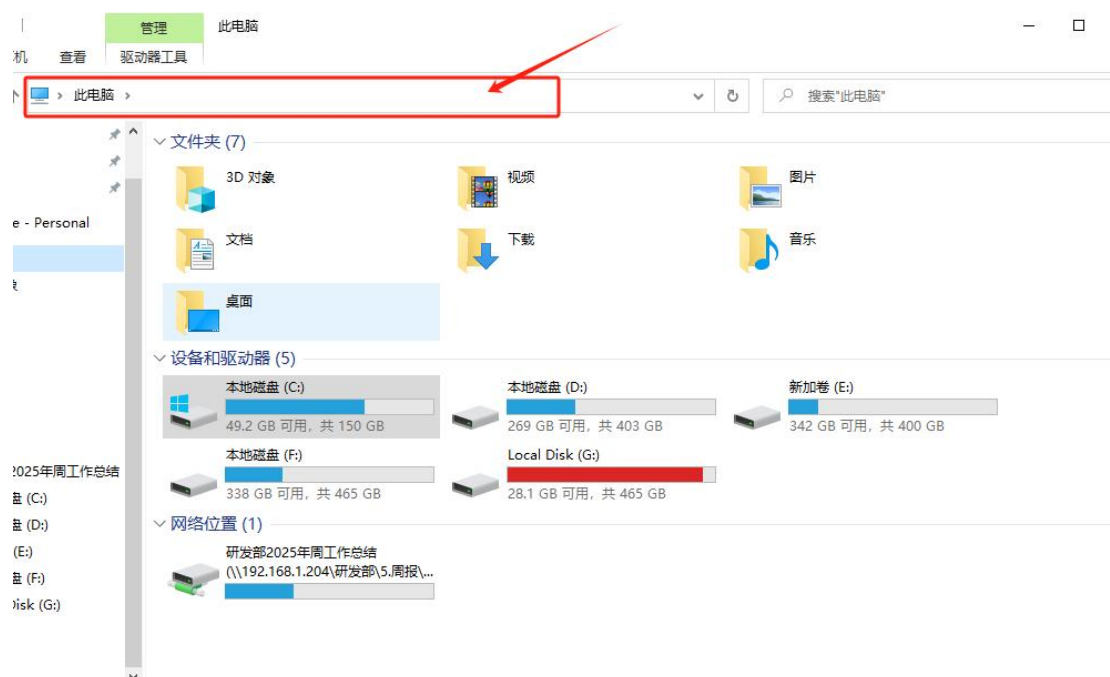
10.0.0.0/8 X 172.16.0.0/12 X 192.168.0.0/16 X 169.254.0.0/16 X fe80::/10 X fe00::/7 X

Allowed Hosts

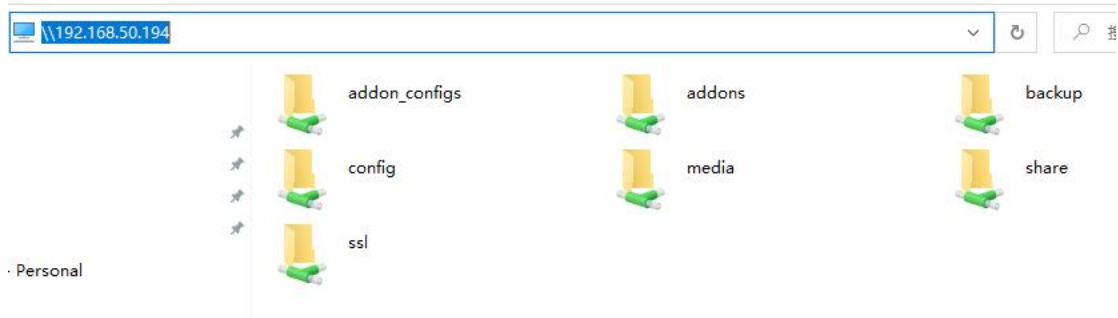
List of hosts/networks allowed to access the shared folders.

Save

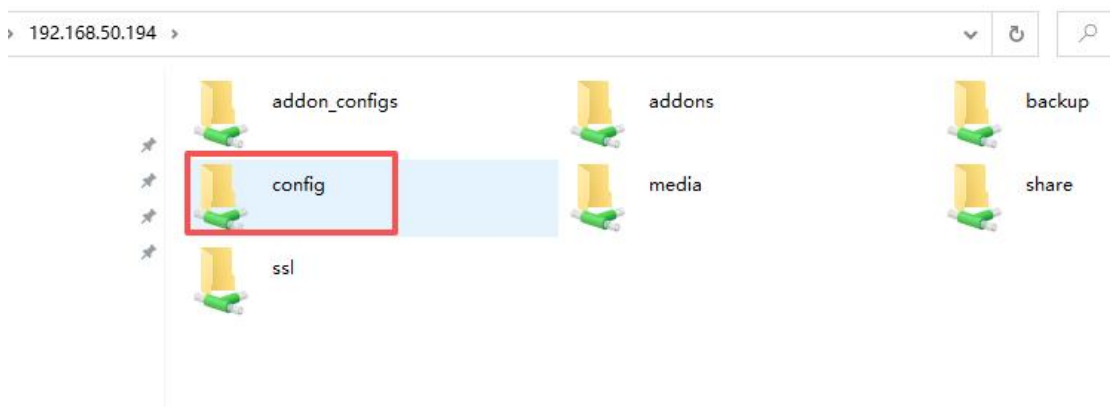
After saving, go to the File Explorer on your computer



Enter \\ + your Home Assistant IP address



Go to config



Then choose esphome

192.168.50.194
config

名称	修改日期	类型	大小
.cache	2026/3/30 15:05	文件夹	
.cloud	2026/3/30 15:02	文件夹	
.storage	2026/3/30 18:22	文件夹	
blueprints	2026/3/30 15:02	文件夹	
deps	2026/3/30 15:02	文件夹	
esphome	2026/3/30 18:09	文件夹	
tts	2026/3/30 15:02	文件夹	
.ha_run.lock	2026/3/30 18:07	LOCK 文件	1 KB
.HA_VERSION	2026/3/30 15:02	HA_VERSION 文件	1 KB
automations.yaml	2026/3/30 15:02	Yaml 源文件	1 KB
configuration.yaml	2026/3/30 15:02	Yaml 源文件	1 KB
home-assistant.log.fault	2026/3/30 15:02	FAULT 文件	0 KB
home-assistant_v2.db	2026/3/30 18:07	Data Base File	288 KB
home-assistant_v2.db-shm	2026/3/30 18:31	DB-SHM 文件	32 KB
home-assistant_v2.db-wal	2026/3/30 18:31	DB-WAL 文件	584 KB
scenes.yaml	2026/3/30 15:02	Yaml 源文件	0 KB
scripts.yaml	2026/3/30 15:02	Yaml 源文件	0 KB
secrets.yaml	2026/3/30 15:02	Yaml 源文件	1 KB

Put the materials you just downloaded into the esphome folder

192.168.50.194 > config > esphome

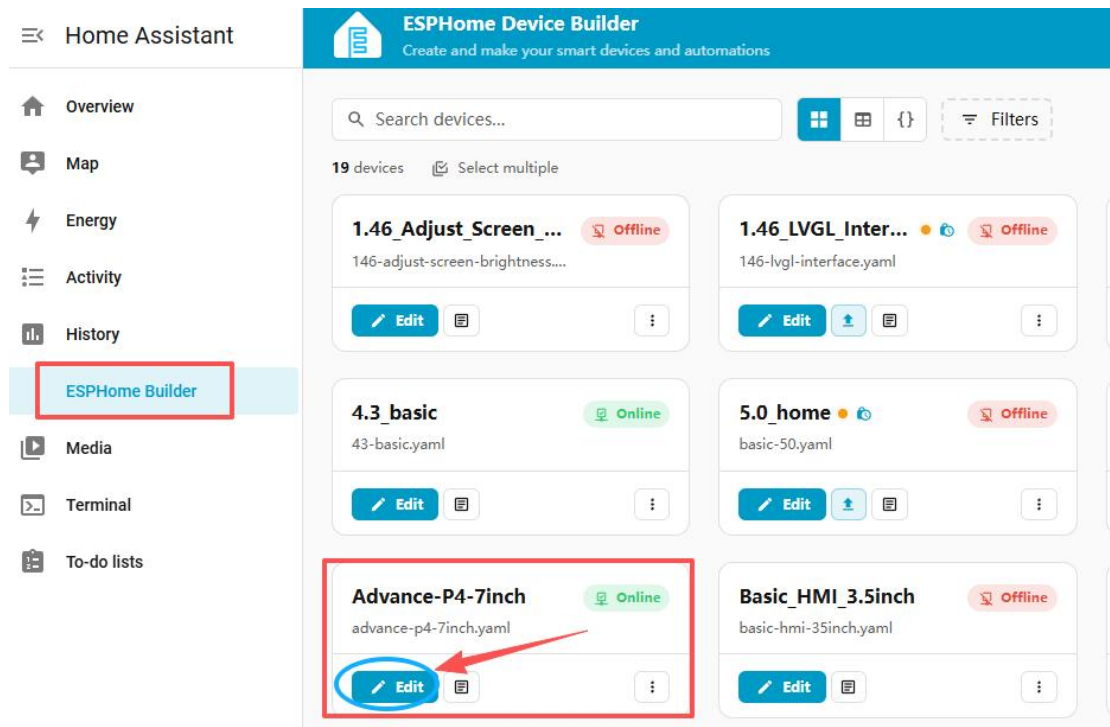
名称	修改日期	类型	大小
 .gitignore	2026/3/30 15:43	文本文档	1 KB
 advance-p4-7-inch.yaml	2026/3/30 18:09	Yaml 源文件	1 KB
 secrets.yaml	2026/3/30 18:09	Yaml 源文件	1 KB
 light.png	2025/6/13 15:15	PNG 图片文件	3 KB
 no_light.png	2026/3/30 17:06	PNG 图片文件	3 KB
 small_hum.png	2026/3/30 17:06	PNG 图片文件	4 KB
 small_logo.png	2026/3/30 17:06	PNG 图片文件	11 KB
 small_temp.png	2026/3/30 17:05	PNG 图片文件	4 KB

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

```
1  esphome:
2    name: advance-p4-7-inch
3    friendly_name: Advance-P4-7-inch
4    on_boot:
5      priority: 800
6      then:
7        - logger.log: "Backlight ON"
8        - output.turn_on: backlight_pwm
9        - delay: 200ms
10       - output.turn_off: power_light
11       - delay: 500ms
12
13     - light.turn_on:
14       id: back_light
15       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;

This on_boot startup script is based on the optimized correct execution logic according to the hardware schematic. After the device is powered on, it runs with a high priority of 800 and starts by marking the backlight power startup process through the log. Immediately, it turns on the power_light backlight main power corresponding to GPIO29 to ensure the normal operation of the backlight power supply circuit. After waiting for 200ms for the voltage to stabilize with the hardware, it uses light.turn_on to set the backlight brightness of GPIO31-controlled back_light to 100% to achieve a safe, orderly and stable lighting of the screen backlight.

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".

```

17  esp32:
18    · variant: esp32p4
19    · engineering_sample: true
20    · cpu_frequency: 360MHZ
21    · flash_size: 16MB
22    · framework:
23      · type: esp-idf
24      · advanced:
25        · execute_from_psram: true
26        · 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".

```

28  psram:
29  |   speed: 200MHz
30
31  esp_0do:
32  |   - channel: 3
33  |     voltage: 2.5V
34  |   - channel: 4
35  |     voltage: 2.5V
36
37  # Enable logging
38  logger:
39  |   level: debug
40  |   logs:
41  |     lvgl: info
42  |   hardware_uart: UART0
43
44  # Enable Home Assistant API
45  api:
46
47  |   encryption:
48  |     key: "Dp4Wgl1jj5F++6mnzSma7lHvgHCGfghlE/ohT96S+jw="
49
50  ota:
51  |   - platform: esphome
52  |     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_0do. 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:

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

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".

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.

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

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: .....

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 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.

```
76  i2c:
77    - id: bus_a
78      sda: GPIO45
79      scl: GPIO46
80      frequency: 400kHz
81
82  sensor:
83    - platform: aht10
84      variant: AHT20
85      temperature:
86        name: "Room Temperature"
87        id: room_temp
88        unit_of_measurement: "°C"
89        accuracy_decimals: 1
90      humidity:
91        name: "Room Humidity"
92        id: room_hum
93        unit_of_measurement: "%"
94        accuracy_decimals: 1
95      update_interval: 1s
96
```

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.

```
97  touchscreen:
98      - platform: gt911
99          id: my_touchscreen
100         i2c_id: bus_a
101         reset_pin: 40
102         interrupt_pin: 42
103         update_interval: 50ms
104         transform:
105             swap_xy: false
106             mirror_x: false
107             mirror_y: false
108         # Enabled Touch Logger here
109         on_touch:
110             - logger.log:
111                 format: "Touch at (%d, %d)"
112                 args: [touch.x, touch.y]
113             - lambda: |-
114                 ESP_LOGI("cal", "x=%d, y=%d, x_raw=%d, y_raw=%d",
115                     touch.x,
116                     touch.y,
117                     touch.x_raw,
118                     touch.y_raw
119                 );
120
```

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.

```

121  switch:
122    - platform: gpio
123      pin: GPIO48
124      name: "LED"
125      id: led_output
126      on_turn_on:
127        # Switch bulb diagram
128        - lvgl.image.update:
129          id: lvgl_light
130          src: light_on
131        # Switch the status label text to ON
132        - lvgl.label.update:
133          id: label_light_state
134          text: "ON"
135      on_turn_off:
136        # Switch bulb diagram
137        - lvgl.image.update:
138          id: lvgl_light
139          src: light_off
140        # Switch the status label text to OFF
141        - lvgl.label.update:
142          id: label_light_state
143          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

```

145 output:
146   - platform: ledc
147     pin: GPIO31
148     id: backlight_pwm
149   - platform: gpio
150     id: power_light
151     pin: GPIO29
152     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.

```

162 display:
163   - platform: mipi_dsi
164     id: my_display
165     model: WAVESHARE-ESP32-P4-WIFI6-TOUCH-LCD-7B
166     reset_pin:
167       number: 41
168     update_interval: never
169     auto_clear_enabled: false
170     dimensions:
171       width: 1024
172       height: 600
173     color_order: RGB
174     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.

```
176 image:
177   - file: "small_temp.png"
178     id: small_temp
179     resize: 200x200
180     type: RGB565
181     byte_order: little_endian
182   - file: "small_hum.png"
183     id: small_hum
184     resize: 200x200
185     type: RGB565
186     byte_order: little_endian
187   - file: "small_logo.png"
188     id: small_logo
189     resize: 200x100
190     type: RGB565
191     byte_order: little_endian
192   - file: "light.png"
193     id: light_on
194     resize: 200x200
195     type: RGB565
196     transparency: alpha_channel
197     byte_order: little_endian
198   - file: "no_light.png"
199     id: light_off
200     resize: 200x200
201     type: RGB565
202     transparency: alpha_channel
203     byte_order: little_endian
204
```

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.

```
205 lvgl:
206     buffer_size: 50%
207     byte_order: little_endian
208     color_depth: 16
209     log_level: INFO
210     bg_color: 0xFFFFFF
211     text_font: montserrat_26
212     widgets:
213         - image:
214             id: lvgl_logo
215             src: small_logo
216             x: 400
217             y: 20
218         - image:
219             id: lvgl_temp
220             src: small_temp
221             x: 250
222             y: 250
223         - image:
224             id: lvgl_hum
225             src: small_hum
226             x: 500
227             y: 250
228         - image:
229             id: lvgl_light
230             src: light_off # Default 'Turn off lights' image
231             x: 750 # You can adjust the position by yourself
232             y: 250
233         - label:
234             id: label_temp
235             x: 190
236             y: 350
237             text: "Temp: --°C"
238         - label:
239             id: label_hum
240             x: 450
241             y: 350
242             text: "Humi: --%"
243         - label:
244             id: label_light_state
245             x: 750
246             y: 350
247             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: --°C", "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.

```
249 interval:
250   - interval: 1s
251     then:
252       - lambda: |-
253           char temp_str[20];
254           char hum_str[20];
255           snprintf(temp_str, sizeof(temp_str), "Temp: %.1f°C", id(room_temp).state);
256           snprintf(hum_str, sizeof(hum_str), "Humi: %.1f%", id(room_hum).state);
257           lv_label_set_text(id(label_temp), temp_str);
258           lv_label_set_text(id(label_hum), hum_str);
259
```

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

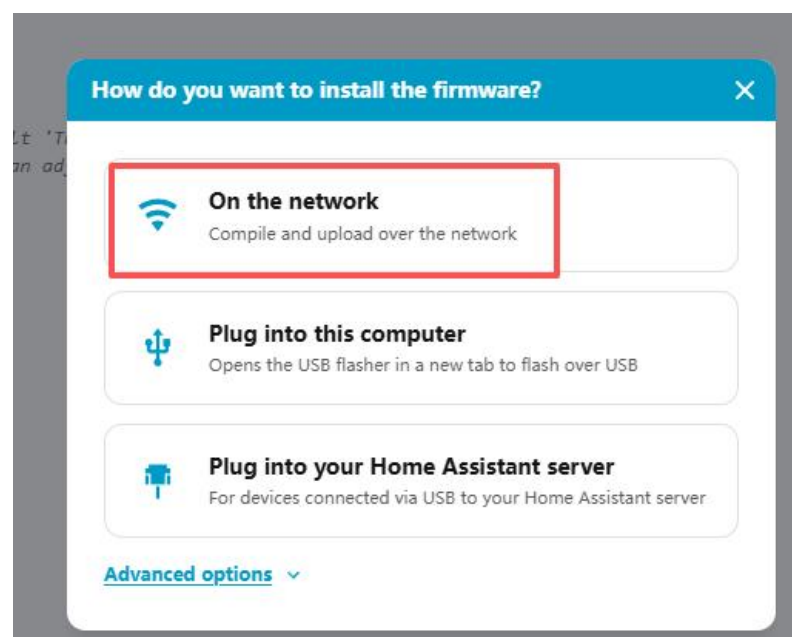
Now, all the codes have been prepared. Next, we will upload the codes.



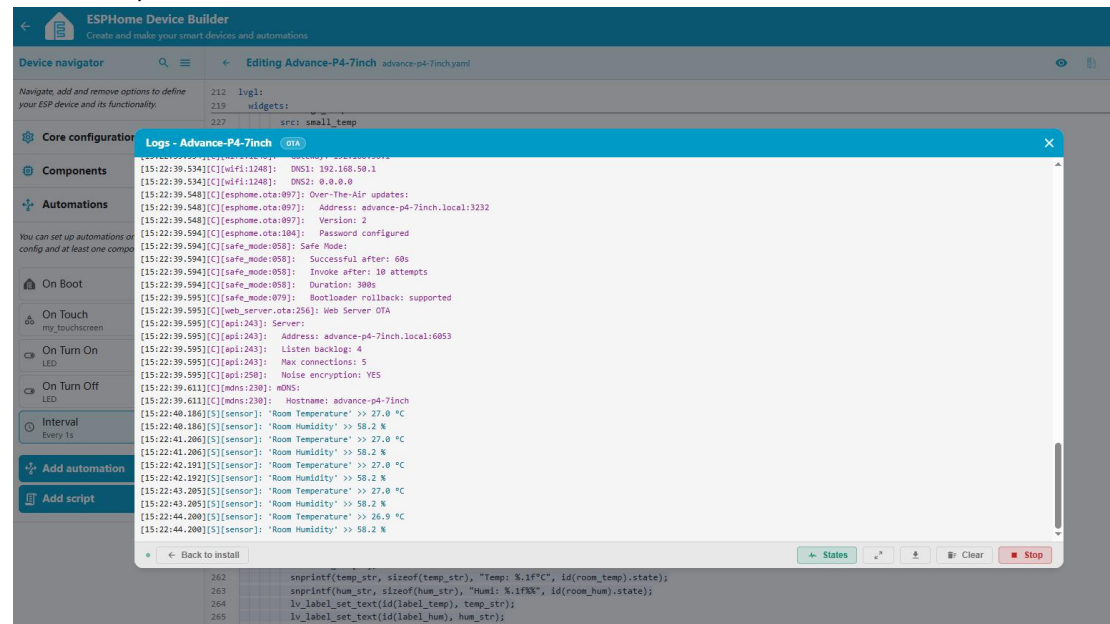
Then select "On the network"

From now on, as long as the network and OTA are functioning properly, just upload the code in this way.

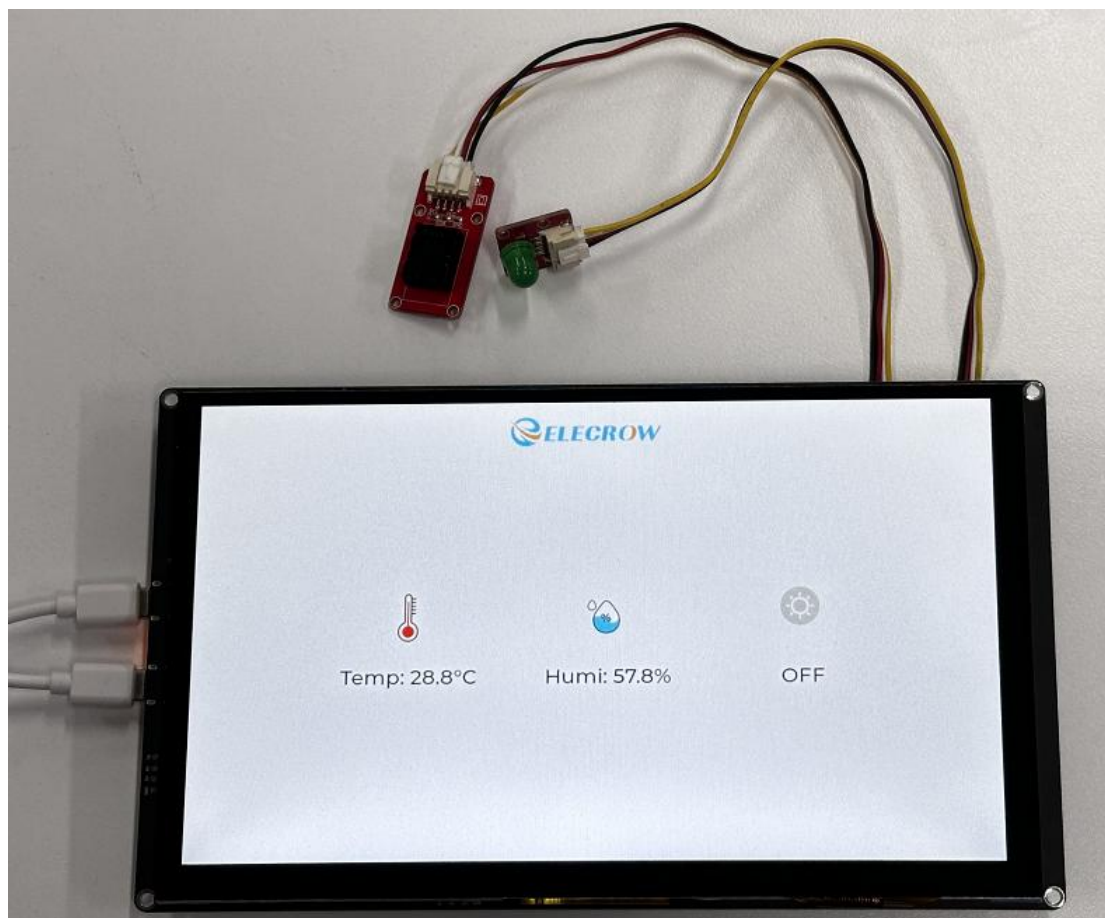
(If this method doesn't work, then use the installation method described in "[8. First upload of code](#)".)



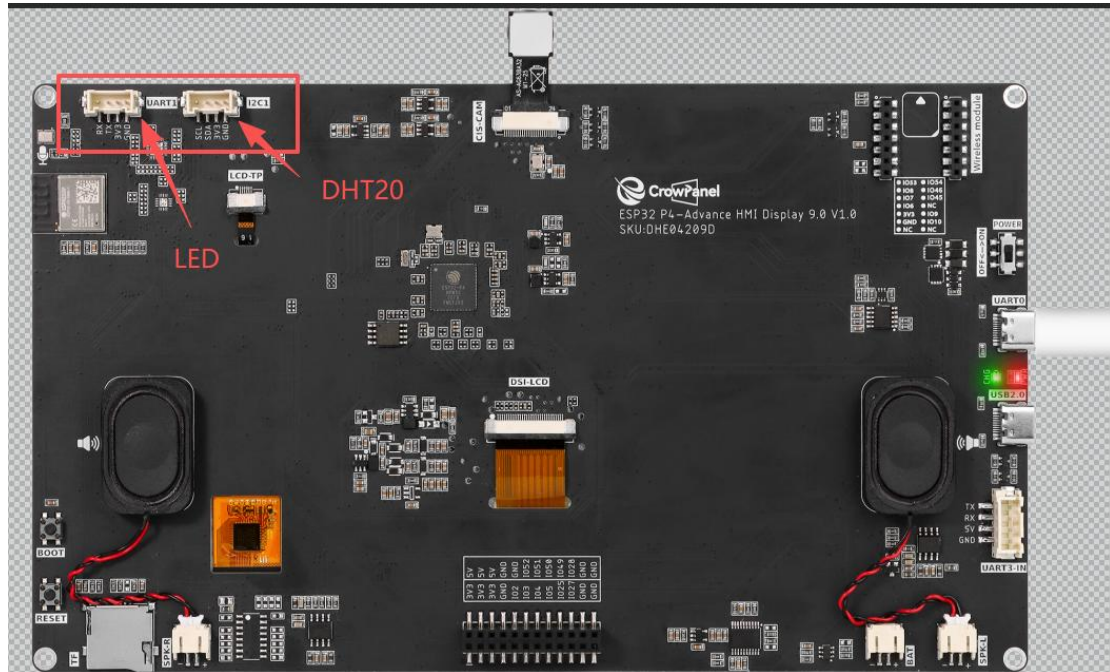
After the code is uploaded, the temperature and humidity values connected to the product will start to be printed.



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.

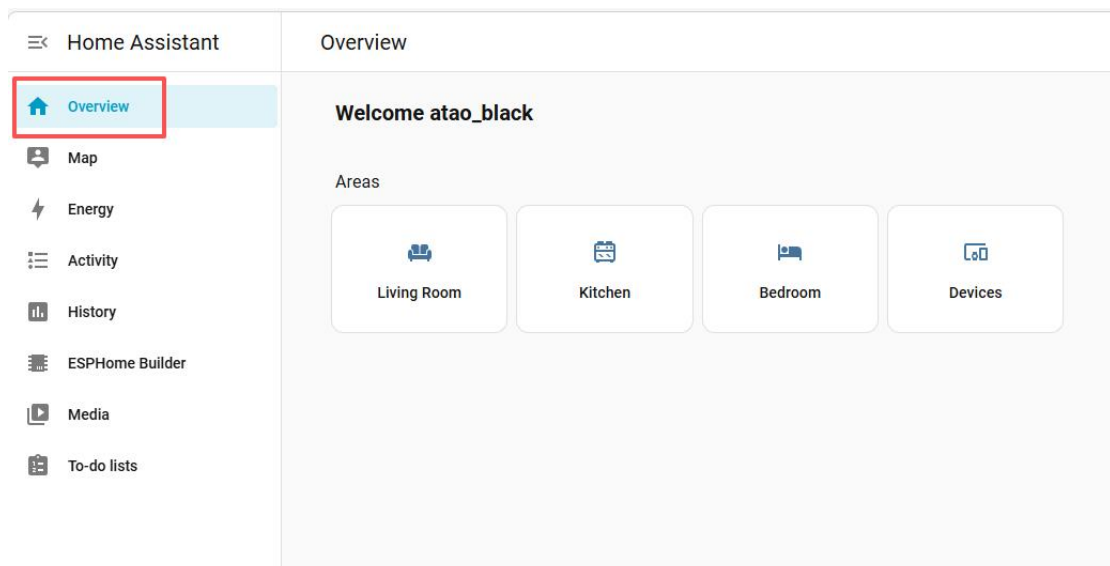


Then you will be able to see the real-time display of temperature and humidity data on the screen.

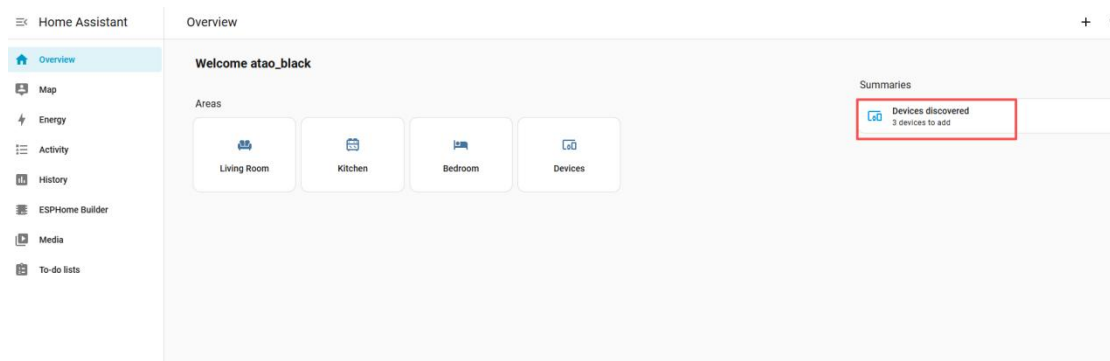
13. Remote observation and control

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.

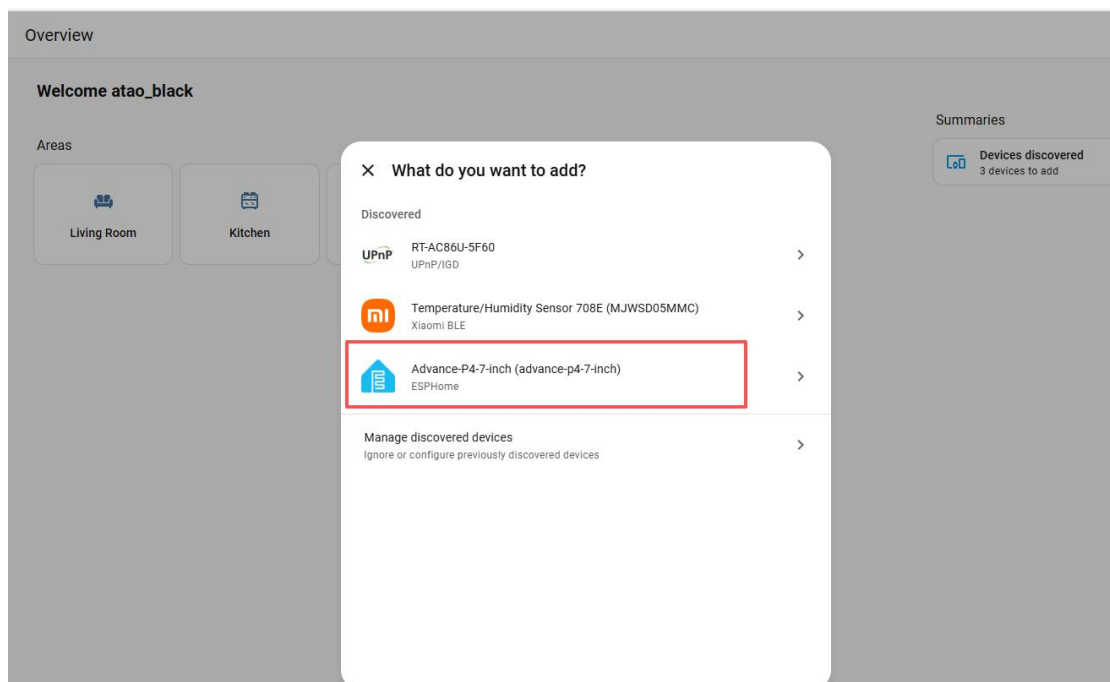
Let's return to the initial interface.



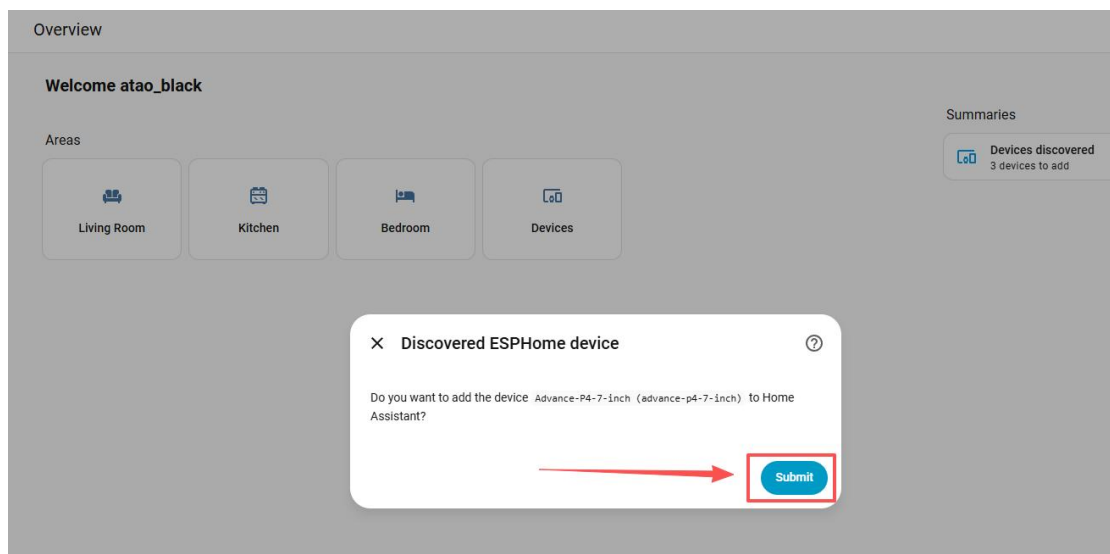
Once your code is uploaded successfully, you will be able to see your device information here.



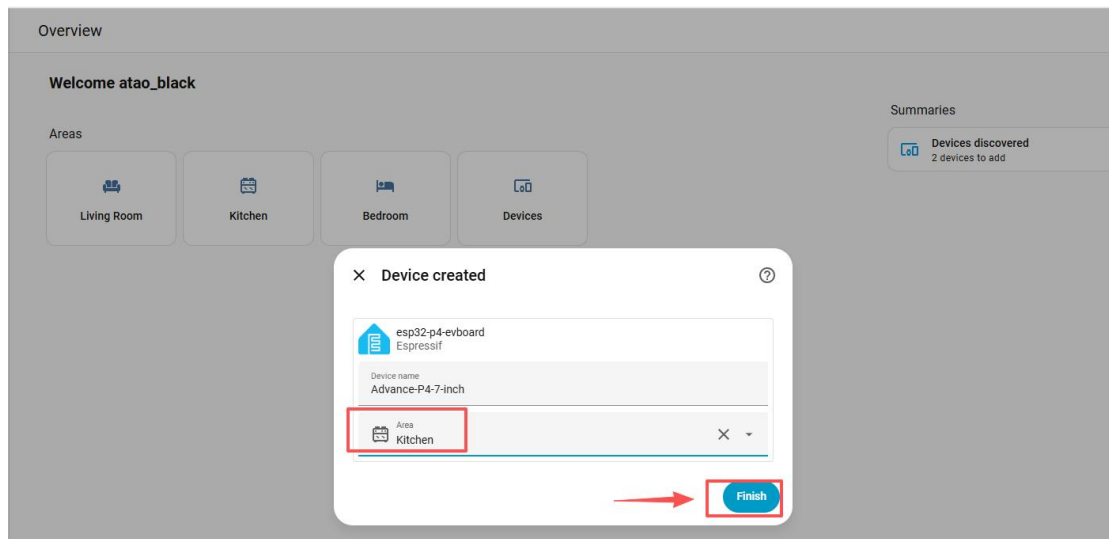
This is the system's scan of the devices nearby you.



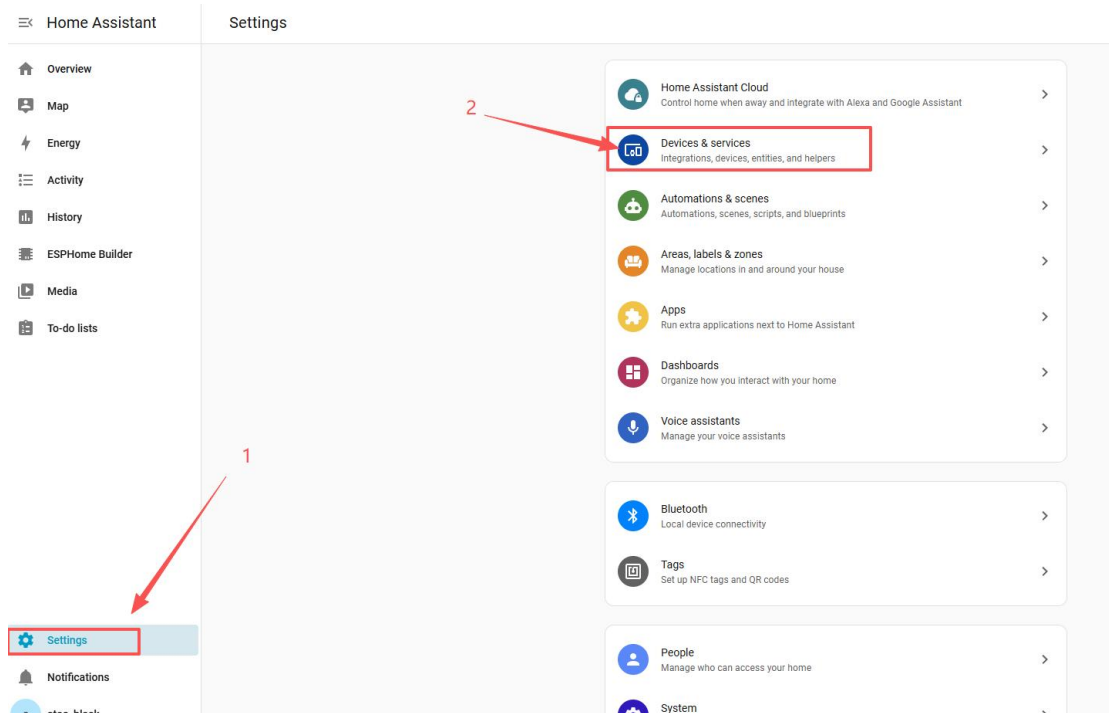
Click on our device and confirm the addition.



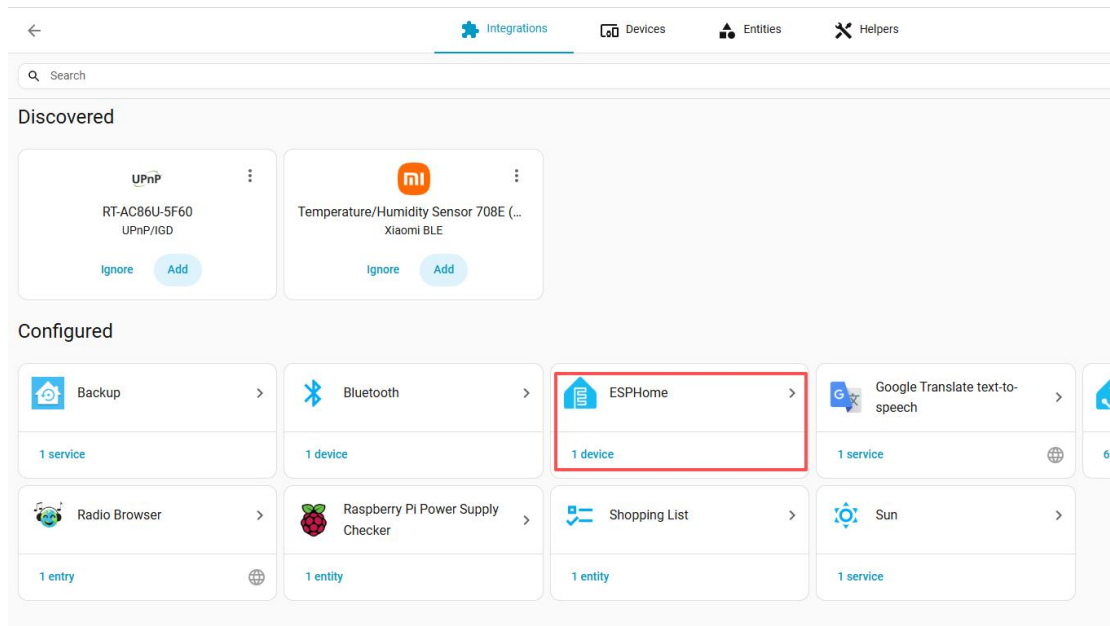
You can also create a zone for it. Here, I choose to place it in "kitchen".



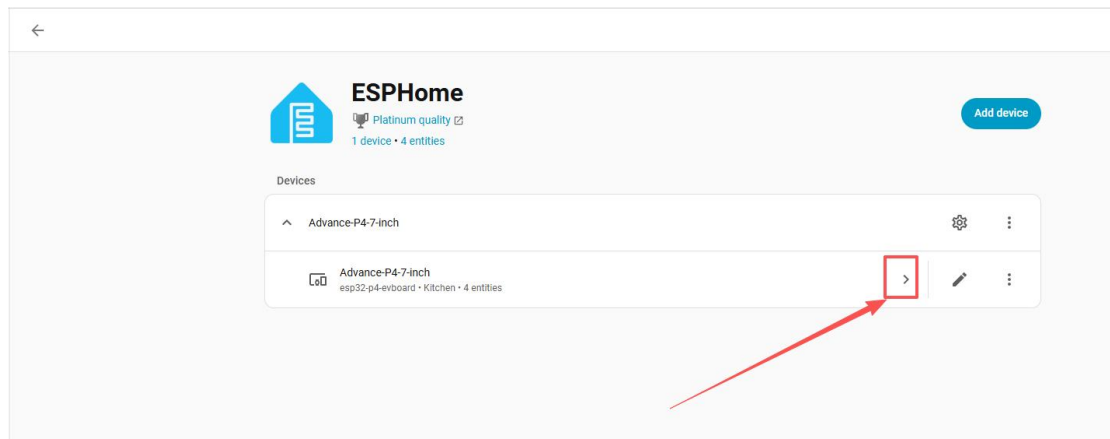
After completion, go to settings and select devices

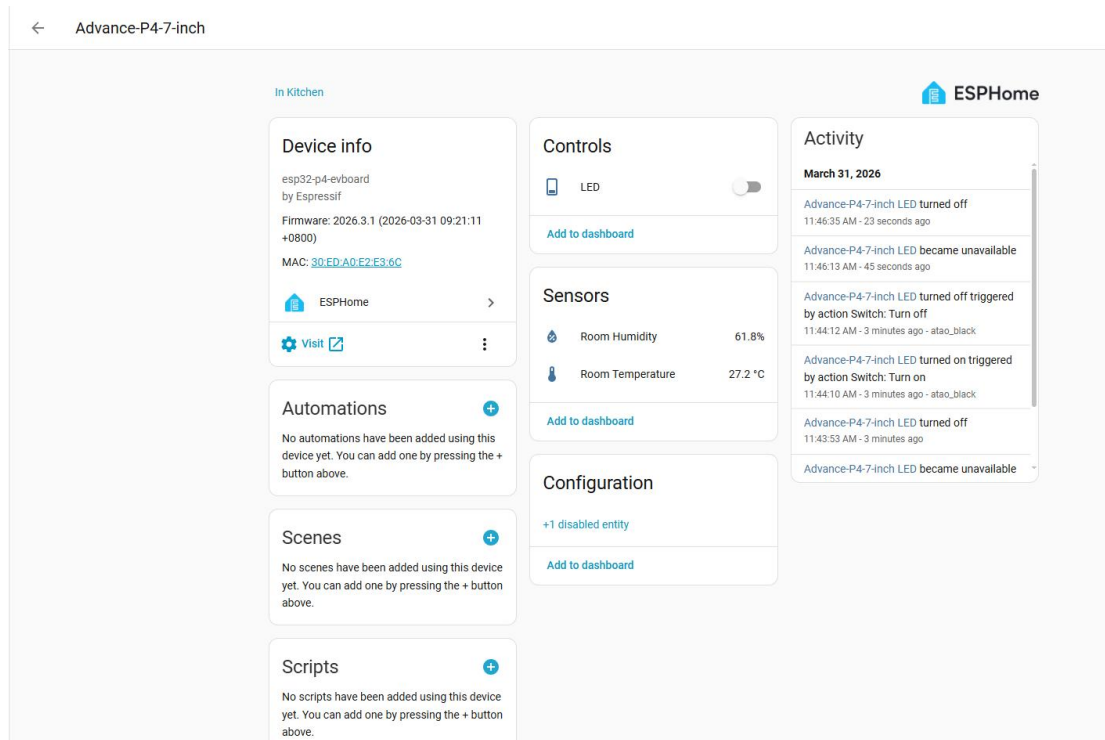


Choose ESPHome

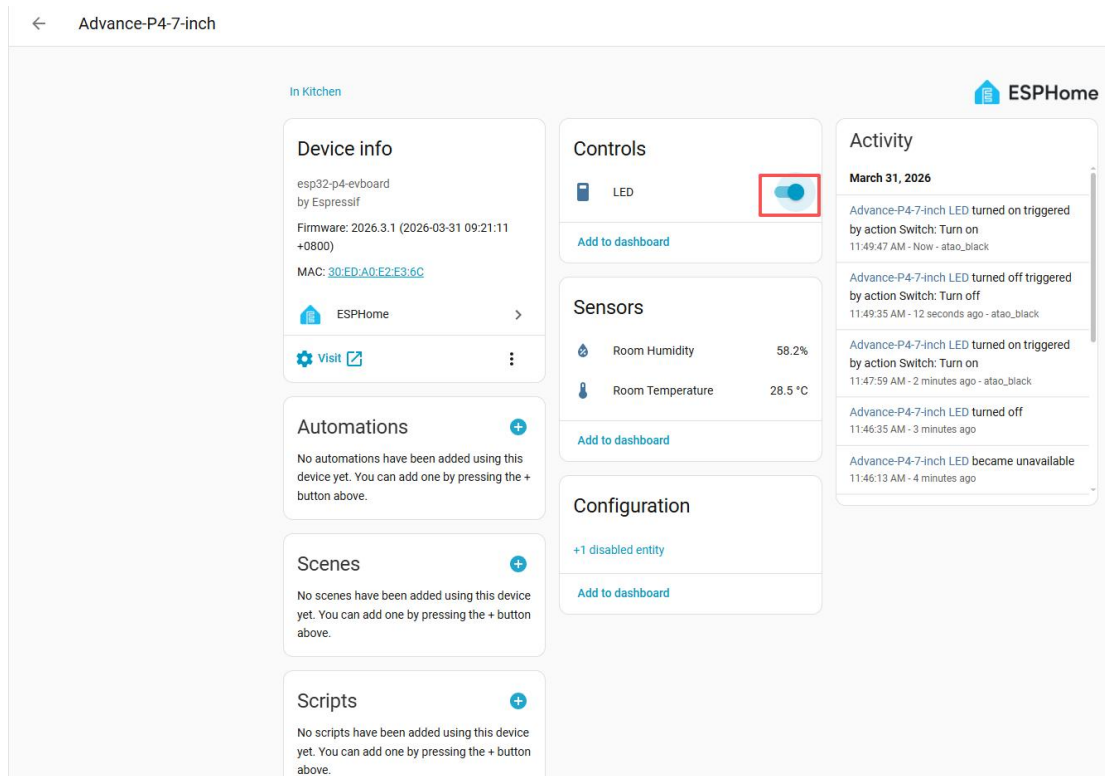


Here you can see the specific details of the equipment we just added.





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.

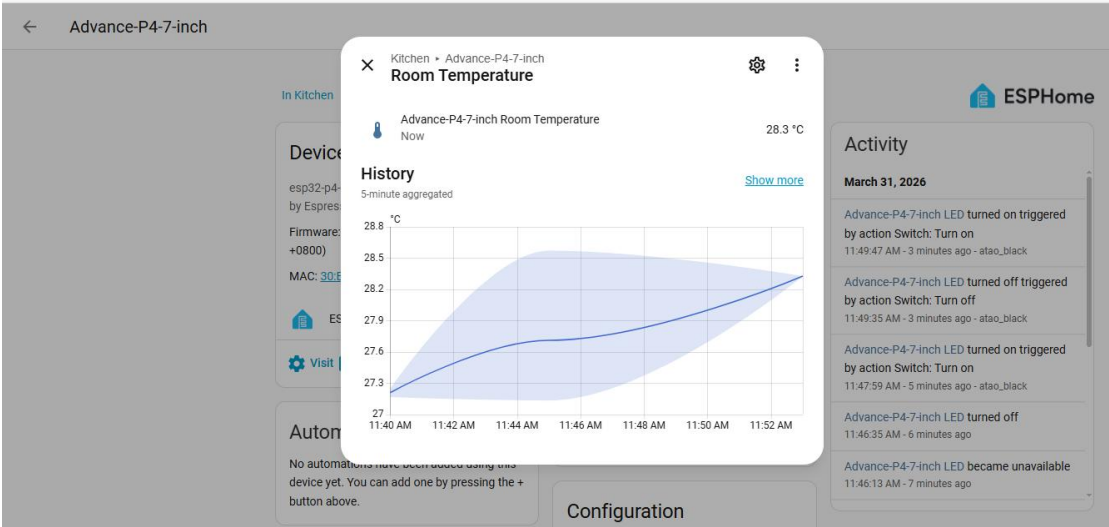
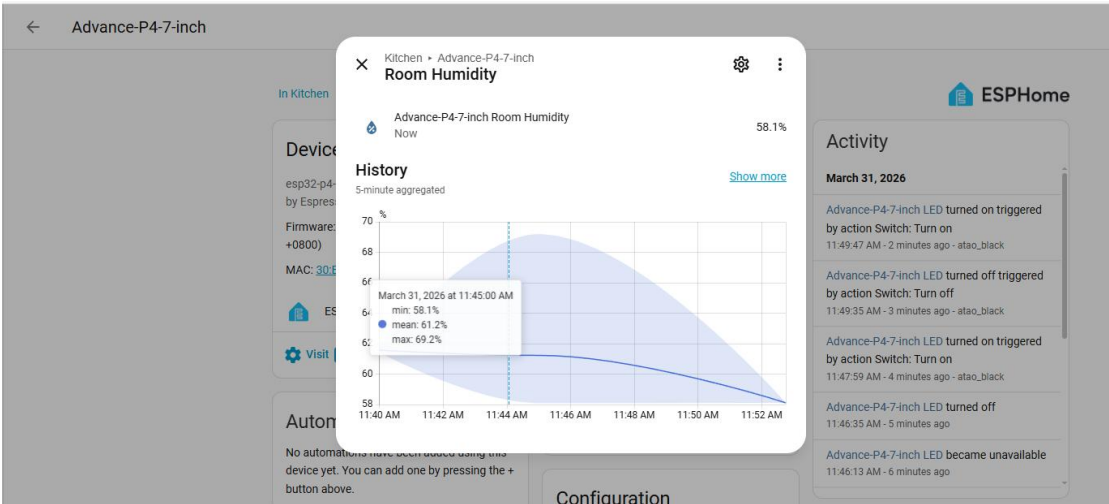
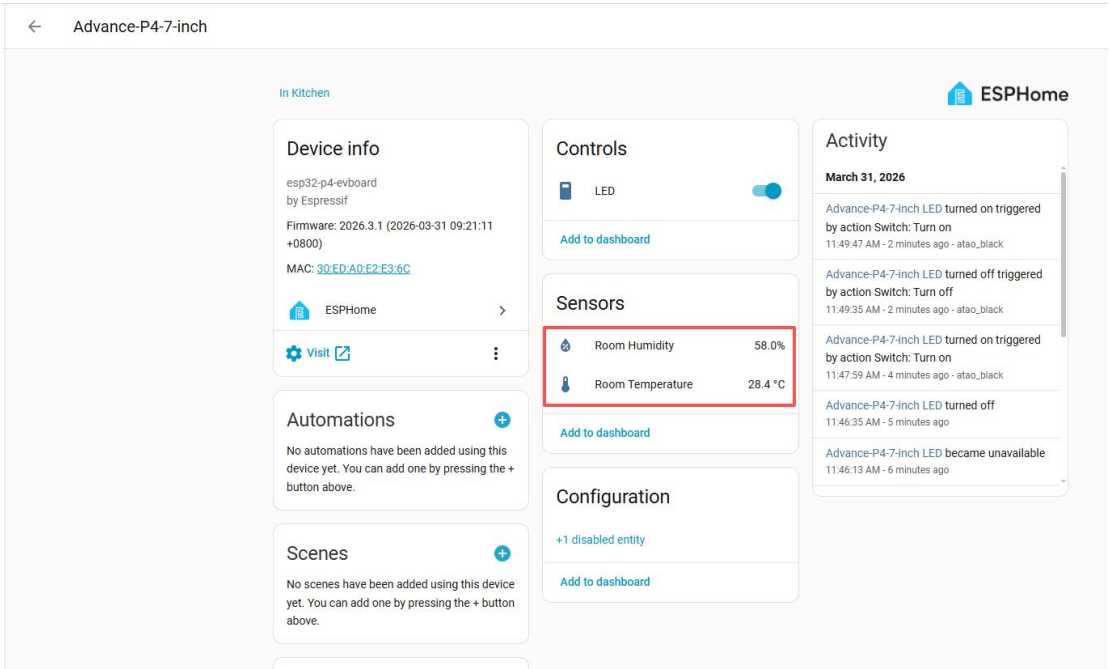




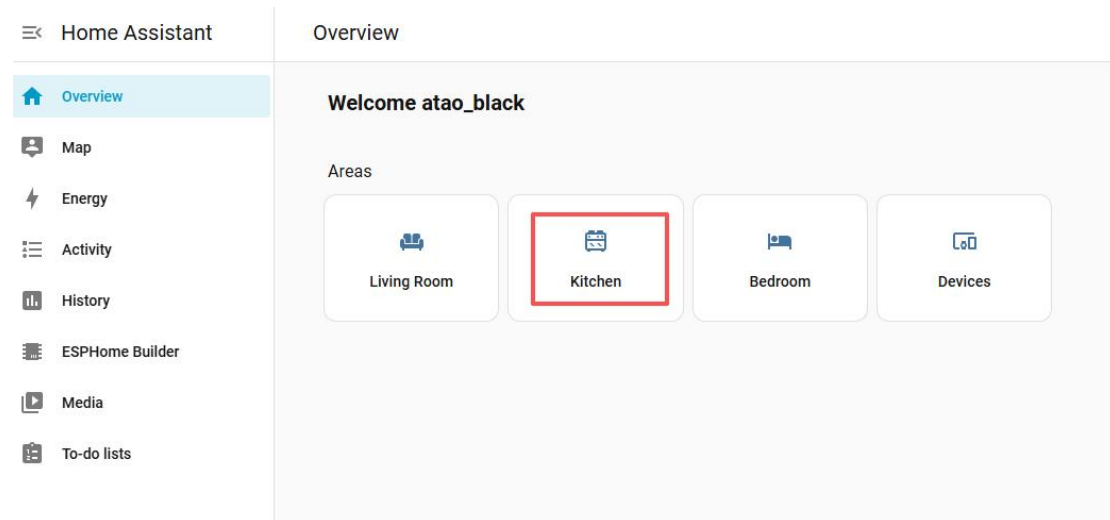
Next, when you click the switch of the LED, you will be able to see that the display light on the screen turns off, and the feedback from the LED is also quite obvious.



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.

