Embodied Intelligence Core Source Code Interpretation

1. Course Content

1. AI Large Model Embodied Intelligence is a complex function that involves the coordinated implementation of multiple node programs. This section explains the core programs.

2. Overall Source Code Package Structure

2.1 Functional Package Structure

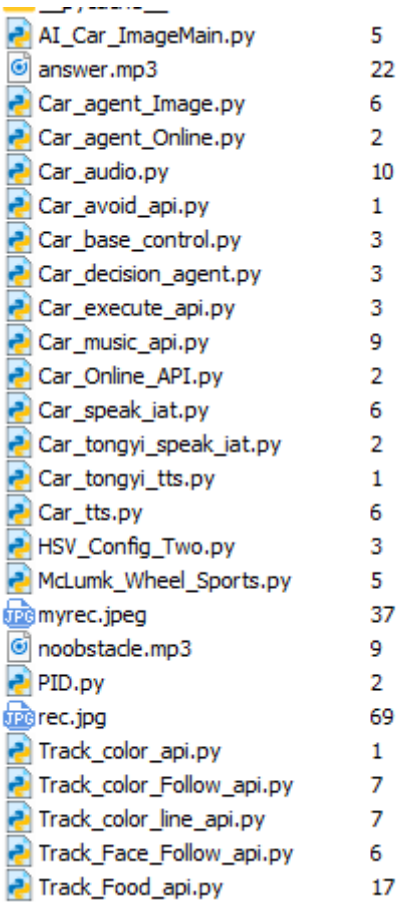
Dual-Model AI Large Model Embodied Intelligence:

Jetson Orin Nano, Jetson Orin NX Hosts:

```
#Chinese version
/home/pi/project_demo/09.AI_Big_Model/AI_CarAgent/

#English version
/home/pi/project_demo/09.AI_Big_Model/AI_CarAgent_en/
```

The file structure is as follows:



The folder and file functions are as follows:

```
├── AI_Car_ImageMain.py # Main entrance of dual model program
├── Car_agent_Image.py #Local agent description (execution layer)
├── Car_agent_Online.py #Depends on the cloud-based agent description (execution
layer)
├── Car_audio.py #Recording files
├── Car_avoid_api.py #Obstacle avoidance and ranging file
├── Car_base_control.py #Basic sports movements
├── Car_decision_agent.py #Decision-making model
├── Car_execute_api.py #Calling the execution layer model
├── Car_music_api.py #Audio playback
├── Car_Online_API.py #Online multimodal model interface
├── Car_speak_iat.py #Speech Recognition 1
├── Car_tongyi_speak_iat.py #Speech Recognition 2
├── Car_tongyi_tts.py #Speech Synthesis 1
├── Car_tts.py #Speech Synthesis 2
├── HSV_Config_Two.py #Color Analysis
├── McLumk_Wheel_Sports.py #Car chassis control library
├── PID.py #Tracking and line patrol PID interface
├── Track_color_api.py #Tracking and line patrol interface
├── Track_color_Follow_api.py #Color follow function
├── Track_color_line_api.py #Color line patrol function
├── Track_Face_Follow_api.py #Face tracking function
└── Track_Food_api.py #Object tracking function
```

2.2 Inter-program call relationship diagram

3. Speech Recognition Function

The speech recognition function includes two parts: VAD voice activity detection (VAD) and speech-to-text conversion.

3.1 VAD Voice Activity Detection

Implementation: The listen_for_speech method in the ASRNode class

Program Explanation: Records audio in real time from a specified microphone and uses VAD (Voice Activity Detection) to determine whether speech is currently occurring. When a segment of speech is detected (continuous silence exceeds a set number of frames), recording stops and the valid audio content is saved to a file.

Detailed logic is as follows:

1. Initialize the audio stream and configure parameters (such as sampling rate and number of channels).
2. Continuously read audio frames and perform voice activity detection.
3. If speech is detected, the audio frame is added to the buffer. If continuous silence exceeds a threshold (90 frames, approximately 1 second), recording ends.
4. After recording is complete, the trailing silence is removed and the valid audio content is saved as a WAV file.
5. If no valid speech is detected, the file is not saved.

The specific recording source code path is:

```
/home/pi/project_demo/09.AI_Big_Model/AI_CarAgent/Car_audio.py
```

3.2 ASR Speech Recognition

Implementation Program: Convert the recording file into text

Call the speech recognition model interface functions in the large model interface file Car_tongyi_speak_iat.py

Program Explanation:

```
def rec_wav_music_Tongyi():
    global rec_text
    recognition.start()
    try:
        audio_data: bytes = None
        f = open(DIR_PATH_WAV, 'rb')
        if os.path.getsize(DIR_PATH_WAV):
            while True:
                audio_data = f.read(3200)
                if not audio_data:
                    break
                else:
                    recognition.send_audio_frame(audio_data)
                time.sleep(0.1)
        else:
            raise Exception(
                'The supplied file was empty (zero bytes long)')
        f.close()
    except Exception as e:
        raise e

    recognition.stop()

    return rec_text
```

rec_wav_music_Tongyi The recorded wav file will be uploaded to the speech recognition model, and the speech recognition result will be returned to the function using this interface.

4. Dual-Model Inference

4.1 Dual-Model Inference (International Version)

1. The program implementation logic is the same as the domestic version, with the difference being in API_KEY.py.

- DIFY_SWITCH = True: The international version requests the local Dify application, which then requests the large cloud model.
- DIFY_SWITCH = False: The international version requests the agent in the local Python file.

```
def Car_decison_action(decision_str='Decision makers input parameters'):
    print("Action:"+decision_str)
    try:
        if DIFY_SWITCH ==False:
            agent_plan_execute = eval(Car_Agent_Plan_Image(decision_str))
        else:
            agent_plan_execute = eval(Car_Agent_Plan_Image_Dify(decision_str))
#dify
```

```
        exeute_relay = agent_plan_execute['response']
    except:
        display_text = "Error in obtaining action information, please try
again..."
        print(display_text)
        return


    for each in agent_plan_execute['function']: # Run each function of
intelligent agent planning and orchestration
        if xuanxin == 1:
            print("Interrupted by wake-up, please rephrase the command")
            return

        try:
            print('Start executing action', each)
            eval(each)
        except:
            g_fail_ex = g_fail_ex+1
            if g_fail_ex == 1: # Action execution failed
                print("Action try Start again!")
                os.system("pkill mplayer") # Only this time will thread audio be
played
                time.sleep(0.5)
                Xinghou_speaktts('Action execution failed, prepare to try
again')
                return 1
            elif g_fail_ex>1:
                print("Action fail!")
                Xinghou_speaktts('Action execution failed, end task
prematurely')
                g_fail_ex =0
                return 2
```

5. Interruption Functionality

The robot supports interruption at any stage, specifically during the conversation phase and the action phase. The principles of interruption in each phase are explained here.

5.1 Interruption During the Conversation Phase

If you are dissatisfied with the robot's response or do not want it to continue, you can interrupt it using the wake-up word and begin recording. You can then speak new commands to the robot (while still in the current task cycle).

- Simply determine whether the wake-up word flag is detected to end the current task cycle and start a new one. You can also terminate all currently playing audio using os.system("pkill mplayer").

```
if detect_keyword():
        xuanxin = 1
        os.system("pkill mplayer")
```

5.2 Interrupting the Action Phase

If the robot is awakened during an action, it will stop the current action and resume its initial position. This can be categorized as either a standard action interruption or an action interruption with a child process.

Based on the wakeup flag, the currently executing child thread task and robot motion action can be terminated using a special process ID (PID), causing the robot to stop.

```
#Kill after interruption
def PIDkill_process(KillPath):
        try:
            # Use pgrep to find scripts
            result = subprocess.run(['pgrep', '-f', KillPath],
capture_output=True, text=True, check=True)
            pids = result.stdout.strip().split('\n')
            # Iterate over all found PIDs
            for pid in pids:
                try:
                    # Terminate a process
                    subprocess.run(['kill', pid], check=True)
                    print(f"Process {pid} has been terminated.")
                except subprocess.CalledProcessError:
                    print(f"Failed to terminate process {pid}.")
        except subprocess.CalledProcessError:
            pass
            #print("No matching processes found.")


PIDkill_process('./AI_CarAgent/Track_color_Follow_api.py')
PIDkill_process('./AI_CarAgent/Track_color_line_api.py')
PIDkill_process('./AI_CarAgent/Track_Face_Follow_api.py')
Car_Reset() #Car reset
```