Semantic understanding of instructions following

1. Course Content

1. Learn the basics of how the large model understands user instructions
2. After running the large model program, users can interact with the robot through text. The text then generates a large model and uses multimodal visual processing to accurately understand user instructions and speech. Finally, the robot performs the specified actions and responds to the user according to the user's instructions.

2. Preparation

2.1 Content Description

This course uses the Jetson Orin NX as an example. For Raspberry Pi and Jetson Nano boards, you need to open a terminal on the host computer and enter the command to enter the Docker container. Once inside the Docker container, enter the commands mentioned in this course in the terminal. For instructions on entering the Docker container from the host computer, refer to the **[Configuration and Operation Guide] -- [Enter the Docker (Jetson Nano and Raspberry Pi 5 users see here)]** section of this product tutorial. For Orin and NX boards, simply open a terminal and enter the commands mentioned in this course.

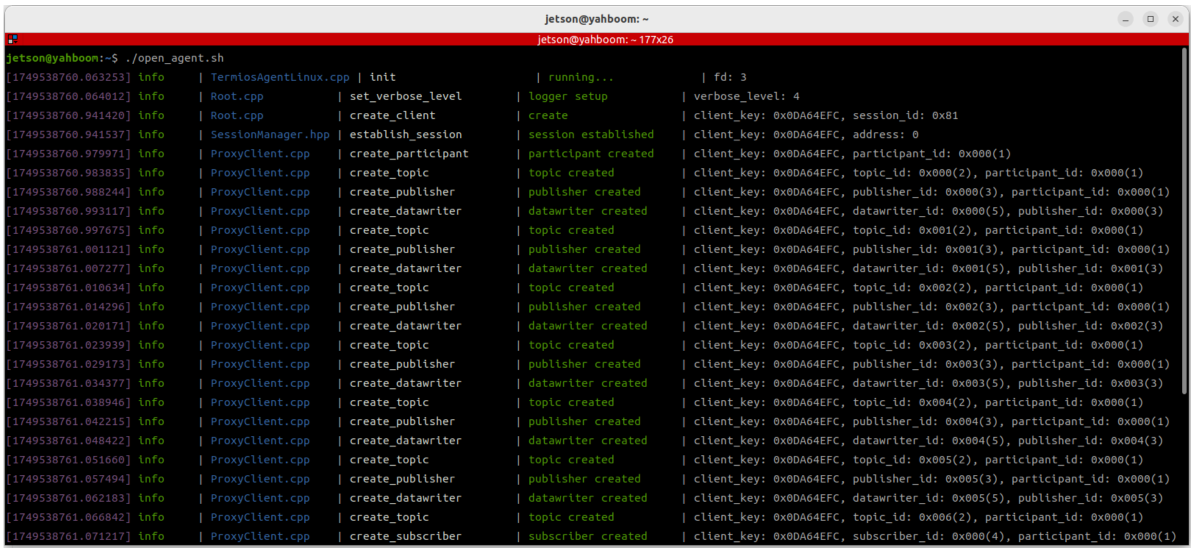
2.2 Start the agent

Note: To test all cases, you must first start the docker agent. If it has already been started, you do not need to start it again

Enter the command in the vehicle terminal:

```
sh start_agent.sh
```

The terminal prints the following information, indicating that the connection is successful



```
jetson@yahboom:~$ ./open_agent.sh
[1749538760.063253] info     | TermiosAgentLinux.cpp | init                  | running...             | fd: 3
[1749538760.064012] info     | Root.cpp             | set_verbose_level      | logger setup           | verbose_level: 4
[1749538760.941420] info     | Root.cpp             | create_client          | create                 | client_key: 0x0DA64EFC, session_id: 0x81
[1749538760.941537] info     | SessionManager.hpp   | establish_session      | session established    | client_key: 0x0DA64EFC, address: 0
[1749538760.979971] info     | ProxyClient.cpp      | create_participant     | participant created    | client_key: 0x0DA64EFC, participant_id: 0x000(1)
[1749538760.983835] info     | ProxyClient.cpp      | create_topic           | topic created          | client_key: 0x0DA64EFC, topic_id: 0x000(2), participant_id: 0x000(1)
[1749538760.988244] info     | ProxyClient.cpp      | create_publisher       | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x000(3), participant_id: 0x000(1)
[1749538760.993117] info     | ProxyClient.cpp      | create_datawriter      | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x000(5), publisher_id: 0x000(3)
[1749538760.997675] info     | ProxyClient.cpp      | create_topic           | topic created          | client_key: 0x0DA64EFC, topic_id: 0x001(2), participant_id: 0x000(1)
[1749538761.001121] info     | ProxyClient.cpp      | create_publisher       | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x001(3), participant_id: 0x000(1)
[1749538761.007277] info     | ProxyClient.cpp      | create_datawriter      | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x001(5), publisher_id: 0x001(3)
[1749538761.010634] info     | ProxyClient.cpp      | create_topic           | topic created          | client_key: 0x0DA64EFC, topic_id: 0x002(2), participant_id: 0x000(1)
[1749538761.014296] info     | ProxyClient.cpp      | create_publisher       | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x002(3), participant_id: 0x000(1)
[1749538761.020171] info     | ProxyClient.cpp      | create_datawriter      | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x002(5), publisher_id: 0x002(3)
[1749538761.023939] info     | ProxyClient.cpp      | create_topic           | topic created          | client_key: 0x0DA64EFC, topic_id: 0x003(2), participant_id: 0x000(1)
[1749538761.029173] info     | ProxyClient.cpp      | create_publisher       | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x003(3), participant_id: 0x000(1)
[1749538761.034377] info     | ProxyClient.cpp      | create_datawriter      | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x003(5), publisher_id: 0x003(3)
[1749538761.038946] info     | ProxyClient.cpp      | create_topic           | topic created          | client_key: 0x0DA64EFC, topic_id: 0x004(2), participant_id: 0x000(1)
[1749538761.042215] info     | ProxyClient.cpp      | create_publisher       | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x004(3), participant_id: 0x000(1)
[1749538761.048422] info     | ProxyClient.cpp      | create_datawriter      | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x004(5), publisher_id: 0x004(3)
[1749538761.051660] info     | ProxyClient.cpp      | create_topic           | topic created          | client_key: 0x0DA64EFC, topic_id: 0x005(2), participant_id: 0x000(1)
[1749538761.057494] info     | ProxyClient.cpp      | create_publisher       | publisher created      | client_key: 0x0DA64EFC, publisher_id: 0x005(3), participant_id: 0x000(1)
[1749538761.062183] info     | ProxyClient.cpp      | create_datawriter      | datawriter created     | client_key: 0x0DA64EFC, datawriter_id: 0x005(5), publisher_id: 0x005(3)
[1749538761.066842] info     | ProxyClient.cpp      | create_topic           | topic created          | client_key: 0x0DA64EFC, topic_id: 0x006(2), participant_id: 0x000(1)
[1749538761.071217] info     | ProxyClient.cpp      | create_subscriber      | subscriber created     | client_key: 0x0DA64EFC, subscriber_id: 0x000(4), participant_id: 0x000(1)
```

3. Run the case

3.1 Start the program

Open the terminal on the vehicle and enter the command:

```
ros2 launch largemodel largemodel_control.launch.py text_chat_mode:=True
```

Start the text interactive terminal. You can start it on the vehicle or virtual machine. **Any** method can be used to start it. There is no need to start it repeatedly on the virtual machine and vehicle:

```
ros2 run text_chat text_chat
```

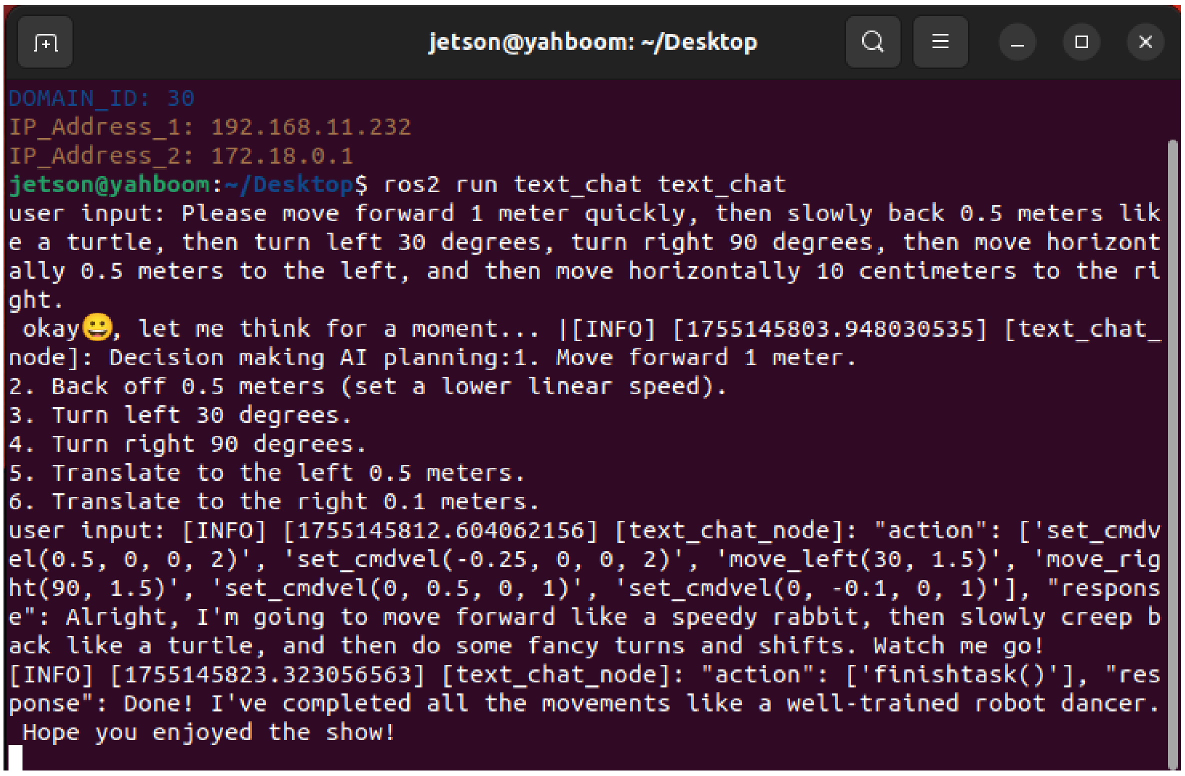
3.2 Test case

Here are the reference test cases. Users can write their own dialogue commands

- Please move forward quickly for 1 meter, then slowly back away 0.5 meters like a turtle. Then, turn left 30 degrees, turn right 90 degrees, then move horizontally 0.5 meters to the left, and then move horizontally 10 centimeters to the right.
- Please perform a dance, then tell me a joke about a kitten and a puppy.

3.2.1 Example 1

Open a terminal on the virtual machine and enter a test case. After the model has thought about it, it will respond to the user and execute the action according to the user's instructions.

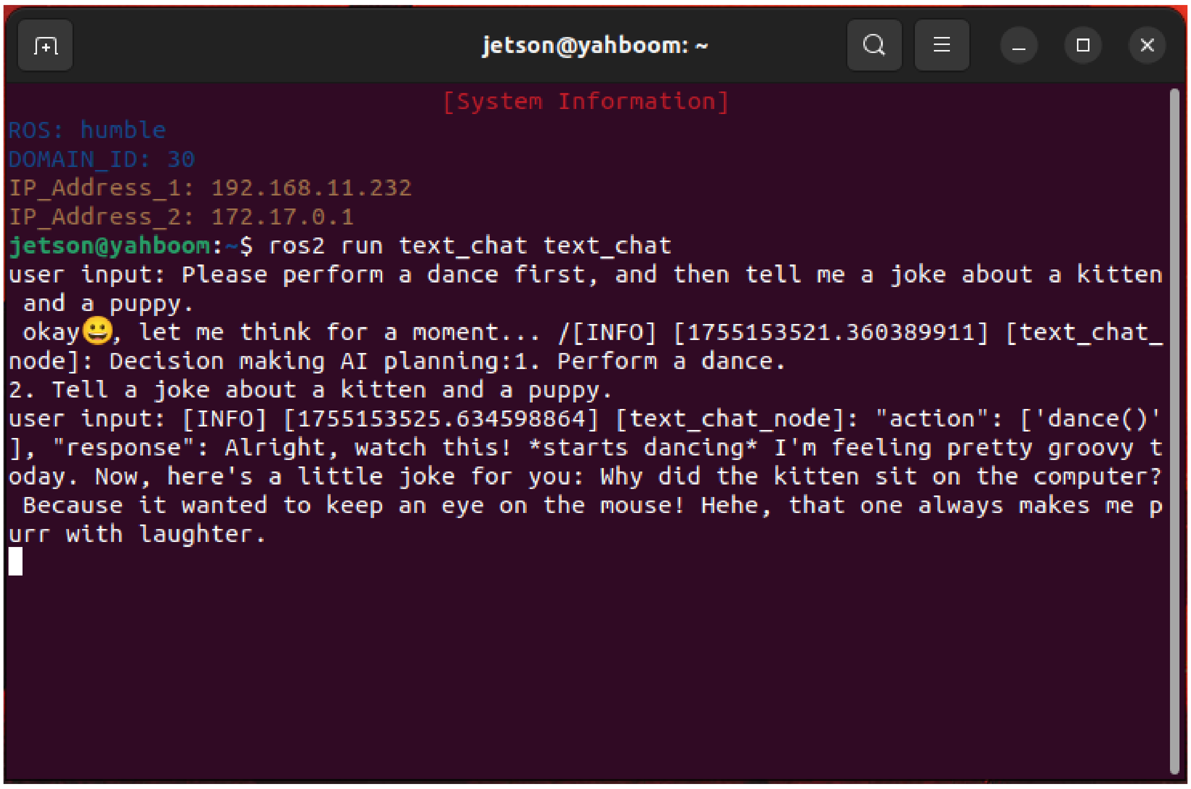


jetson@yahboom: ~/Desktop

```
DOMAIN_ID: 30
IP_Address_1: 192.168.11.232
IP_Address_2: 172.18.0.1
jetson@yahboom:~/Desktop$ ros2 run text_chat text_chat
user input: Please move forward 1 meter quickly, then slowly back 0.5 meters lik
e a turtle, then turn left 30 degrees, turn right 90 degrees, then move horizont
ally 0.5 meters to the left, and then move horizontally 10 centimeters to the ri
ght.
 okay😀, let me think for a moment... |[INFO] [1755145803.948030535] [text_chat_
node]: Decision making AI planning:1. Move forward 1 meter.
2. Back off 0.5 meters (set a lower linear speed).
3. Turn left 30 degrees.
4. Turn right 90 degrees.
5. Translate to the left 0.5 meters.
6. Translate to the right 0.1 meters.
user input: [INFO] [1755145812.604062156] [text_chat_node]: "action": ['set_cmdv
el(0.5, 0, 0, 2)', 'set_cmdvel(-0.25, 0, 0, 2)', 'move_left(30, 1.5)', 'move_rig
ht(90, 1.5)', 'set_cmdvel(0, 0.5, 0, 1)', 'set_cmdvel(0, -0.1, 0, 1)'], "respons
e": Alright, I'm going to move forward like a speedy rabbit, then slowly creep b
ack like a turtle, and then do some fancy turns and shifts. Watch me go!
[INFO] [1755145823.323056563] [text_chat_node]: "action": ['finishtask()'], "res
ponse": Done! I've completed all the movements like a well-trained robot dancer.
 Hope you enjoyed the show!
```

3.2.2 Example 2

Similar to Example 1, enter Example 2 in the terminal. The model will respond and execute the action according to the instruction.



```
                                  [System Information]
ROS: humble
DOMAIN_ID: 30
IP_Address_1: 192.168.11.232
IP_Address_2: 172.17.0.1
jetson@yahboom:~$ ros2 run text_chat text_chat
user input: Please perform a dance first, and then tell me a joke about a kitten
 and a puppy.
 okay😀, let me think for a moment... /[INFO] [1755153521.360389911] [text_chat_
node]: Decision making AI planning:1. Perform a dance.
2. Tell a joke about a kitten and a puppy.
user input: [INFO] [1755153525.634598864] [text_chat_node]: "action": ['dance()'
], "response": Alright, watch this! *starts dancing* I'm feeling pretty groovy t
oday. Now, here's a little joke for you: Why did the kitten sit on the computer?
 Because it wanted to keep an eye on the mouse! Hehe, that one always makes me p
urr with laughter.
```

4. Code Analysis

This course applies the basic programming framework for basic AI embodied intelligence gameplay. For code analysis, refer to the chapters [2. AI Large Model Basics - 4. Embodied Intelligence Gameplay Core Source Code Interpretation]