Human-Robot Interaction Development Log

06/03

- Configured Scoop, FFmpeg, Visual C++ Build Tools, CUDA, Git, Npm/Node, Ollama.
- Experimented with RealTimeSTT and RealTimeTTS (https://github.com/KoljaB/RealtimeSTT, https://github.com/KoljaB/RealtimeTTS).
- Switched from FasterWhisper to deploying OpenAI-Whisper due to HuggingFace network issues.
- Still need to update NVIDIA drivers and fix Pytorch issues.

06/04

- Fixed the issue where Pytorch could not detect CUDA and NVIDIA drivers.
    - https://forums.developer.nvidia.com/t/cuda-not-detected-in-pytorch-unable-to-use-gpu-for-yolo-training/316244
    - Solution: Downgraded CUDA version, cleared pip cache, and re-downloaded stable versions of cu126 torch, torchvision, torchaudio.
- Attempted to use KokoroTTS and OmniVoice (https://github.com/nazdridoy/kokoro-tts, https://github.com/k2-fsa/OmniVoice/).
    - Could not run successfully due to network issues since assets are still stored on Huggingface.
- Successfully deployed a Chinese female voice TTS based on the SystemTTS Engine.
- Implemented Agent deployment using CrewAI (https://github.com/crewAIInc/crewAI).
- Integrated the company's LLM SDK and CrewAI for deploying the Agent (https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/llm-sdk.py).
- Initially developed a network scripting tool for the LLM model.
    - Could not use MCP (https://api.you.com) due to network issues.
    - Switched to using gemma3-tools:12b-ft and CrewAI's tools function (https://ollama.com/orieg/gemma3-tools).
    - Specific functions still need debugging.
- Compared the Chinese comprehension abilities of qwen2.5:7b and gemma3:4b.
    - qwen2.5:7b has stronger comprehension; switched to using qwen as the agent.
- Tested the new LLM system prompts using simulated data.
    - A total of 100 test sets, with an accuracy rate of around 70%.
    - Main program: https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/agent-testing.py
    - Simple test results: https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/test.txt
    - Hard test results: https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/hard-test.txt
    - Simulated data and functions can be further refined.
- Configured Ubuntu using Virtual Box.

06/05

- Basic flow:

    - ~~Used Whisper for STT (can use ASR API later to speed up recognition).~~
    - Used RealTimeSTT based on FasterWhisper for speech recognition.
    - Converted the location list and price list into strings.
    - Passed information to the LLM Agent.
    - Agent processed the information and output JSON.
    - JSON includes information:
        - User intent classification (intent): CHAT | INSPECTION | PRODUCT_QUERY | DISPLAY_CHECK | NAVIGATION | TASK_EXECUTION.
        - Whether command execution is required (requires_action): true/false.
        - Command location recognition (target_location): null or ["A", "B", "C", "D"].
        - Command action recognition (commands): null or ["MoveTo:X", "Scan", "PickUp"].
        - User response (response): String replying to the user.
- Test results:
    - JSON outputs from 103 test cases all met the requirements.
    - Test code: https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/interpreter.py
    - Test output: https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/checks.txt
- Points to note:
    - Available commands must be integrated into the Agent Prompt, otherwise the Agent has a high probability of outputting fake commands.
    - Similarly, lowering the LLM Agent Temperature can also prevent outputting fake commands.
    - Limited context window may cause the Agent to forget longer location lists/item lists.
    - ~~Theoretically, FasterWhisper inference speed will be significantly faster than Whisper, but it cannot be deployed temporarily due to HuggingFace link issues.~~
    - Successfully deployed RealTimeSTT based on FasterWhisper through a mirror site.
- Subsequent work:
    - ☑ Add historical chat record/context function.
    - ☑ Verify more simulated commands.
    - ☑ Dock with the robot hardware company to develop command interfaces.
- Demo:
    - Used FastAPI for backend deployment (https://github.com/fastapi/fastapi).
    - Added wake-up function based on openWakeWord (https://github.com/dscripka/openWakeWord).
    - Deployed real-time ASR function using RealTimeSTT.
    - Restart the interaction loop after the LLM Agent provides an answer.
    - Added memory function based on a rolling window.
    - ~~Initial website backend: https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/demo-website.py~~
    - ~~Initial website frontend: https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/index.html~~
    - Website backend: https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/demo-final.py
    - Website frontend: https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/index-final.html

06/08

- Experimented with the smaller qwen2.5:3b model; confirmed qwen2.5:7b as the smallest model that retains functionality.
- Docked with the robotics company, confirmed HTTP interface functions.
- Simulated HTTP interface docking: https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/http-test.py
- Replaced qwen2.5:7b with qwen3:8b to achieve better command comprehension.
- Switched to using FunASR to speed up STT (https://modelscope.github.io/FunASR/zh/).
- Started integrating FunASR into the RealtimeSTT structure.

06/09

- Successfully integrated FunASR into RealtimeSTT and consolidated most functions.
- Attempted to deploy RealtimeSTT's native FastAPI service.
- Due to the repository not being updated, the Demo is still using Faster-Whisper at the moment.
- Created a Demo repository and verified the deployment process.
- Added text input functionality to the web version Demo.
- https://git.aimall-tech.com/Research/Projects/lm/robot_interaction/-/tree/dev-1.0

06/10

- Tested the performance of the local LLM Agent when facing multiple commands.
    - The local model exhibited hallucinations or crashed during reasoning.
- Attempted to solve the local Agent's problems.
    - The same behavior occurred after switching to qwen3.5:9b.
    - The same behavior persisted even after enabling thinking mode.
    - If max_iter is limited, the Agent will not adhere to the output structure.
- Switched to using the company SDK as the Agent LLM.
    - Completely integrated the company SDK with CrewAI.
    - The Agent based on the company SDK can complete complex tasks normally.
    - Updated Demo 2.0 using the company SDK.
    - https://git.aimall-tech.com/Research/Projects/lm/robot_interaction/-/tree/dev-2.0
- Added relevant information to the website Demo.

06/11

- Docked with the robotics company's interfaces.
- Helped debug port connection issues.
- Initially verified getting port information.
- Tested mobile terminal control and mapping for the robot.
- Established map and points for the Demo.
- ~~Attempted to debug robot movement commands.~~
    - ~~Known issues:~~
        - ~~The mobile terminal lacks the move-to-point function due to porting issues.~~
        - ~~Random path is an experimental feature and cannot be used on this robot.~~
        - ~~The robot's position must be calibrated before running the program.~~
        - ~~Main program running failure.~~
- Update: Robot movement ports have been successfully deployed.
- Integrated Ntfy into the main program to bypass company computer network connection issues.

- Updated architecture:
  - The company computer is responsible for deploying the LLM Agent.
  - Uses Ntfy to send results after getting the output.
  - Uses another computer connected to the robot network to receive commands.
  - The second computer compiles the commands.
  - Finally, the compiled commands are sent by the second computer to the robot's HTTP interface.
  - Fallback: Use a long network cable to connect the computer and the robot (program can run offline).
- Successfully adjusted the output to the correct interface format.

06/12

- Tested obstacle avoidance function.
  - The robot attempts straight-line movement and gets stuck after being blocked by an obstacle.
  - Requires manual movement of the robot or using the mobile software after getting stuck.
  - Issue resolved after changing the minimum distance for movement settings.
- Implemented multi-command functionality.
  - The robot sends a series of commands via a List.
  - The code executes each command in a loop.
  - Waits for robot status change after sending a command.
  - Checks task completion status after the state changes.
  - Executes the next task after the current one is completed.
- Next steps plan:
  - ☑ Test the embedded system.
  - ☑ Deploy code to the embedded system.
  - ☑ Fully implement the embedded system human-robot interaction flow.
- Target architecture:
  - Embedded system captures audio.
  - Audio is sent to the backend server via Websocket.
  - Server performs STT.
  - STT text is sent by the backend server to the Agent using the SDK.
  - Agent returns text to the server.
  - Server returns text to the embedded system.
  - Embedded system recognizes the request.
  - Embedded system sends commands to the robot.
- Bill of Materials:
  - ESP32-S3-DevKitC-1
  - ZTS6672 Microphone
  - MAX98357A
  - Jumper wires
  - 400-point Breadboard * 2

06/15

- Set up an MCP server based on the robot API endpoint using FastMCP.

- Set up MCP server calls using AstrBot.
    - Since the company SDK still does not support MCP, the Demo will use the standard interface flow.
- Tested robot network settings.
    - The robot's local network can send out requests and process requests.
    - Embedded devices can normally execute requests in the program after connecting to the robot's Wifi.
- Added TTS output to the Demo.
    - Tested several TTS engines:
        - KokoroTTS (slow speed)
        - ChatTTS (compilation error)
        - PocketTTS (fast speed, only supports English synthesis)
    - When testing embedded devices, audio files can be sent via Websockets to be played by the embedded device.
- Prepared for embedded device deployment.
    - ESP32 wake word setup (https://docs.espressif.com/projects/esp-sr/en/latest/esp32s3/wake_word_engine/README.html).
    - Attempted to test code on Wokwi.
        - Code could not compile correctly.
    - Attempted to configure PlatformIO for code testing.
        - Both VSCode and CLion could not be configured correctly (CLion compatibility is still a beta version).
    - Configured Arduino IDE.
    - Determined ESP32 setup functions:
        - Use requests to ask for robot information.
        - Collect audio data through the microphone.
        - Transmit robot and microphone data to the server via Websocket.
        - Send commands to the robot via requests after receiving the server's return.

06/16

- Implemented network acceleration function using Watt Toolkit.
- Achieved backend deployment on a public URL by setting up Ngrok.
    - Dynamically selects ws and wss interfaces for remote websocket connection.
- Simplified STT and Agent settings (https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/backend.py).
- Successfully implemented client interaction flow.
    - Client sends map point information to the backend.
    - Client streams audio output.
    - Backend begins identifying audio after recognizing the wakeword.
    - Recognized text and information are sent to the LLM Agent.
    - Backend returns the LLM Agent output.
- Tested experimental multi-client docking.
    - The CrewAI architecture based on LLM API can process multiple dialogues through multiple Agents.
    - Uses client_id to record information from different clients.

    - Voice recognition will require more testing later (might need more lightweight voice recognition functions).

06/17

- Switched to using FunASR as the STT backend to process audio (near real-time recognition speed).
- Added preservation of map information for different clients.
- Added preservation of item prices for different clients.
- Added preservation of chat records/context for different clients.
- https://git.aimall-tech.com/shengzhen.wang/human-robot-interaction/-/blob/main/multi_client.py
- ESP32 Architecture:
    - Requests data from the robot after connecting to the server.
    - Sends robot data to the server.
    - The main program uses the microphone to continuously send audio data to the server in a loop.
    - Waits for the server's return within the loop.
    - Sends commands to the robot after the server returns.
- Architecture Choice:
    - Since using the Websockets library is primarily based on the Arduino IDE.
    - If Arduino IDE efficiency cannot meet usage requirements, switch to using ESP-IDF.
- Set up Arduino IDE environment.
- Prepared ESP-IDF compilation environment.

06/22

- Uploaded the test program using a USB cable, but it failed to run successfully.
    - Attempted uploading directly using the bin file.
    - Attempted different interface settings (DIO and DOUT).
    - Attempted different compilation settings.
    - Attempted using a merged bin.
    - Attempted changing PSRAM settings.
    - Attempted changing Partition settings.
- ~~Subsequent programs need to be compiled from source code.~~
- Successfully compiled Arduino IDE code.
- Successfully connected to the robot Wifi.
- Successfully obtained robot data using the HTTP interface.

06/23

- Implemented sending GET and POST requests on the ESP32, as well as calling the microphone and transferring data via Websockets.
- Avoided high RAM usage caused by excessively long JSONs.
- Resolved HTTPS and Websocket connection issues (used Websockets instead of Arduino Websockets).
- Correctly configured audio tone and speed.
- Attempted to resolve Websocket audio transmission lag issues.
    - Highly likely caused by the ESP32's own limited resources.

    - VAD can be used later to reduce unnecessary data transmission.
- Websockets connection remains unstable.
    - May be caused by resource contention.
    - Requires further testing later.

06/24

- Successfully resolved Websocket delay/instability issues.
    - Separated recording and data sending tasks via FreeRTOS.
    - Adjusted WiFi module and Websocket settings.
    - Added a volume threshold to reduce unnecessary data transmission.
- Implemented cyclic querying of robot status using FreeRTOS.
    - Begins executing query after sending a command.
    - Executes the next command after confirming the previous command has ended.
    - Clears memory after each query to ensure sufficient memory space.
- Fully tested multi-command execution flow.
    - Switched to using the Fun-ASR-MLT-Nano speech recognition model to support more languages.
    - Used Deepseek-V4-Pro to support multi-language output.

06/25

- Attempted to add screen display functionality.
    - Screen and microphone use the same GPIO pins simultaneously.
    - Display function needs to use Core 1 separately to execute tasks.
    - The burden on the ESP32 may be too large; requires more testing before adding later.
- Preparing for backend TTS functionality implementation.
    - Open-source TTS supporting simultaneous multi-language use is limited.
    - There is still a series of issues with TTS voice transmission and playback.
        - Unstable Websocket transmission.
        - ESP32 needs to simultaneously record audio and play TTS.
- Initially tested the LLM Agent's networking functionality.
    - Agent analyzes the request and outputs search keywords.
    - Uses Firecrawl to call the search API and retrieve keywords.
    - Another Agent uses the returned results to reply to the user.
- Initially tested the interaction process between the LLM Agent and the database.
    - Uses DuckDB to process CSV files.
    - Provides the Agent with data types and data samples.
    - Agent outputs SQL commands.
    - DuckDB executes the SQL commands and returns results.
- Integrated both the networking Agent and database Agent into the backend.
- Also added logic to judge the Agent and decide whether it is necessary to call the newly added Agents.

06/26

- Split the main program into smaller modules to increase subsequent adaptability.
- server.py

    - Backend built using FastAPI.
    - Responsible for providing HTTP API interfaces and voice data websockets.
- models.py
    - Stores all custom data types.
    - System State: Stores map data, database names, historical records, and connected clients.
    - MapPointsInput: API interface model for map data.
    - DatabaseInput: API interface model for database names.
    - TextInputRequest: API interface model for manual input requests.
- sdk_wrapper.py
    - Company SDK interface developed for crewai.
    - Defaults to using Deepseek V4 Pro.
    - Because the company SDK's ChatV2Response cannot use temperature, the current main function is to change the model used.
- build_agent.py
    - Functions responsible for creating Agents.
    - build_isolated_crew: Creates the final responding Agent.
    - build_intent_crew: Creates the Agent responsible for identifying whether network and database usage is needed.
    - build_research_crew: Creates the search Agent.
    - build_database_crew: Creates the database Agent.
- crew_pipeline.py
    - Master function responsible for calling Agents.
    - First calls the Intent Agent to decide whether to call other Agents.
    - Executes other Agent pipelines based on the Intent Agent's results.
    - Retrieves map data, search data, database data, and chat records.
    - Uses all data to call the Response Agent.
    - Sends JSON results.
- audio_processing.py
    - Functions responsible for processing audio.
    - route_audio_async: Responsible for calling run_crew_pipeline based on recognized data.
    - load_wakeword: Responsible for initializing the wake word engine.
    - load_FunASR_engine: Responsible for initializing the FunASR speech recognition engine.
    - native_audio_processing_engine: The main engine that calls the wake word and speech recognition engines.
- https://git.aimall-tech.com/Research/Projects/lm/robot_interaction/-/tree/dev-4.0
- Confirmed standard for the robot image transmission interface.

06/29

- Tested the interface for sending images.
- Added the photo request interface to the ESP32 code.
- Initial compilation of demonstration materials.
- Human-Robot Interaction
- Tested TTS functionality implementation.
    - Tested GPT-SOVITS.
    - Currently, GPT-SOVITS mainly uses a web UI with no direct software library available.
    - Prepared F5TTS environment.

06/30

- Tested F5TTS, Chatterbox, VoxCPM, StyleTTS.
  - FFMPEG version incompatible.
  - Torch version incompatible.
  - Could not successfully generate voice.
  - Tokenizer could not build wheel.
- Successfully synthesized voice using GenieTTS.
  - Currently only supports Chinese, Japanese, and English.
  - Cannot synthesize multiple languages simultaneously.
  - Other minority languages can be synthesized using simple ONNX models.
- Set up a TTS server using GenieTTS.
  - Can send TTS requests using the HTTP port.
- Successfully tested the photo request endpoint.
- Integrated image requests into the ESP32 program.
- Subsequent tasks:
  - [ ] Integrate backend recognition return results.
  - [x] Deploy/integrate TTS service.
  - [x] Complete interaction loop.

07/01

- Checked host audio hardware settings.
  - Adjusted audio settings using Alsamixer.
- Prepared TTS server Docker.
  - Download speed was too slow; no improvement after changing mirrors.
  - Switched to testing direct compilation first.
- Attempted to run FastAPI server directly using Python.
  - Could not use the system's built-in Python3.8; switched to using Python3.9.
  - hf-xet setup failed; directly used a pre-compiled wheel.
  - The pre-compiled wheel did not exist.
- Used PiperTTS for synthesis.
  - English text synthesized correctly.
  - Chinese synthesis requires g2pw, which is more complex to set up and slower to process.
- Adopted new architecture.
  - Backend can use more complex models.
  - Will not increase computation tasks on the robot host.
- Specific plans:
  - [x] Integrate GenieTTS native FastAPI endpoints.
  - [x] Complete robot speaker installation.
  - [ ] ~~Dock and add robot host TTS endpoint.~~
  - [x] Migrate code to the robot host.

07/02

- Successfully integrated GenieTTS endpoint.

    - Because the backend must run FunASR and GenieTTS simultaneously, there is a large RAM requirement.
    - Cannot load GenieTTS after FunASR has been loaded.
    - Solution is to load the pre-set GenieTTS model first.
- ESP32 Websocket connection is still unstable.
    - Code can be transferred to the robot host after the robot connects the microphone.
- Installed robot speakers.
    - Audio output still needs configuration.
    - Alsamixer setup unresponsive.
    - Direct output using kernel commands was unsuccessful.
    - Used Sunlogin VNC for remote operation.
    - Successfully played audio through the speakers via pavucontrol and Ubuntu's built-in settings.

07/03

- Successfully set up microphone input.
- Completed code migration from ESP32 to the robot host.
- Integrated TTS synthesis and playback into the interaction loop.
- Perfected task completion check logic.
- https://git.aimall-tech.com/Research/Projects/lm/robot_interaction/-/tree/dev-5.0

07/06

- Set up a background daemon to complete robot host program deployment.
- Changed wake word sensitivity and LLM Agent prompt.
- Increased TTS request timeout limit.
- Updated the demo.

07/07

- Replaced GenieTTS with PiperTTS.
    - Inference speed increased significantly.
- Canceled the usage of the punc model.
    - Further shortened voice recognition speed.

07/08

- Added follow-up questioning functionality to the Agent Prompt.