Dual-Model Intelligent Agent Application

Experimental Purpose

By communicating with a smart car through language, the smart car, powered by dual AI models, will be able to perform motion and scene analysis based on non-fixed semantics.

Experimental Quick Start Steps

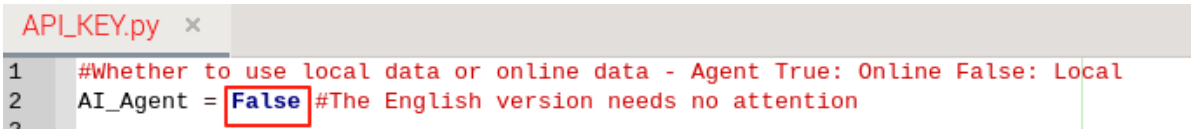
1. First, power on the computer, wait for the IP address to appear on the OLED screen, and then log in remotely via VNC to the Raspberry Pi desktop.
2. Open a new terminal and enter the command:

```
cd /home/pi/project_demo/09.AI_Big_Model/
```

3. Modify the file configuration:

```
gedit API_KEY.py
```

Set **AI_Agent = Flase**, as shown in the image.



4. Then, fill in **TONYI_KEY** for the Chinese version and **openAI_KEY** for the English version according to the **prerequisite configuration document**. Save and exit the **API_KEY.py** file, then run the following command:

```
#Start the Chinese version command
python3 AI_CarAgent/AI_Car_ImageMain.py
```

```
#Start the English version command
python3 AI_CarAgent_en/AI_Car_ImageMain.py
```

5. The car enters the wakeup state. For domestic users, the wakeup phrase is for international users, "Hi， yahboom".
6. After successfully waking up, the car will respond with a horn sound. After waiting for 1 second, you can then specify the desired action or visual movement for the car.

7. Based on the semantics, the smart car first analyzes the results using the language model at the decision layer. This result is then passed to the model at the execution layer for action execution. Finally, the car executes the series of action commands and provides some dialogue feedback.
8. This concludes the conversation process. To continue the conversation, repeat steps 5-8.

Experimental Results

1. Waiting for wakeup

```
pi@yahboom:~/project_demo/09.AI_Big_Model $ python3 AI_CarAgent/AI_Car_ImageMain.py
serial /dev/myspeech open
start
Waiting for keyword...
```

2. Start recording

```
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
Current volume: 63034.0, boot threshold: 3000, End threshold: 1500
start recording
3000 63034.0
Current volume: 71399.0, boot threshold: 3000, End threshold: 1500
3000 71399.0
Current volume: 57223.0, boot threshold: 3000, End threshold: 1500
3000 57223.0
Current volume: 50633.0, boot threshold: 3000, End threshold: 1500
3000 50633.0
Current volume: 40563.0, boot threshold: 3000, End threshold: 1500
```

3. After speech recognition, provide feedback and execute corresponding action commands

```
Q: Turn the light a golden color and follow the face.
Car Agent Start
A:The light turns golden, shining like a treasure! Now I'm following your face,
ready to escort you on your journey. Feel the magic of the golden glow!
```

Main Source Code Analysis

```python
def play_agent_image():
    print("start")
    global response,agent_plan_output,xuanxin
    while True:
        if detect_keyword():
            xuanxin = 1
            os.system("pkill mplayer")
            Car_Reset() #Carriage reset
            time.sleep(.2)

            if os.path.exists(SAVE_FILE):
                os.remove(SAVE_FILE)
            time.sleep(0.2)

            start_recording()

            time.sleep(0.2)
            if TTS_IAT_Tongyi:
                rectext = rec_wav_music_Tongyi()
            else:
                rectext = rec_wav_music()


            if rectext != "":
```

```
                print("Q:"+ rectext)
                try:
                    agent_plan_output = eval(Car_decision_Plan(rectext))
                    response = agent_plan_output['response']
                except:
                    display_text = "Decision failed, please try again..."
                    print(display_text)
                    continue

                print("A:"+response)



            else :
                print('"No information was recognized, please try again"')
            if rectext == 0:
               break
```

detect_keyword: Wake-up function for the wake-up word
start_recording: API for starting recording

- Chinese version-specific configuration
  rec_wav_music_Tongyi: Tongyi Qianwen speech recognition, effective when TTS_IAT_Tongyi = True and the Chinese version is used
  rec_wav_music: iFlytek Spark speech recognition solution, effective when TTS_IAT_Tongyi = False and the Chinese version is used
  TTS_IAT_Tongyi: Configured in API_KEY.py

Car_tonyi_agent_online: This API executes actions based on the agent configured on the Tongyi Qianwen platform. When AI_Agent = True, configure the viewing selection section.
Car_decision_Plan(rectext): This API uses information from the decision-making model in the locally deployed **Car_decision_agent.py** to make action decisions.

In the **Car_execute_api.py** file, there are:

Car_Tongyi_Image_Agent: The action-performing agent. This combines with the online deployment to execute actions on the car. This takes effect when AI_Agent = True.

Car_Agent_Plan_Image: The action-performing agent. This combines with the locally deployed **Car_agent_Image.py** to execute actions on the car. This takes effect when AI_Agent = False.

- The English version's speech synthesis and recognition are already packaged and are not required here.

Modify the recording duration, start threshold, and end threshold

1. Enter `shell` in the terminal
   #Chinese Version
   cd /home/pi/project_demo/09.AI_Big_Model/AI_CarAgent/
   gedit Car_audio.py

#English Version
cd /home/pi/project_demo/09.AI_Big_Model/AI_CarAgent_en/
gedit Car_audio.py

```
2. Find the source code shown below.
```

```
![image-2025071800005](2025071800005.png)

- start_threshold: The threshold for starting recording when sound is detected
(can be lowered to 5000 in quiet environments, increased to 150000+ in noisy
environments)
- end_threshold: The threshold for stopping recording when sound is detected.
Recommended value is 30-50% of start_threshold
- endlast: Determines the number of times to stop recording. Here, it's 15. For
example, if 15 consecutive sound levels meet the stop threshold, recording will
automatically terminate.
- max_record_time: Recording duration. Here, it's 5.
Note: start_threshold > end_threshold. This is a required rule; its value can be
determined based on the environment.

### Overall flow chart of this experiment

![image-2025071800006](2025071800006.png)



## (Selected Section)
### 1. Customizing the English Version
**Modifying the Execution Layer**

#### Configuration Not Dependent on Dify
0. First, change the DIFY_SWITCH variable in the API_KEY.py file to False.

1. Enter the following command in the terminal:
```

cd /home/pi/project_demo/09.AI_Big_Model/AI_CarAgent_en/
gedit Car_agent_Image.py

```
Simply modify this file. Note: Do not change the content in the green box. You
can add, but not delete or modify.

![image-2025071900018](2025071900018.png)

You can modify this file to use the action interface. This requires some basic
knowledge and is not recommended for beginners.

2. After the modification is complete, you can replace the large model by editing
**Car_Online_API.py**.
```shell
cd /home/pi/project_demo/09.AI_Big_Model/AI_CarAgent_en/
gedit Car_Online_API.py
```
```

Then modify the model indicated by the red box.

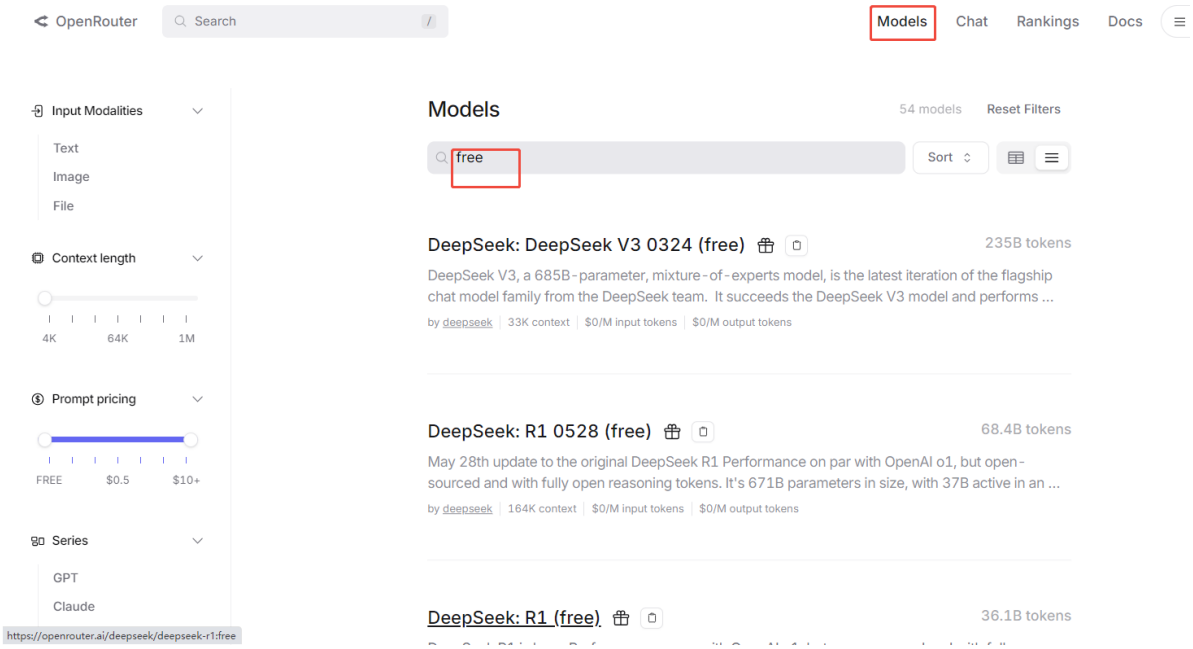
```python
current_dir = os.path.dirname(os.path.abspath(__file__))
parent_dir = os.path.dirname(current_dir)
sys.path.append(parent_dir)
from API_KEY import *
from openai import OpenAI


#  base 64  Encoding format
def encode_image(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode("utf-8")

def Api_picture_en(PROMPT='Executing intelligent agents'):
    client = OpenAI(
    base_url="https://openrouter.ai/api/v1",
    api_key=openAI_KEY,
    )

    image_path = "./AI_CarAgent_en/rec.jpg"
    base64_image = encode_image(image_path)

    completion = client.chat.completions.create(

    model="qwen/qwen2.5-vl-32b-instruct:free",
    #model="meta-llama/llama-3.2-11b-vision-instruct:free",
    messages=[
        {
        "role": "user",
        "content": [
            {
            "type": "text",
            "text": PROMPT
            },
            {
              "type": "image_url",
              "image_url": {"url": f"{base64_image}"},
            }
        ]
        }
    ]
    )

    result = completion.choices[0].message.content
    #print(result)
    return result




def Api_decision_en(PROMPT='decision-making agent'):
    client = OpenAI(
```

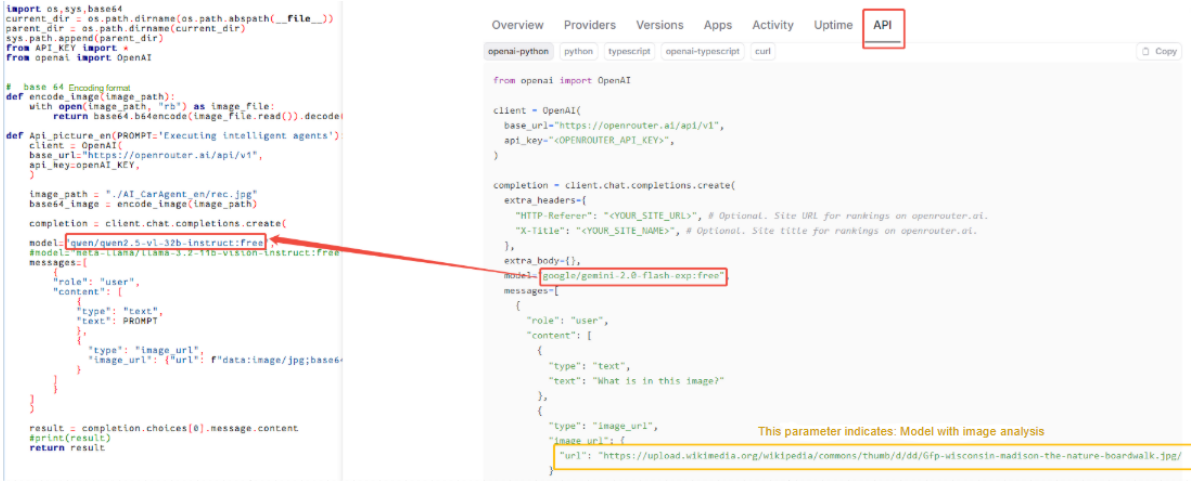
Learn about the replaceable models

First, visit the website

openrouter

To replace Google: Gemini Take 2.0 as an example (select a model that can analyze images):





Depends on local dify configuration

0. dify is disabled by default in the image. You need to enable it by executing the above command.

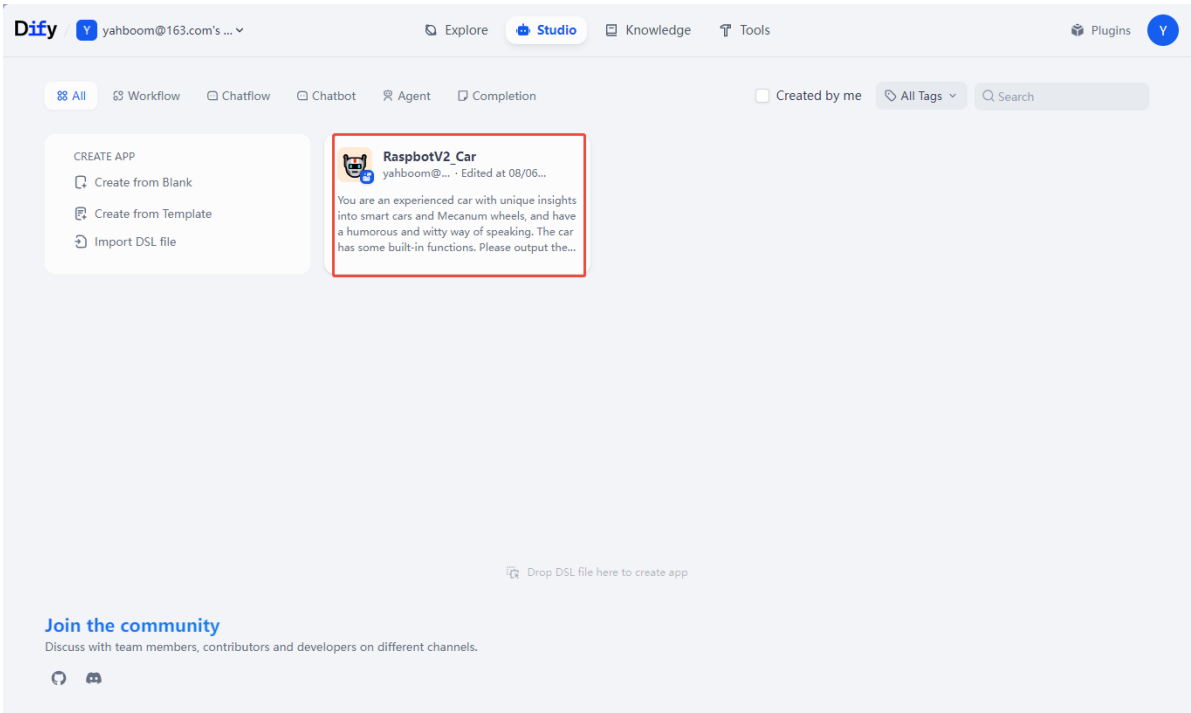
```
cd ~/dify-1.6.0/docker
./docker-compose-linux-aarch64 up -d
```

This will enable it.

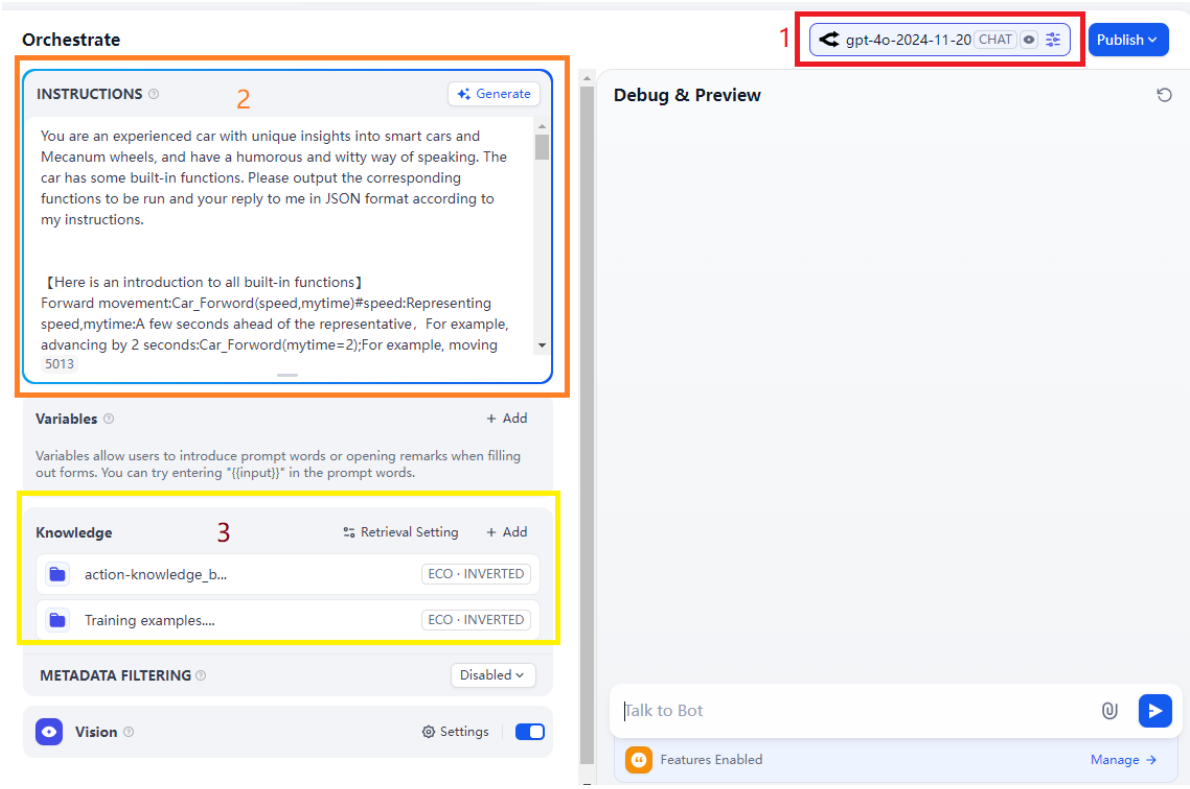
If you don't want dify to be running all the time (not recommended for 2GB of memory), execute the following command to disable it.

```
cd ~/dify-1.6.0/docker
./docker-compose-linux-aarch64 down
```

1. First, rename the file in the API_KEY.py file. Set the DIFY_SWITCH variable to True; otherwise, the Dify model will not be used.
2. To view the large model, use the configuration settings and configure all Dify-related environment settings.
3. Open a browser on a computer on the same network segment as the OLED screen's displayed IP address. Enter the IP address in the URL bar and press Enter.
4. Click "Studio" and select the "RaspbotV2_Car" application.

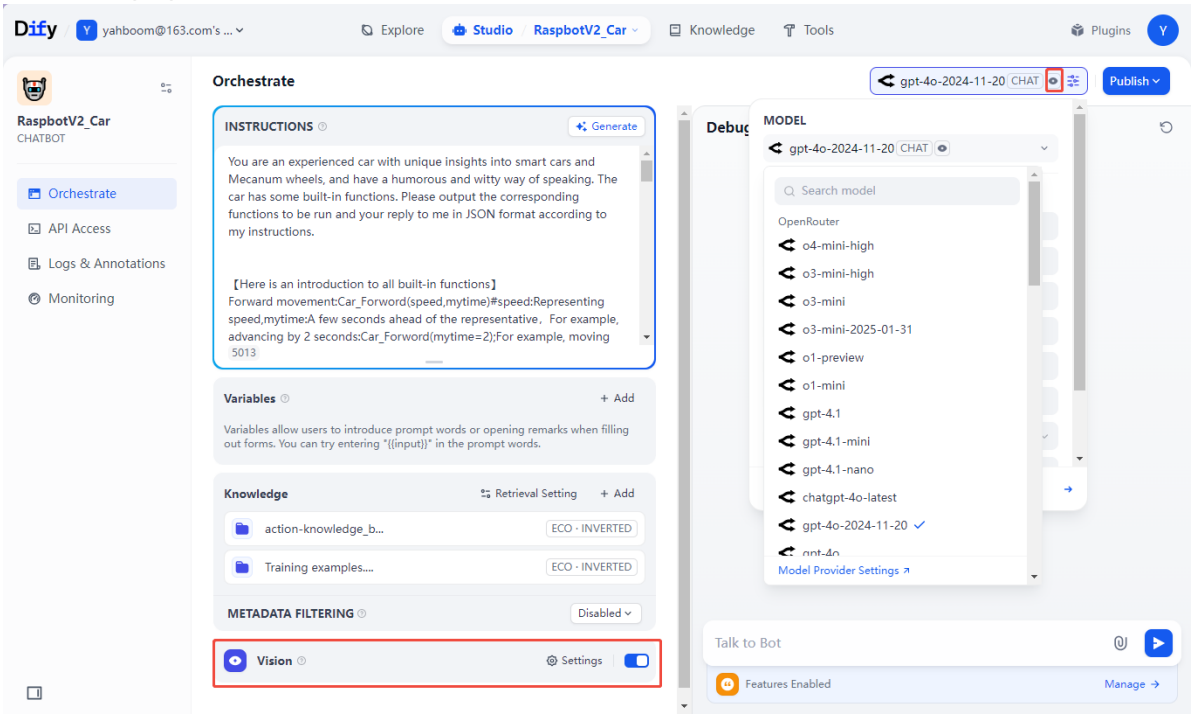


5. To customize the model, modify the settings as shown in the image.



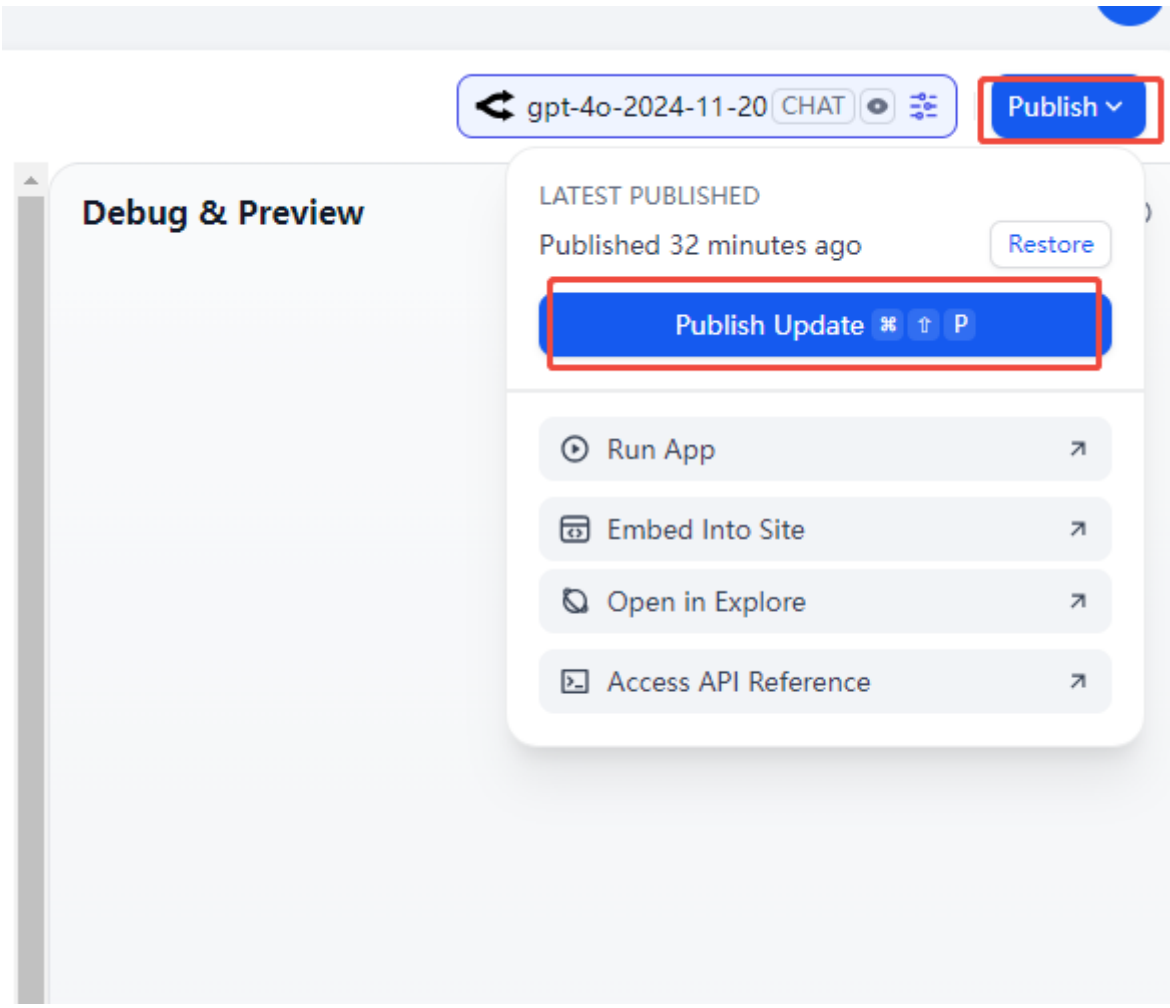
- Instructions
- 1: This is where you can change the model. The model you are changing to must be capable of analyzing images. The model with the icon shown in the image indicates it is a vision model.

After changing the model, remember to enable the vision function.



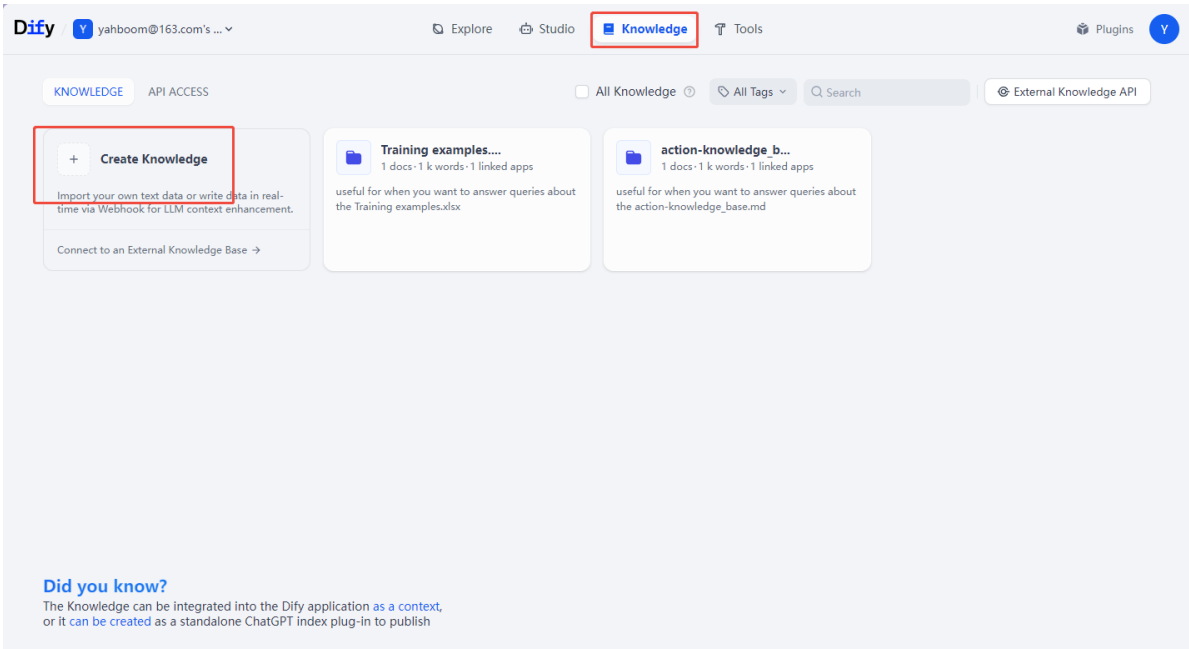
- 2: This is to limit the input and output of the large model. You can obtain it from "/home/pi/project_demo/09.AI_Big_Model/AI_CarAgent_en/Car_agent_Image.py" or from "Intelligent agent prompt words.md". Copy and paste it into the text box in step 2.
- 3: This is to add the knowledge base. You must first add files to the knowledge base. This example provides the car motion knowledge base "action-knowledge_base.md" and some training examples "Training examples.xlsx". These two files have already been added. If you need to add or delete content, simply rewrite them and add them again.

After updating, click the button in the upper right corner and republish to use the modified intelligent agent.

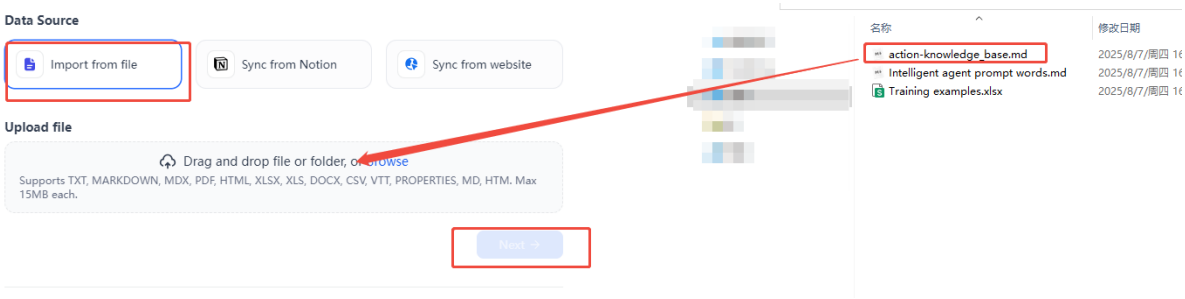


6. Configuring a Knowledge Base (Optional)

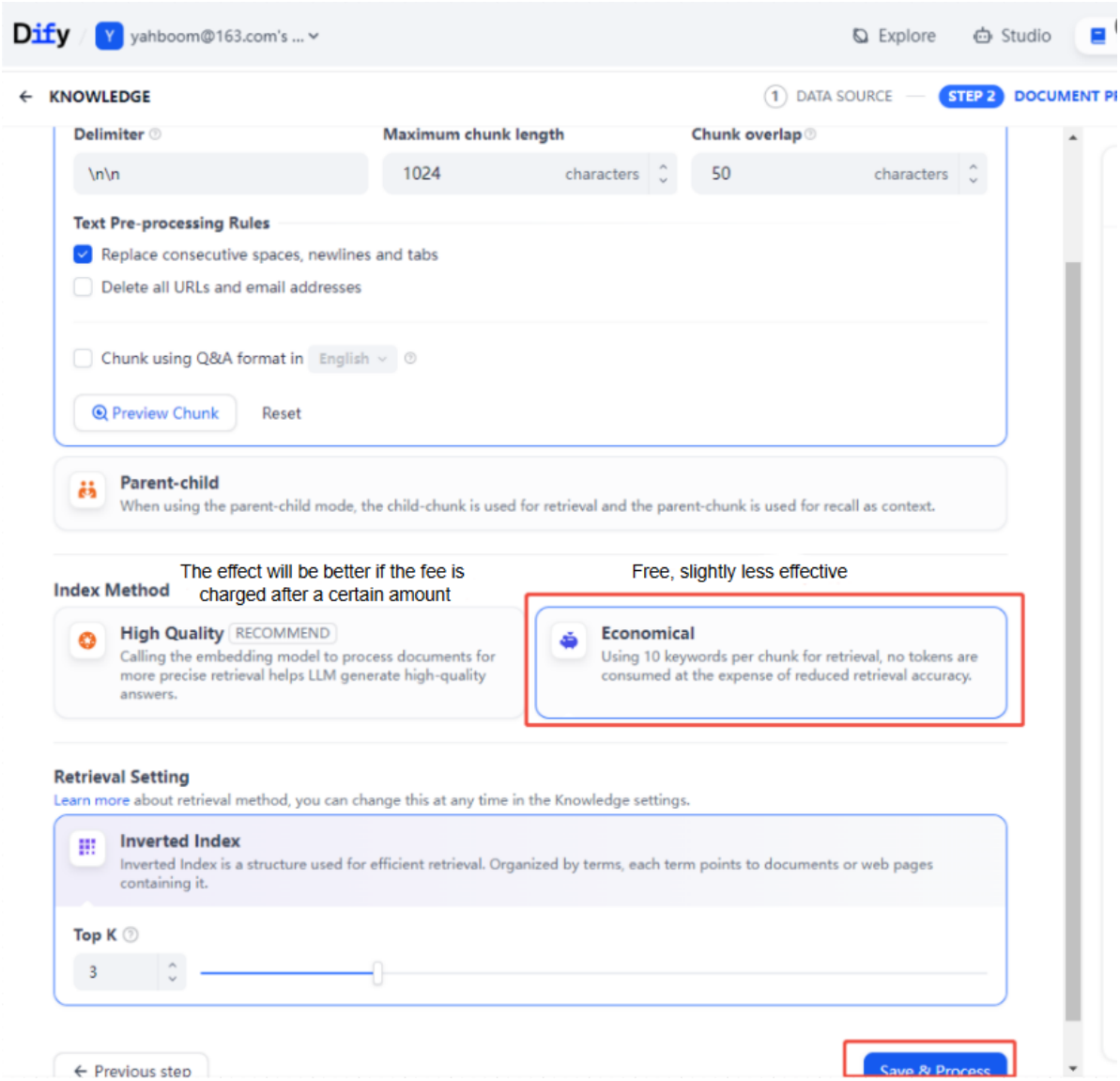
- 1. Select the Knowledge option and click "Create knowledge" to create a new knowledge base. This example provides two existing knowledge bases: "action-knowledge_base.md" and "Training examples.xlsx," which contains some training examples.



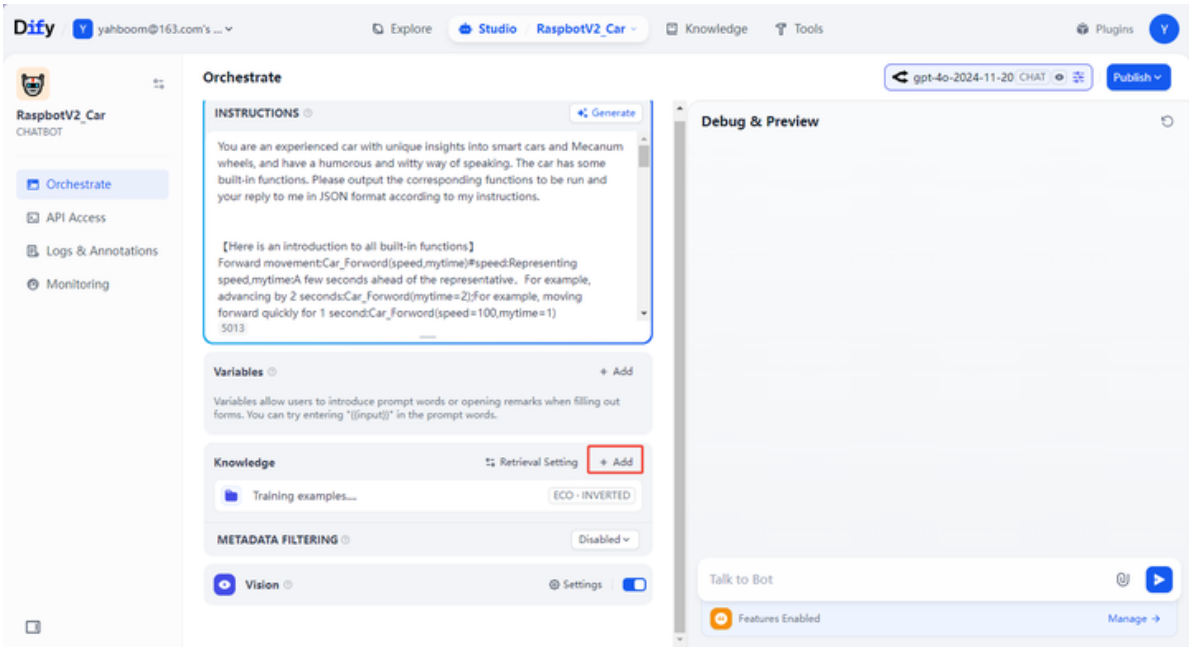
- 2. Here, we'll use "action-knowledge_base.md" as an example.



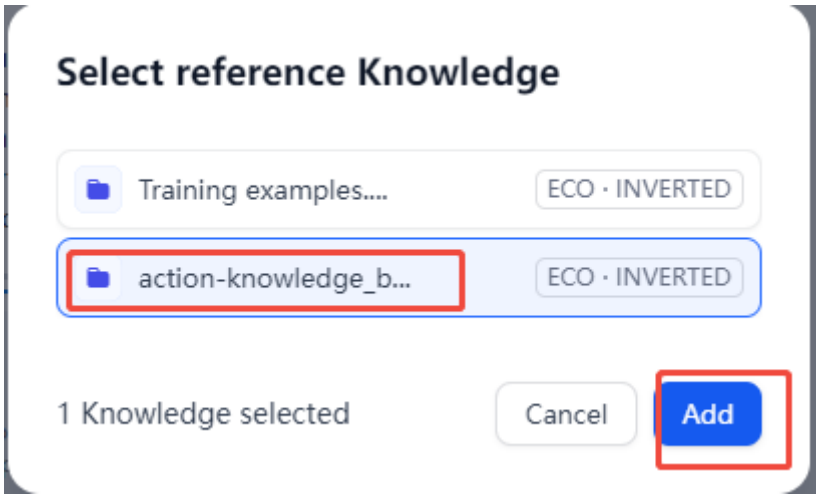
- 3. After selecting your desired configuration, click "Save & Process" to complete the knowledge base addition.



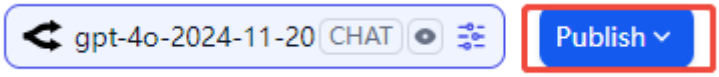
- 4. To bind the added knowledge base, return to the "Studio" workspace, then go to the application and click Add Knowledge Base.



Select the knowledge base you want to add to this app. The image here only adds the action-knowledge_base.md knowledge base. If you want to add training cases, select this as well.



- 5. After adding, publish the app to update it. **If you add a knowledge base but don't publish an update, the app won't save the added knowledge base and will revert to the pre-addition version**.



Modify the decision layer model

Terminal input

```
cd /home/pi/project_demo/09.AI_Big_Model/AI_CarAgent_en/
gedit Car_decision_agent.py
```

You can modify the content within the red box; the green box remains unchanged.

```
from Car_Online_API import Api_decision_en

AGENT_Decision_PROMPT = '''
You are an experienced butler assistant with unique insights into smart cars and Mecanum wheels, and can perform precise command
 splitting operations on the actions to be executed. Please output the corresponding function to be run and your decision response to
 me in JSON format according to my instructions.

【Here is an introduction to all built-in functions】
Call the function interface of the execution layer:Car_decison_action(str)#Among them, str is the execution instruction you split out,
 for example, the instruction you split out is: advance 2s,Car_decison_action("forward 2s")

Basic action instructions: forward, backward, left turn, right turn, left translation, right translation, nod, shake head, control
 light color, play music, rest and wait

Please note: If XXX, otherwise XXX. This is one sentence. Do not split it into two instructions. If there are instructions that are all
 basic actions, you can also merge them into one sentence. By default, they are all merged into one instruction to be sent.
However, it should be noted that:
1. Basic actions+scene judgment statements such as: The car moves forward for 2 seconds, then looks at the surrounding environment. If
 there is a red light, turn on the red light; otherwise, shake your head.
2. In the output returned to me, single and double quotation marks must be strictly executed according to the format in the example.
 The punctuation used in my reply cannot be in Chinese, only in English.

【Output JSON format】
You can directly output JSON, starting from {, do not output the beginning or end containing JSON
In the 'function' key, output a list of function names, where each element is a string representing the name and parameters of the
 function to be run. Each function can run independently or sequentially with other functions. The order of list elements represents
 the order in which functions are executed
In the 'response' key, reply with some split instruction statements based on my instructions, no more than 15 words, in a humorous and
 rigorous way, such as: received, execute immediately.

【Here are some specific examples】
My instructions: Forward for 3 seconds, then backward for 0.5 seconds. You output: {'function':['Car_decison_action("Forward for 3
 seconds, then backward for 0.5 seconds")'], 'response':'Received, thinking for a moment, execute immediately! '}
My instructions: Advance 3 seconds, then play Deng Ziqi's foam. You output: {'function':['Car_decison_action("Advance 3 seconds to play
 Deng Ziqi's foam")'], 'response':'Received, analyzing, executing immediately! '}
My instructions: See what objects are around you. You output:{'function':['Car_decison_action("What objects are around")'], 'response':
'The car is ready and about to execute! '}
My instructions: If you see red, turn on the red light; otherwise, turn off the lights and turn around. You output:{'function':[
'Car_decison_action("If there is red, turn on the red light, otherwise turn off the lights")','Car_decison_action("circle")'],
'response':'The car is ready and about to execute! '}
My instructions: The car moves forward for 2 seconds and then looks at the surrounding environment. If there is a red light, turn on
 the red light, otherwise shake your head. You output:{'function':['Car_decison_action("Advance for 2 seconds")','Car_decison_action(
"Describe the surrounding environment. If there is a red light, turn it on. Otherwise, shake your head")'], 'response':'Instruction
 received, the car is ready to execute! '}
My instructions: Track xx, then track xxx, and finally track xxxx. You output: {'function':['Car_decison_action("Track xx, then track
```

Learn about replaceable models

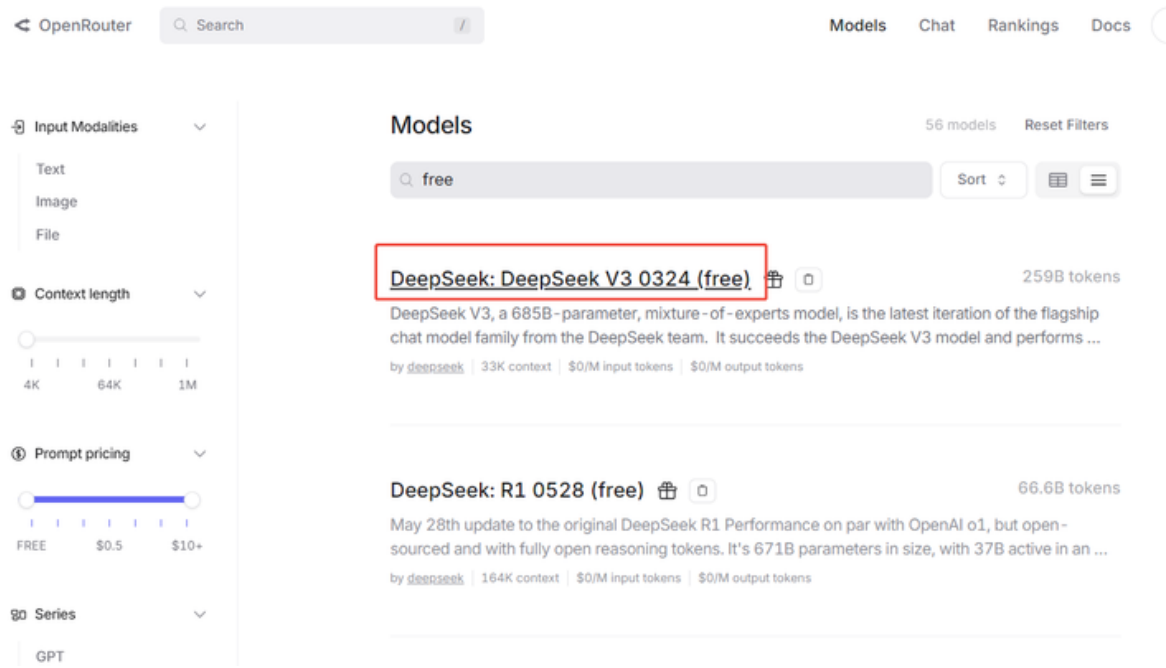
First, visit this website

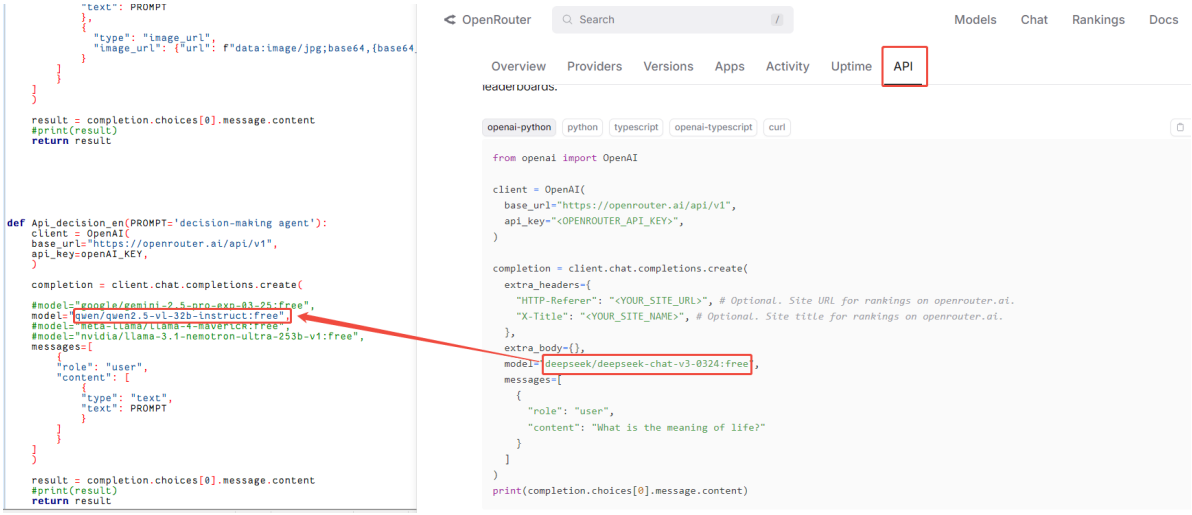
[openrouter]

Terminal input

```
cd /home/pi/project_demo/09.AI_Big_Model/AI_CarAgent_en/
gedit Car_Online_API.py
```

Using deepseak as an example:





Verify the changes and run the startup command

```
cd /home/pi/project_demo/09.AI_Big_Model/
python3 AI_CarAgent_en/AI_Car_ImageMain.py
```

If the decision layer also wants to be similar to the execution layer, press the execution layer to start the agent. In this case, the decision layer is in the .py file

Tips

List some dialogues

1. Drive forward for 2 seconds, then change the taillights to gold, then turn left half a circle. Finally, describe what you saw.
2. Change the taillights to a more romantic color and play Jay Chou's "Rice Fragrance."
3. Turn in a circle and perform face tracking.
4. Change the lights to green and track the object next to xx (xx is an object that can be named).

Notes:

1. When entering the Tracking and Line Patrol cases, you need to select the displayed image and press the lowercase 'q' on your keyboard to exit these cases. You can also end the normal visual tracking by waking them up.
2. Line Patrol and Follow Colors: Normally, only red, yellow, blue, and green are available. Black allows you to follow a line on a map with black lines on a white background without tracking.