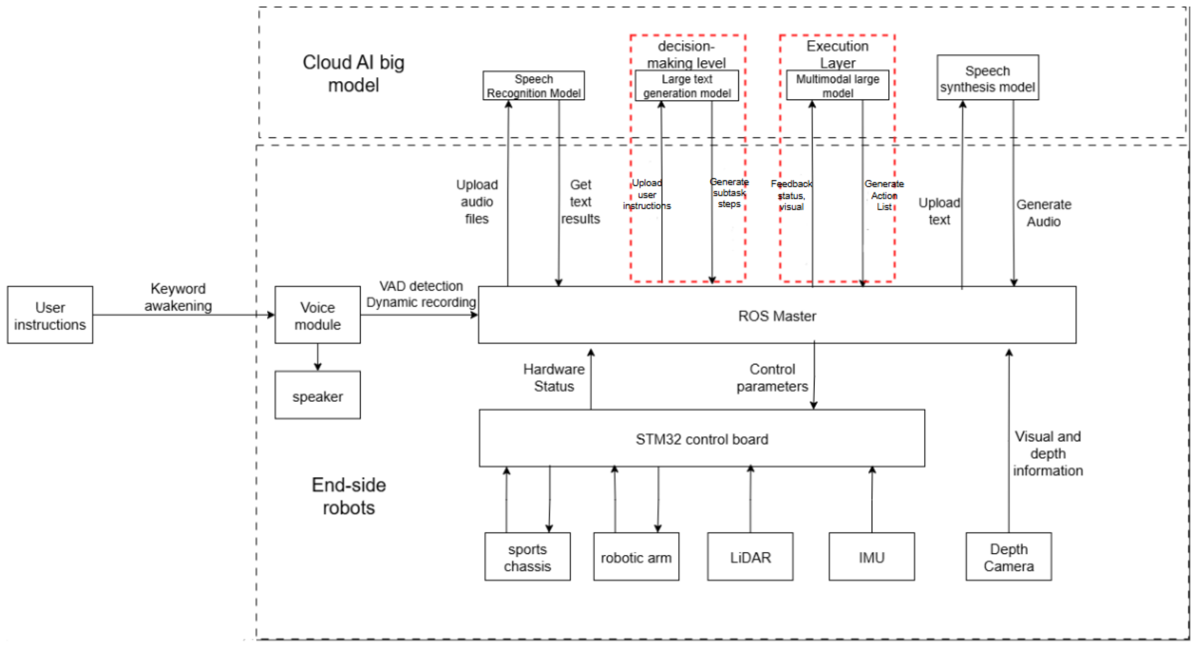
Embodied Intelligent Robot System Architecture

1. Course Content

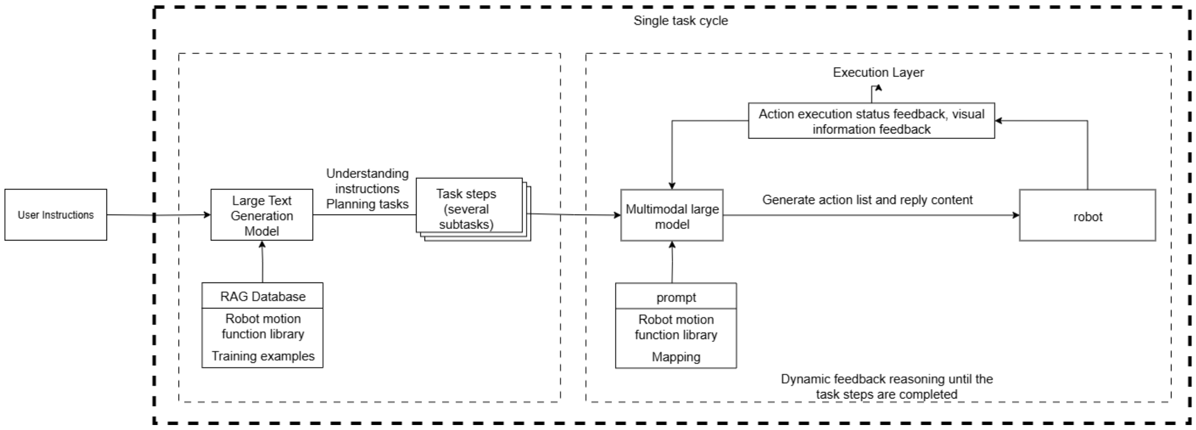
1. Explain the Big Model Reasoning Architecture of the AI Embodied Intelligent Robot
2. Explain the basic concepts of the system to lay the foundation for practical application in subsequent courses

2. System Components



3. AI Big Model Reasoning Architecture Diagram

The robot's programming utilizes dual-model reasoning and dynamic feedback i-reasoning, improving its processing capabilities for long, complex tasks and achieving greater system robustness compared to single-model architectures. Several key concepts are discussed: the decision-layer big model, the execution-layer big model, the task cycle, and the conversation history, each of which will be explained below.



Single task cycle
Execution Layer
Action execution status feedback, visual information feedback
User Instructions
Large Text Generation Model
Understanding instructions Planning tasks
Task steps (several subtasks)
Multimodal large model
Generate action list and reply content
robot
RAG Database
Robot motion function library
Training examples
prompt
Robot motion function library
Mapping
Dynamic feedback reasoning until the task steps are completed

4. Explanation of the Large Model Inference Architecture

4.1 Principles of the Dual-Model Architecture

- The robot's large model control system is designed based on a text-generated large model and a multimodal large model. The text-generated large model serves as the **decision layer**, acting as a high-level task planner and decomposing complex and abstract human instructions.
- The multimodal large model serves as the execution layer, receiving task steps generated by the **decision layer** and temporary user instructions (typically simple instructions instructing the robot to end a task or rest), monitoring the robot's progress, adjusting its execution in real time, and generating a list of action functions that the robot can interpret and responding to the user.
- The action function list contains pre-programmed basic action functions that directly output parameters for controlling the robot's motion.

4.2 Advantages of the Dual-Model Inference Architecture

4.2.1 Decoupling Decision Logic and Action Control

- Large multimodal models lag behind large text-generated models in their ability to understand natural language semantics and perform logical reasoning. They are also prone to modal interference when performing complex tasks. Therefore, the system design decouples decision logic from action control.
- The large text generation model focuses on semantic understanding and task planning. This is just an example and may not be the current product's functionality. (For example, parsing the sentence "Can you help me get the blue square in room A?" breaks down the task into the following steps: "Record current location → Navigate to room A → Obtain the current robot-view image → Locate the blue square's coordinates → Grab the blue square → Return to the starting position → Drop the blue square.") This avoids the complexity of traditional single models that must simultaneously handle action details during semantic parsing.
- The large multimodal model is responsible for action generation and environment interaction (for example, acquiring visual images to locate object coordinates and outputting action functions and parameters). This reduces the confusion and large deviations in action logic that can occur in single models due to task complexity.

4.2.2 Reducing Model Training Sample Complexity

When using a single model for inference, the large multimodal model must simultaneously perform "natural language understanding + environmental perception + process decomposition + action function output," which can easily lead to modal interference (misinterpretation due to ambiguous language or overly long instructions). The dual model uses phased processing, allowing the decision layer to focus on language-space mapping and the execution layer to focus on vision. Motion Space Mapping.

4.2.3 Flexible Expansion

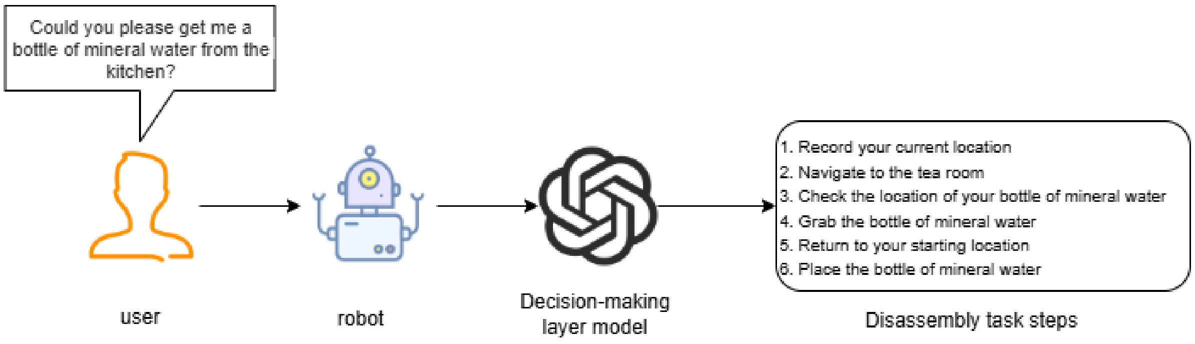
The decision-making and execution layer big models can be flexibly combined using a variety of big models on the Alibaba Bailian Big Model Platform (sometimes also called the Tongyi Qianwen Platform). Through customized training samples and RAG knowledge bases, the models can be adapted to tasks in different scenarios, greatly enhancing the generalization capabilities of AI embodied intelligent robots in different scenarios.

4.3 Decision-making Big Model

4.3.1 Function of the Decision-making Big Model

- The robot's decision-making big model uses the Tongyi Qianwen series of models by default.
- The decision-making big model is primarily responsible for task planning. It can understand complex human instructions and break them down into specific task steps. **This is just an example and does not necessarily reflect the current product functionality**. For example, when receiving the command "Can you help me get a bottle of mineral water from the kitchen?" The large language model breaks it down into several tasks, as shown in the figure below.

Each task step corresponds to the minimum action the robot can perform. The execution layer large model then implements the specific robot movement by calling API functions in the robot's action function library.


Could you please get me a bottle of mineral water from the kitchen?
user
robot
Decision-making layer model
1. Record your current location
2. Navigate to the tea room
3. Check the location of your bottle of mineral water
4. Grab the bottle of mineral water
5. Return to your starting location
6. Place the bottle of mineral water
Disassembly task steps

4.3.2 Robot Action Library (Simplified)

The robot action library specifies the minimum actions that all robots can actually perform. The decision-making model selects and arranges appropriate actions from this library when planning task steps.

Basic Action Class

- Turn Left
- Turn Right
- Dance
- Drift
- Forward
- Backward
- Pan Left
- Pan Right
- Change Light Bar Color
- Play Music
- Nod
- Shake Head

Ultrasonic Distance Measurement Class

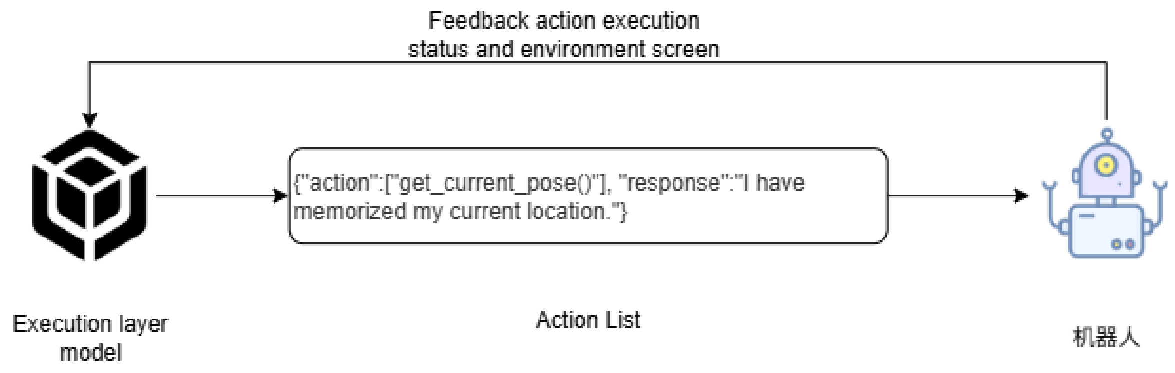
- Obtain the current obstacle distance

Image Acquisition Class

- Obtain a description of the current viewport image
- Obtain a target in the image for tracking/following

4.4 Execution Layer Model

- The default execution layer model for the robot uses the Tongyi Qianwen-VL series model. -VL: A model capable of visual analysis.
- The execution layer model is primarily responsible for generating the action list that controls the robot's movements and responding to the user. As the robot executes the action list, it continuously receives feedback on the robot's action results (success/failure) and visual images. Based on the success or failure of the action, it infers the next action to be performed.
- The execution layer model acts as a supervisor, continuously monitoring the robot's progress in executing task steps. It uses the feedback from the robot's actions and environmental information to determine the next action to be performed, until the task is successfully completed or terminated prematurely due to special circumstances.



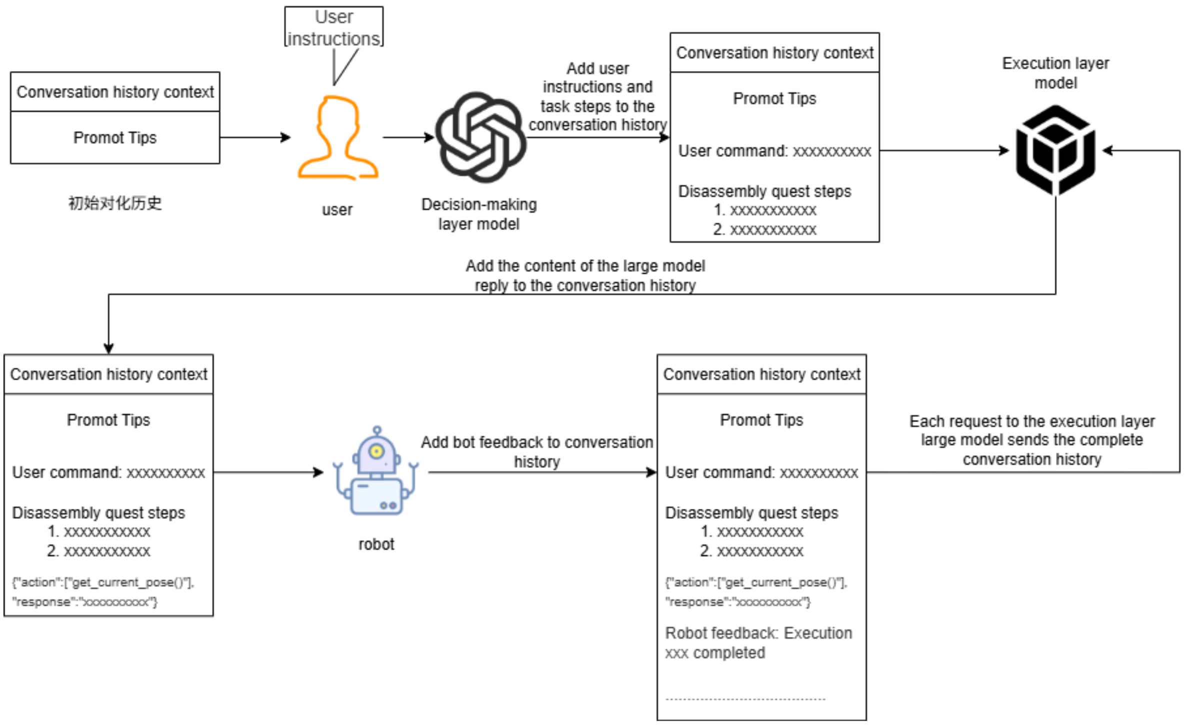
```
{"action":["get_current_pose()"], "response":"I have memorized my current location."}
```

4.4.1 Task Cycle

- In a new task cycle, user commands first pass through the decision layer model and then the execution layer model. If the execution layer model determines that the robot has completed all task steps, it enters a **sleep state**.
- For example, when a new task cycle ends, the robot enters a sleep state and needs to be awakened to start a new task cycle. At this point, all states of the robot are reset.

4.4.2 History Context

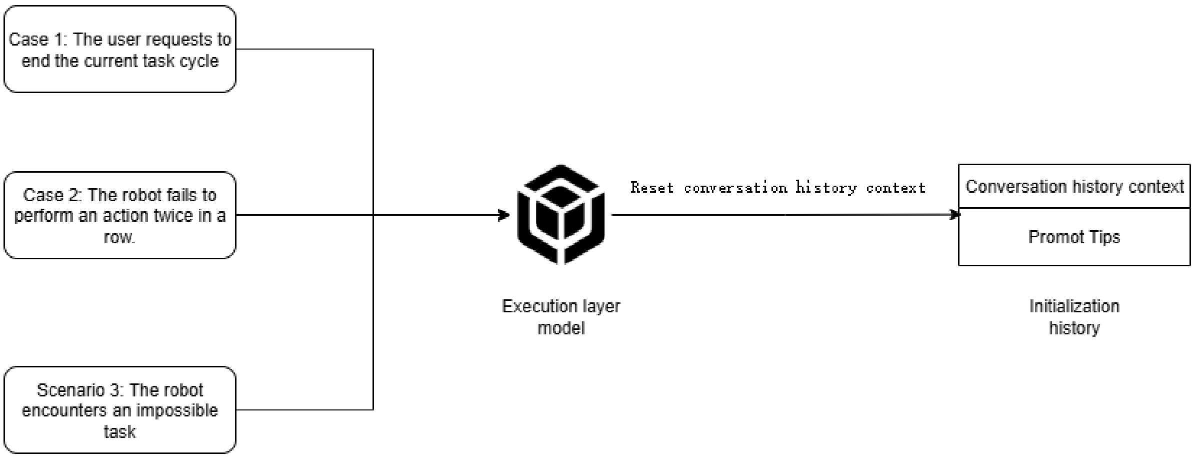
The robot maintains a conversation history context in its local program. This context includes user commands, task steps generated by the decision-layer model, the action list generated by the execution-layer model, and status information provided by the robot. Each time the robot requests the execution-layer model, it sends the complete conversation history. This allows the execution-layer model to understand the robot's progress in executing tasks. The execution-layer model then uses the conversation history context to infer the next action to be performed. In short, the execution-layer model infers the next action based on the sum of past states.


User instructions
Conversation history context
Promot Tips
初始对化历史
user
Decision-making layer model
Add user instructions and task steps to the conversation history
Conversation history context
Promot Tips
User command: xxxxxxxxxx
Disassembly quest steps
1. xxxxxxxxxxx
2. xxxxxxxxxxx
Execution layer model
Add the content of the large model reply to the conversation history
Conversation history context
Promot Tips
User command: xxxxxxxxxx
Disassembly quest steps
1. xxxxxxxxxxx
2. xxxxxxxxxxx
{"action":["get_current_pose()"], "response":"xxxxxxxxxx"}
robot
Add bot feedback to conversation history
Conversation history context
Promot Tips
User command: xxxxxxxxxx
Disassembly quest steps
1. xxxxxxxxxxx
2. xxxxxxxxxxx
{"action":["get_current_pose()"], "response":"xxxxxxxxxx"}
Robot feedback: Execution xxx completed
Each request to the execution layer large model sends the complete conversation history

Under normal circumstances, the execution-layer model determines the progress of task completion based on the conversation history. When the robot determines that all task steps have been completed, it enters a sleep state. This state clears the conversation history and ends the current task cycle. Upon detecting the wake-up word, a new task cycle begins.

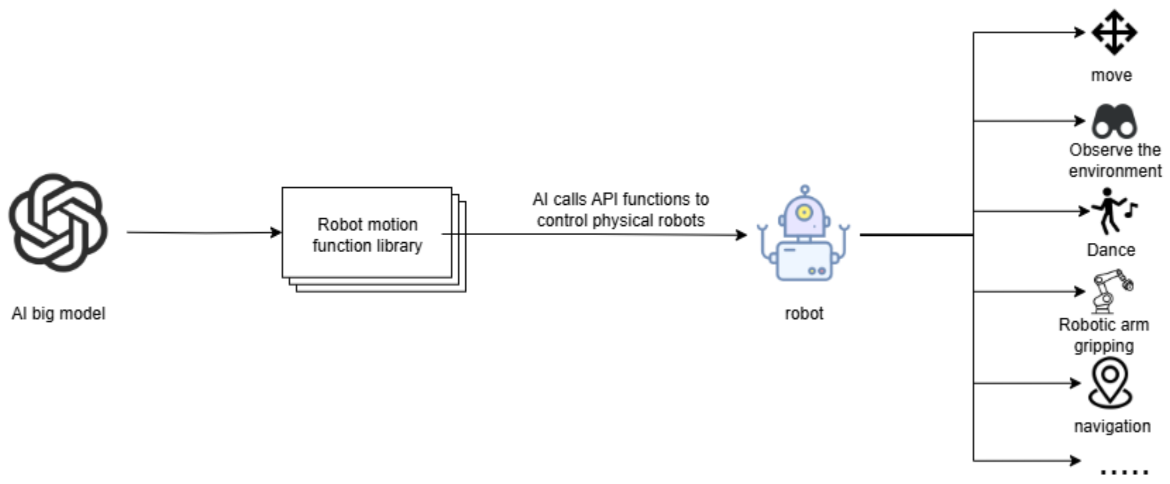
During task execution, if a task cannot be executed, the current task cycle will also end.

- Case 1: The robot fails twice consecutively while executing an action. If an action fails, the execution layer model will instruct the robot to retry at most once. If it fails again, the execution layer model will instruct the robot to prematurely end its task cycle and start a new one.
- Case 2: The robot encounters an unachievable task


Case 1: The user requests to end the current task cycle
Case 2: The robot fails to perform an action twice in a row.
Scenario 3: The robot encounters an impossible task
Execution layer model
Reset conversation history context
Conversation history context
Promot Tips
Initialization history

4.4.3 Robot Action Library

The API functions in the robot action library serve as a bridge between the large model and the real world. These API functions define the minimum actions that the physical robot can perform in the physical world. The API functions operate by controlling the underlying hardware of the physical robot through function interfaces to perform various functions.



All action functions and their corresponding functions are shown in the following table:

Function Name	Parameters	Function Implementation	Call Instructions
Car_Forward(speed,mytime)	speed: represents the speed (fast, medium, or slow), mytime: represents the number of seconds to move forward	how long to move forward at a certain speed	how long to move forward at a certain speed
Car_left(speed,mytime)	speed: represents the speed (fast, medium, or slow), mytime: represents the number of seconds to move forward	how long to move backward at a certain speed	how long to move backward at a certain speed
Car_right(speed,mytime)	speed: represents the speed (fast, medium, or slow), mytime: represents the number of seconds forward	how long does the right turn go at this speed?	how long does the right turn go at this speed?
Car_left_translation(speed,mytime)	speed: represents the speed (fast, medium, or slow), mytime: represents the number of seconds forward	how long does the left turn go at this speed?	how long does the left turn go at this speed?
Car_right_translation(speed,mytime)	speed: represents the speed (fast, medium, or slow), mytime: represents the number of seconds forward	how long does the right turn go at this speed?	how long does the right turn go at this speed?

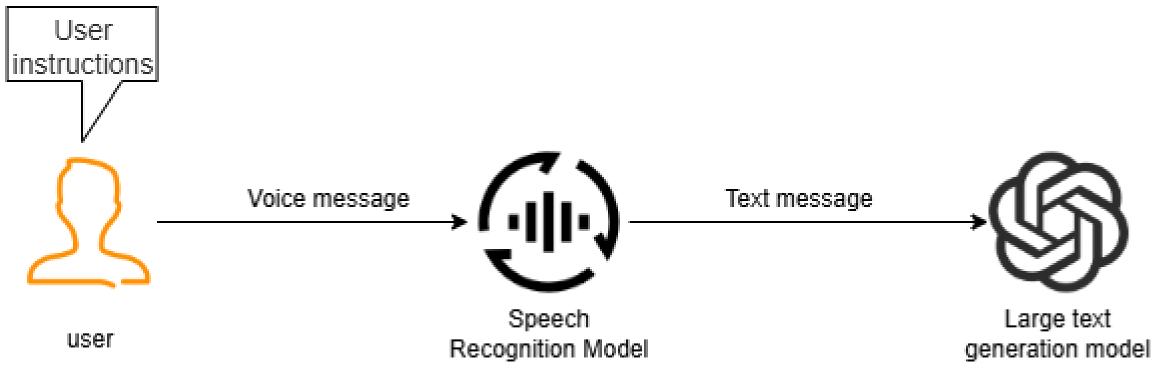
Function Name	Parameters	Function Implementation	Call Instructions
Car_servo_nod()	None	Executes the preset nod gesture	Nod command
Car_servo_sayno()	None	Executes the preset head shake gesture	how long does the head shake command
Car_RGB_Control(R,G,B)	R: represents red, G: green, B: blue These values range from 0 to 255, with higher values resulting in darker colors.	Controls the color of the taillight strip	Controls the color of the light strip
Close_RGB()	None	Turns off the lights	Turns off the lights command
Car_Music_API(strname,strmusic)	strname: The artist's name strmusic: The song title	Plays music online (may not be available due to copyright issues)	Plays music command
Get_dis_obstacle()	None	Measures the distance to an obstacle ahead	Measures the distance to an obstacle
car_avoid_api(fardis,Stopflag)	fardis is the stopping distance from the object, in millimeters; Stopflag: Whether to stop after reaching the fardis distance. Stopflag = 0 means no stopping.	Execute obstacle avoidance	Obstacle avoidance command
Track_Face_Follow()	None	Execute face tracking	Face following command

Function Name	Parameters	Function Implementation	Call Instructions
Track_line(colorline)	colorline: Track line color	Execute line tracking action	Line tracking command
Track_Food(strname)	strname: Object to be tracked	Execute object tracking action	Object tracking command
time.sleep(time)	time: Rest time	Rest wait	Rest wait command

5. Application of the Voice Model in the System

5.1 Voice Recognition

Since the decision-layer and execution-layer models are text generation and visual multimodal models, respectively, they cannot directly receive user voice information. Therefore, a voice recognition model is required to convert user voice commands into text before passing it to the AI model.



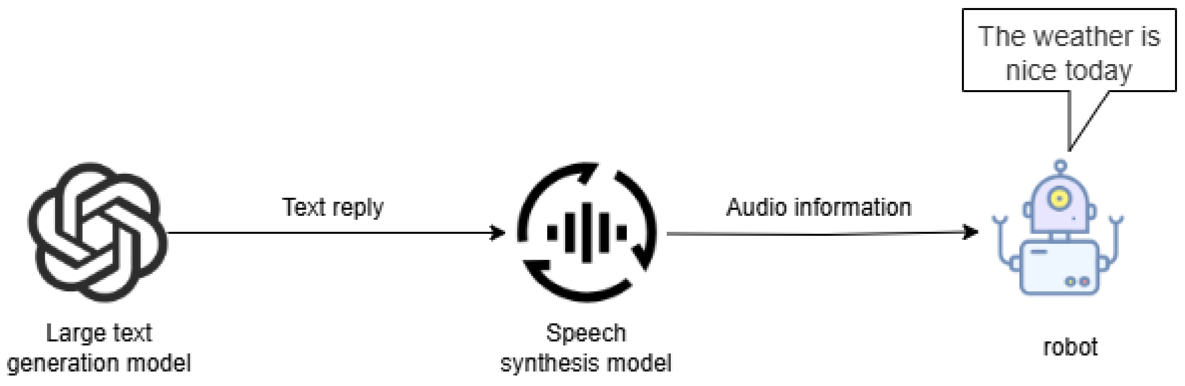
5.2 Voice Activity Detection (VAD)

VAD (Voice Activity Detection) is a technology that automatically distinguishes speech segments from non-speech segments (such as silence and noise) in speech signals. It locates the start and end points of speech in the audio stream and filters out invalid background sounds.

In subsequent practical courses, when the user wakes up the robot using the wake-up word "Hello Xiaoya," VAD voice activity detection will begin. The system automatically detects the user's speaking duration, saves valid audio segments as WAV audio files, and then converts the audio files into text for the voice recognition model.

5.3 Speech Synthesis

The large model converts user-generated text responses into audio via the speech synthesis model, which is then played back through the hardware speakers.


The weather is nice today
Text reply
Audio information
Large text generation model
Speech synthesis model
robot