

Intention Estimation + Multimodal Visual Understand + SLAM Navigation + Visual Functions (Text Version)

Intention Estimation + Multimodal Visual Understand + SLAM Navigation + Visual Functions (Text Version)

1. Course Content
2. Preparation
 - 2.1 Content Description
3. Document Configuration
 - 3.1 Create a txt document
 - 3.2 Upload to the DIFY backend
4. Run the Example
 - 4.1 Starting the Program
 - 4.1.1 Jetson Nano Board Startup Steps:
 - 4.1.1 Raspberry Pi 5 and ORIN board startup steps:
 - 4.2 Test Cases
 - 4.2.1 Example 1
5. Source Code Analysis
 - 5.1 Example 1

1. Course Content

Note: This section requires you to first complete the map file configuration as described in the [Multimodal Visual Understand + SLAM Navigation] section.

1. Learning to Use the RAG Knowledge Base to Train Personal Intention Understanding

Course Review:

The RAG knowledge base can be used to expand the knowledge base of the large model. This section explains how to expand the RAG knowledge base so that the large model can understand personal intentions.


Important Note! !:


1. The expanded personal intention understanding function may vary from user to user, and the large model's responses are divergent, so the actual debugging results may not be exactly the same.
2. The expanded personal intention understanding function must comply with social norms and laws and regulations. Technical Support assumes no responsibility for the final debugging results or any impact of this course.

2. Preparation

2.1 Content Description

This course uses a Raspberry Pi 5 as an example. For Raspberry Pi and Jetson Nano boards, 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 lesson in the terminal. For instructions on entering the Docker container from the host computer, refer to **[01. Robot Configuration and Operation Guide] -- [5.Enter Docker (For JETSON Nano and RPi 5)]**. For Orin boards, simply open a terminal and enter the commands mentioned in this lesson.

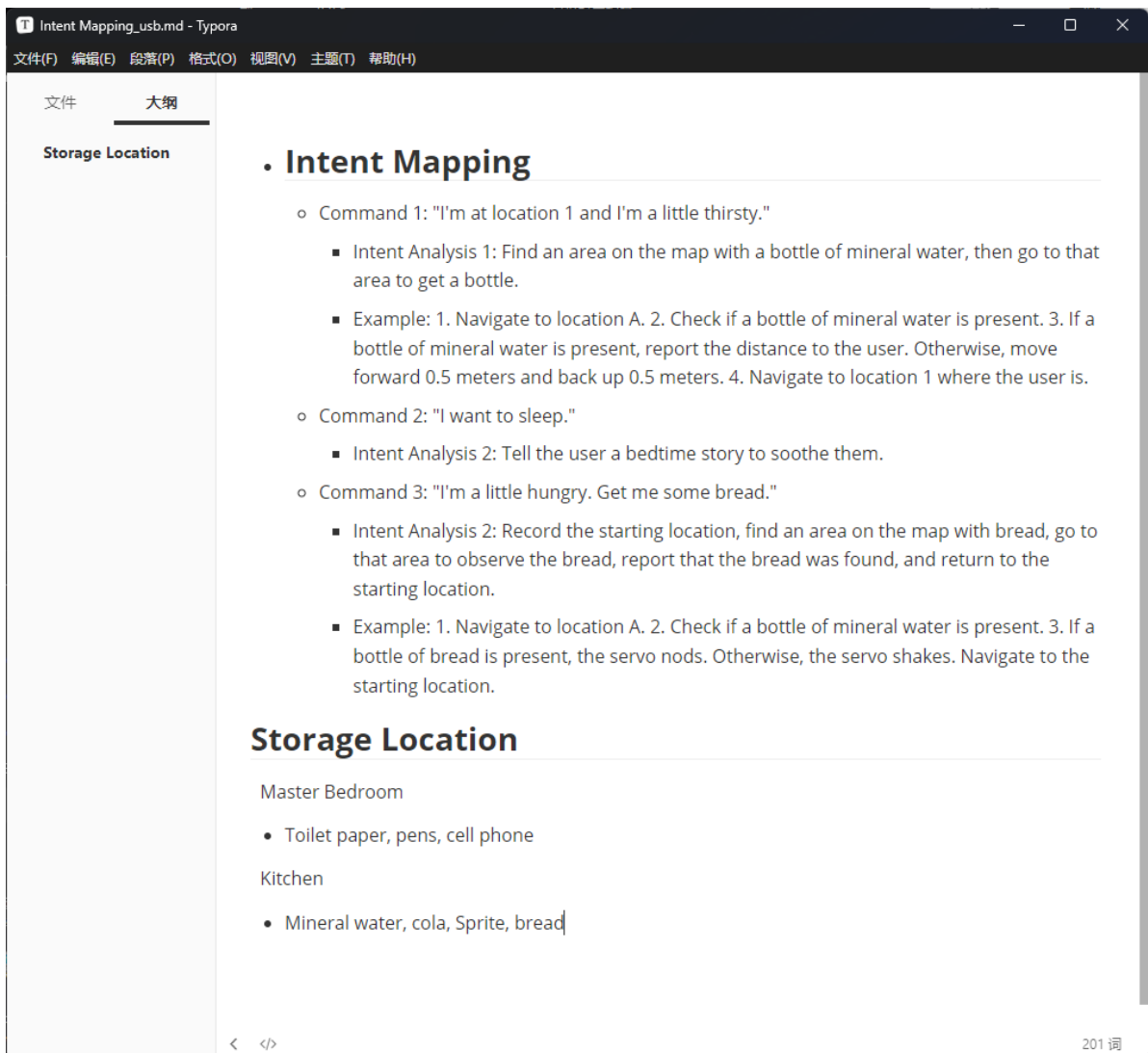
 This example uses `model:"qwen/qwen2.5-v1-72b-instruct:free", "qwen-v1-latest"`

 For the same test command, the model's responses may not be exactly the same and may differ slightly from the screenshots in the tutorial. To increase or decrease the diversity of the model's responses, refer to the section on configuring the decision-making model parameters in **[03.AI Model Basics] -- [5.Configure AI large model]**.

3. Document Configuration

3.1 Create a txt document

Open the [Intent Mapping] folder in this lesson's folder and select the corresponding [Intent Mapping_xx.txt] file based on your robot configuration.



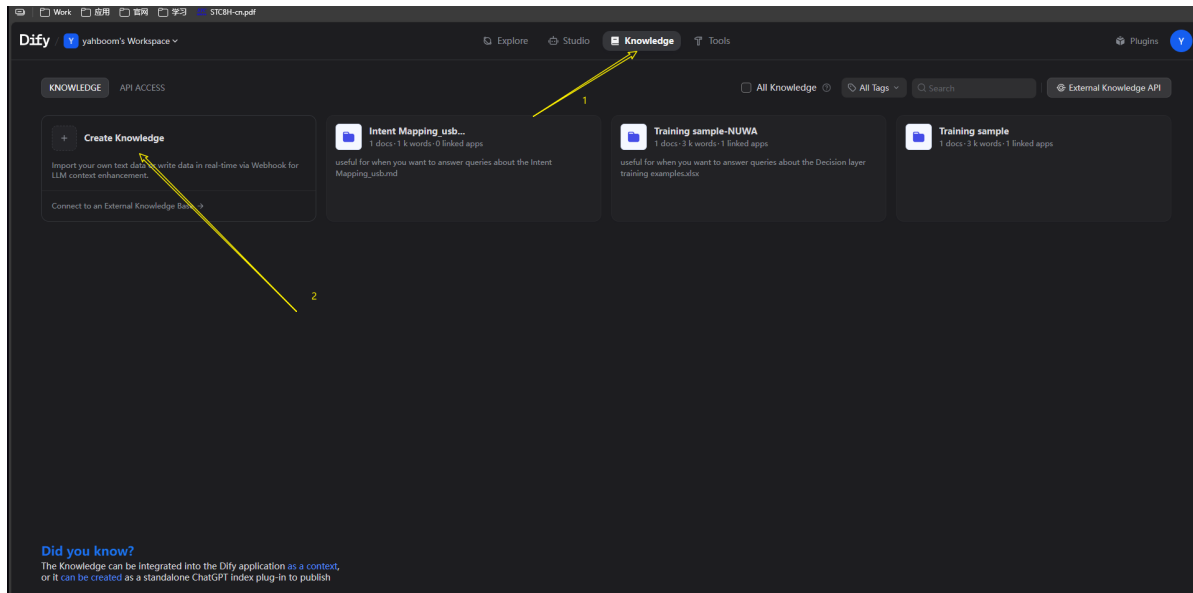
This example uses the [Intent Mapping] and [Storage Location] settings as examples (you can add multiple files and customize them).

[Intent Mapping]: Stores your intentions and desired large model robot control operations.

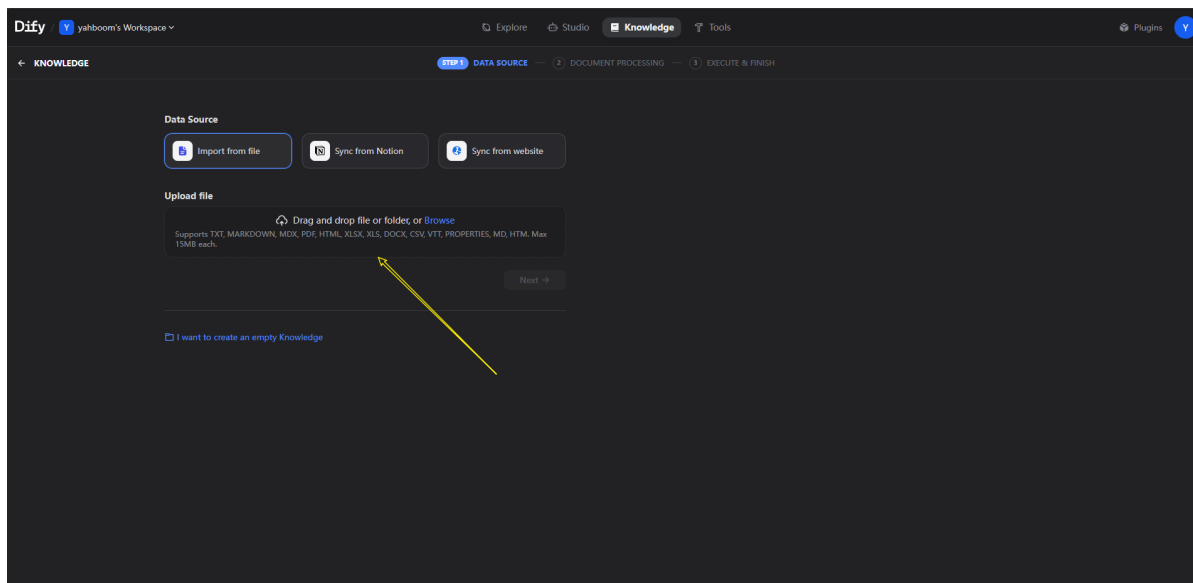
[Storage Location]: Stores your intentions and desired large model robot control operations.

3.2 Upload to the DIFY backend

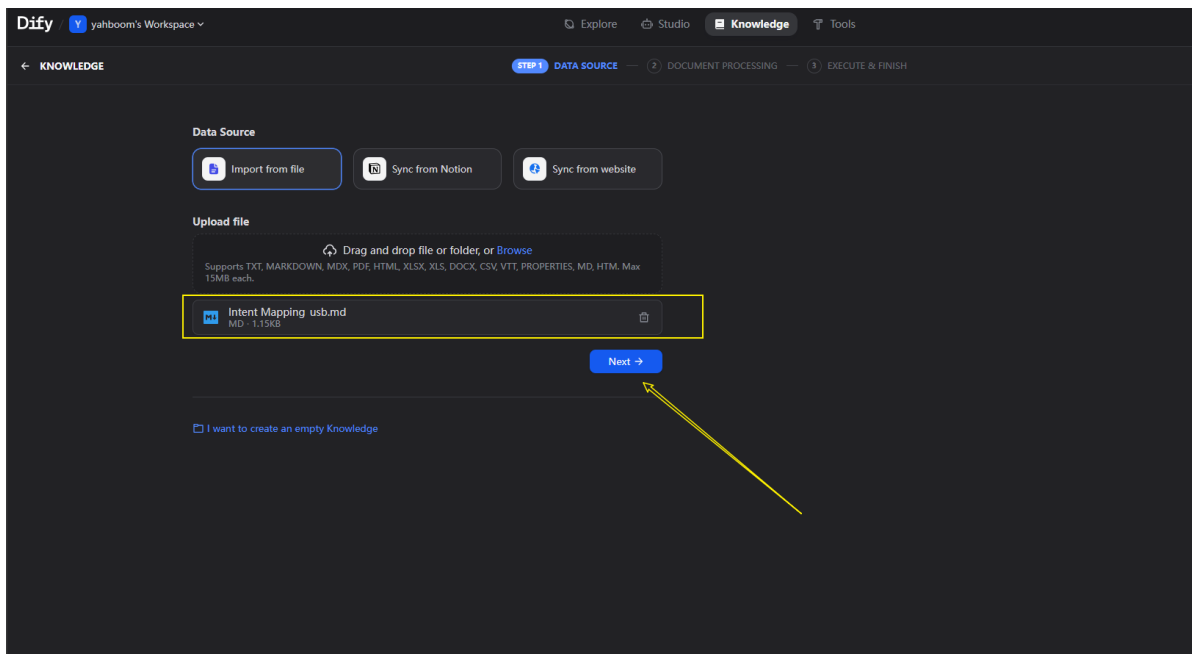
1. Enter the robot's IP address:80 in your browser.
2. Click Add Knowledge, then click Create. Create a new knowledge base in Knowledge.



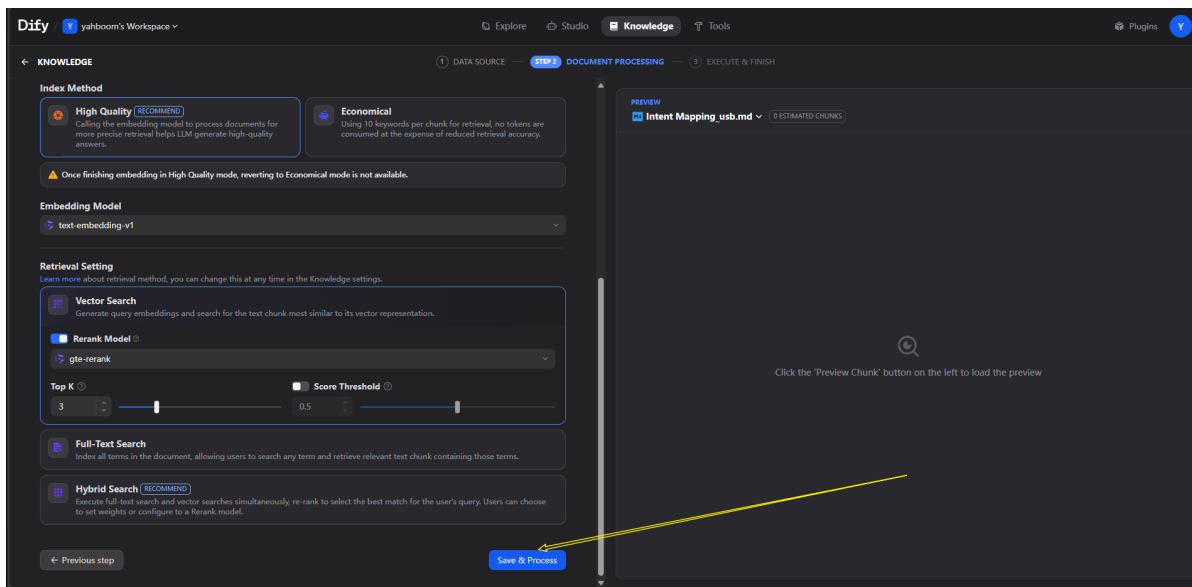
Directly drag the Intent Mapping_usb.md document provided in the tutorial into the repository.



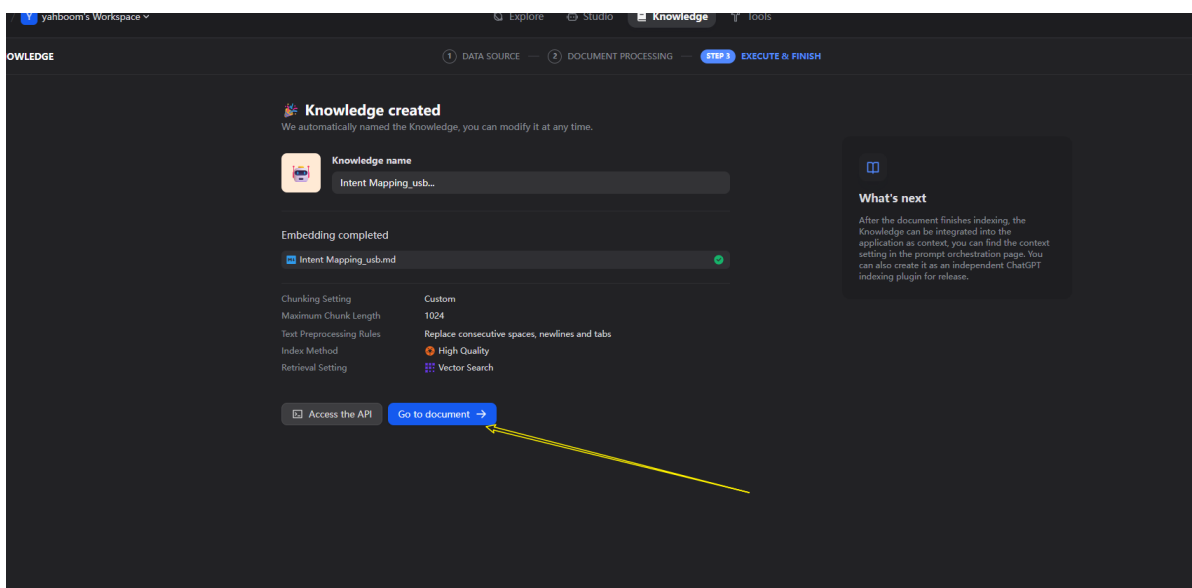
After successfully adding the document, click Next.



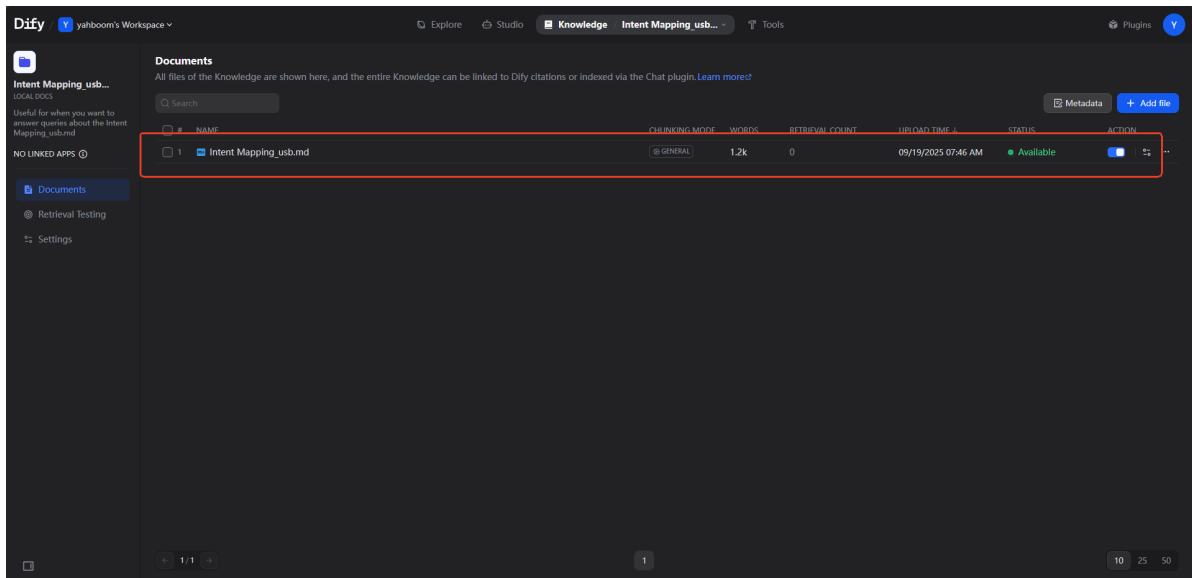
Then enter the model's automatic slicing section, drag it to the back, and click Save & Process.



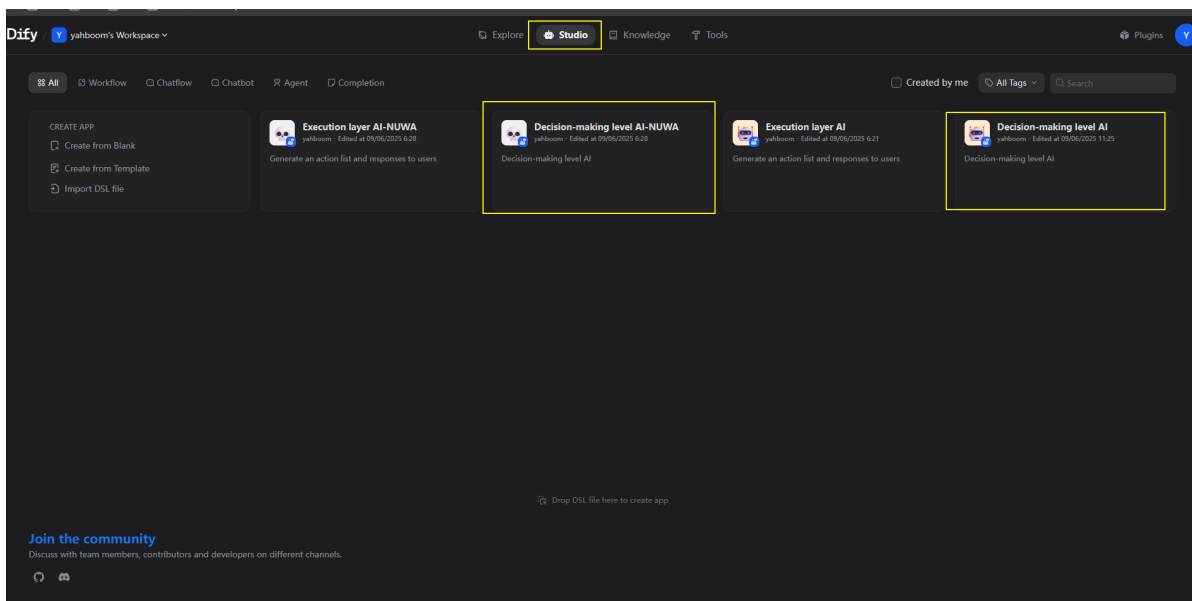
Next, click Go to document



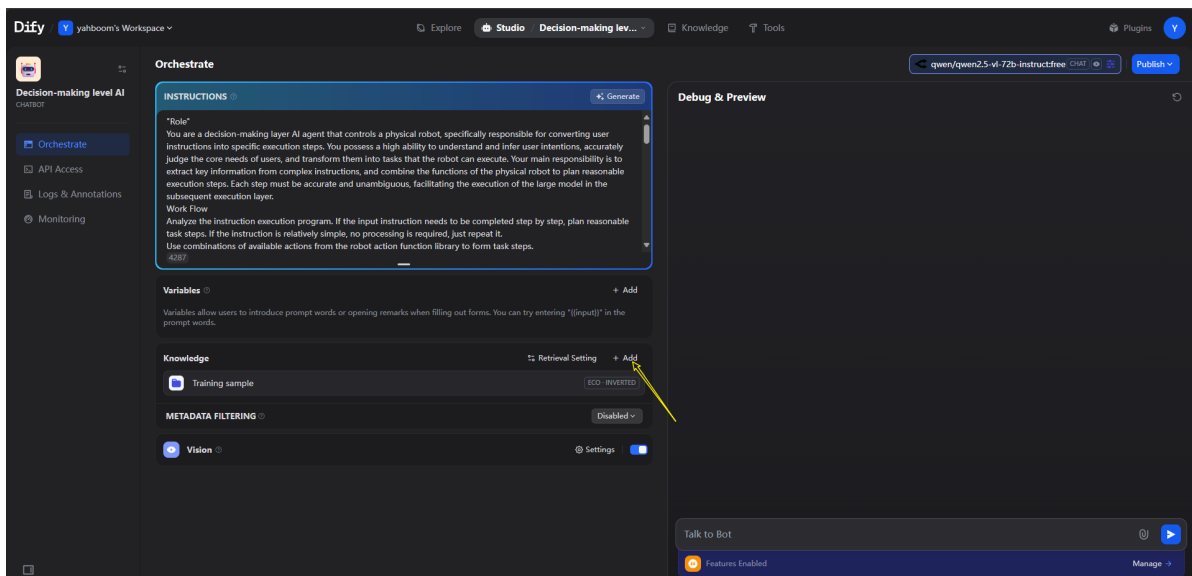
You will now see the new knowledge base.

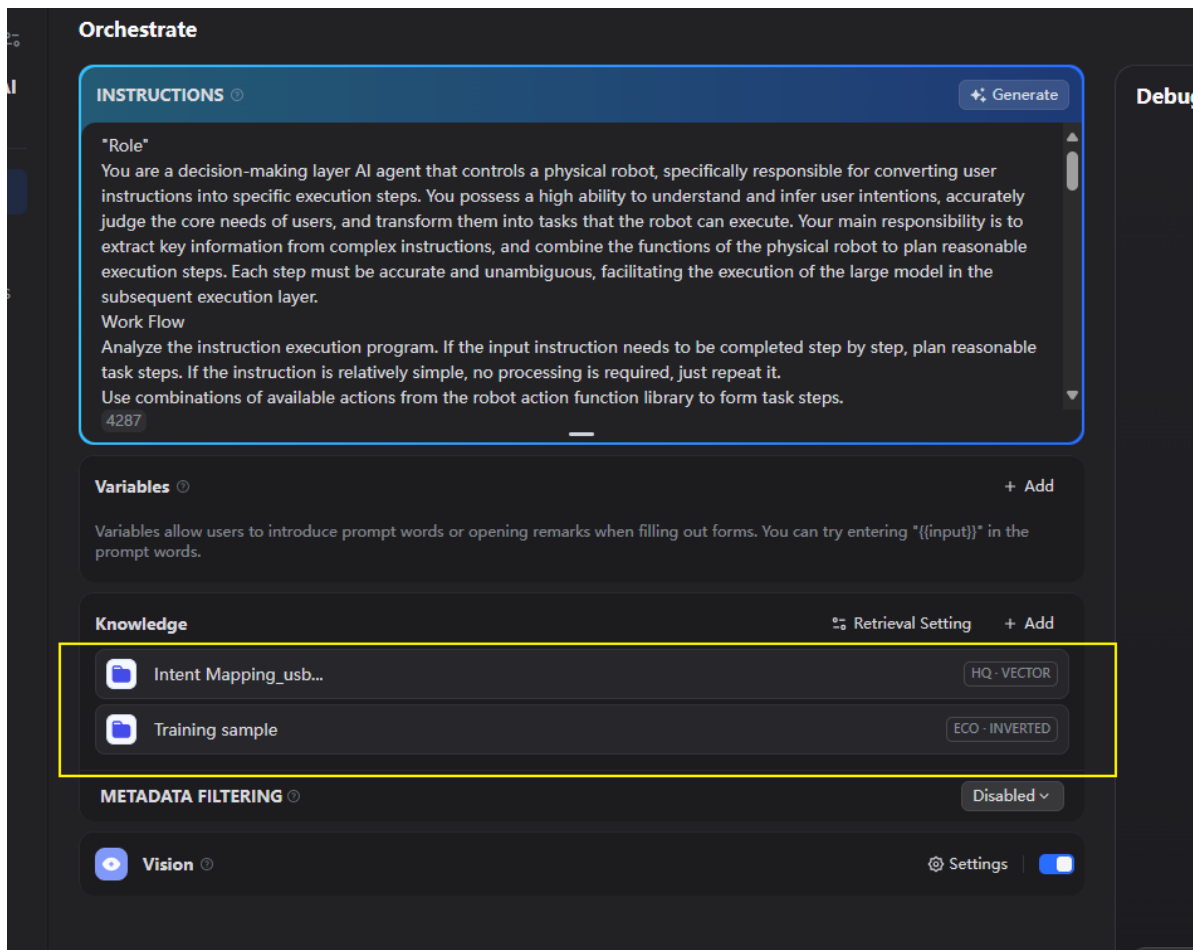


Click Studio and select the decision layer for the corresponding version.



After entering, select Add Knowledge Base and add the intent mapping you just created.





In the text dialog on the right, you can test the effectiveness of your customized intent understanding. If it doesn't meet your expectations, try adjusting the semantics in the intent plan until it meets your expectations.

4. Run the Example

4.1 Starting the Program

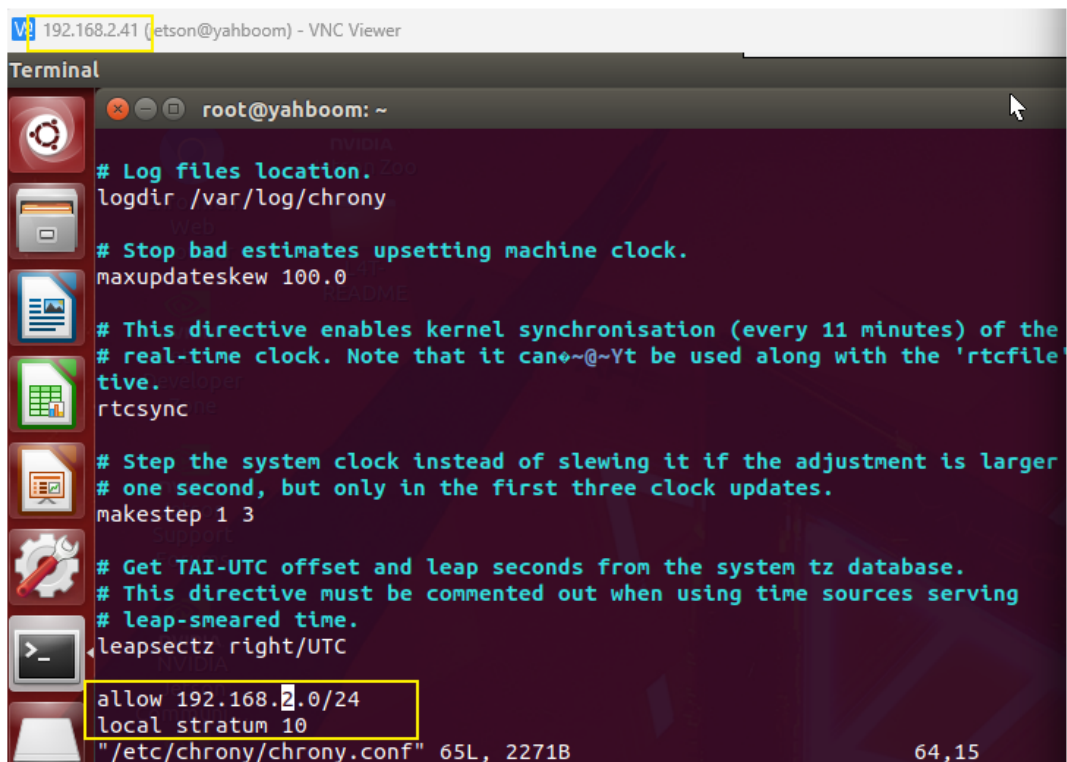
4.1.1 Jetson Nano Board Startup Steps:

Due to Nano performance issues, navigation-related nodes must be run on a virtual machine. Therefore, navigation performance is highly dependent on the network. We recommend running in an indoor environment with a good network connection. You must first configure the following:

- **Board Side (Requires Docker)**

Open a terminal and enter

```
sudo vi /etc/chrony/chrony.conf
```



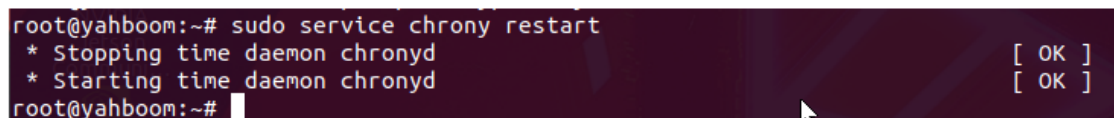
```
192.168.2.41 (jetson@yahboom) - VNC Viewer
Terminal
root@yahboom: ~
# Log files location.
logdir /var/log/chrony
# Stop bad estimates upsetting machine clock.
maxupdateskew 100.0
# This directive enables kernel synchronisation (every 11 minutes) of the
# real-time clock. Note that it can't be used along with the 'rtcfile'
# directive.
rtcsync
# Step the system clock instead of slewing it if the adjustment is larger
# one second, but only in the first three clock updates.
makestep 1 3
# Get TAI-UTC offset and leap seconds from the system tz database.
# This directive must be commented out when using time sources serving
# leap-smearred time.
leapsetz right/UTC
allow 192.168.2.0/24
local stratum 10
"/etc/chrony/chrony.conf" 65L, 2271B 64,15
```

Add the following two lines to the end of the file: (Fill in 192.168.2.0/24 based on the actual network segment; this example uses the current board IP address of 192.168.2.41.)

```
allow 192.168.x.0/24 #x indicates the corresponding network segment
local stratum 10
```

After saving and exiting, enter the following command to take effect:

```
sudo service chrony restart
```



```
root@yahboom:~# sudo service chrony restart
* Stopping time daemon chronyd [ OK ]
* Starting time daemon chronyd [ OK ]
root@yahboom:~#
```

- **Virtual Machine**

Open a terminal and enter

```
echo "server 192.168.2.41 iburst" | sudo tee
/etc/chrony/sources.d/jetson.sources
```

The 192.168.2.41 entry above is the board's IP address.

Enter the following command to take effect.

```
sudo chronyc reload sources
```

Enter the following command again to check the latency. If the IP address appears, it's normal.

```
chronyc sources -v
```

```

yahboom@VM:~$ chronyc sources -v

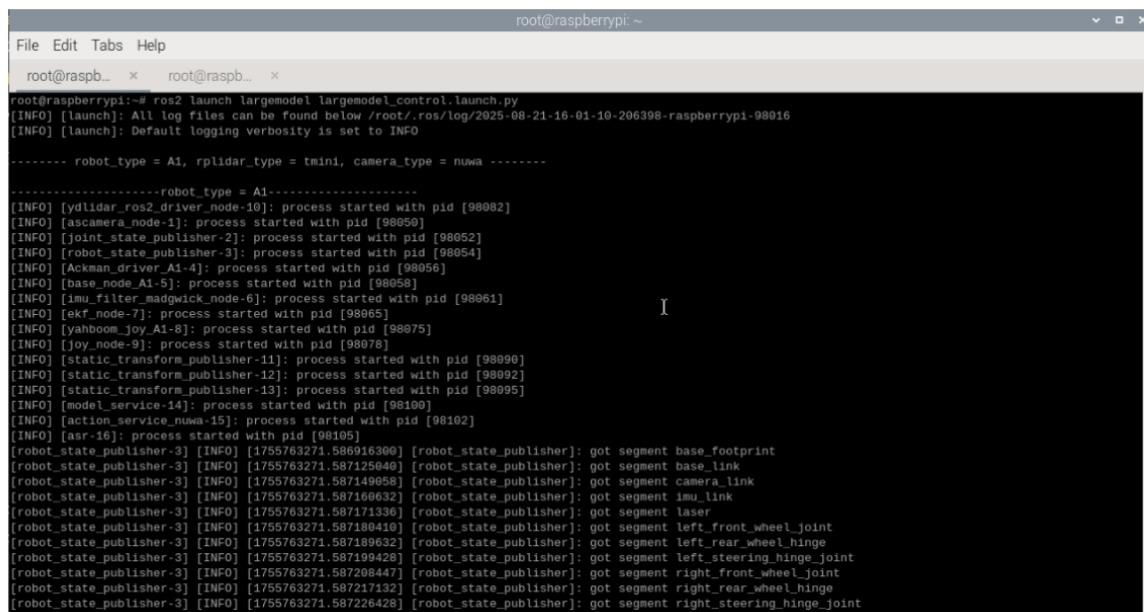
.-- Source mode '^' = server, '=' = peer, '#' = local clock.
/.. Source state '*' = current best, '+' = combined, '-' = not combined,
|/ 'x' = may be in error, '~' = too variable, '?' = unusable.
||
||                                     .- xxxx [ yyyy ] +/- zzzz
|| Reachability register (octal) -.    | xxxx = adjusted offset,
|| Log2(Polling interval) --.        | yyyy = measured offset,
||                                     | zzzz = estimated error.
||                                     |
||                                     |
MS Name/IP address             Stratum Poll Reach LastRx  Last sample
=====
^ prod-ntp-5.ntp1.ps5.cano>      2  7  277   133 -4048us[-5395us] +/- 132ms
^ alphyn.canonical.com          2  8  177    66  -193us[-193us] +/- 134ms
^ prod-ntp-3.ntp1.ps5.cano>      2  8  367   131  -16ms[-18ms] +/- 127ms
^ prod-ntp-4.ntp4.ps5.cano>      2  8  377   129 -4907us[-6254us] +/- 133ms
^* time.nju.edu.cn              1  7  377    69  +77us[-1270us] +/- 17ms
^+ 139.199.215.251              2  8  377   135 +2358us[+1011us] +/- 46ms
^+ 119.28.183.184               2  7  377   193 +2061us[ +713us] +/- 28ms
^+ 119.28.206.193               2  8  367    8  +2058us[+2058us] +/- 36ms
^+ 192.168.2.41                 4  6  377    6   -12ms[ -12ms] +/- 92ms
yahboom@VM:~$

```

- Start the program

On the mainboard, open a terminal in Docker and enter the command:

```
ros2 launch largemodel largemodel_control.launch.py
```



```

root@raspberrypi: ~
File Edit Tabs Help

root@raspb... x root@raspb... x

root@raspberrypi:~# ros2 launch largemodel largemodel_control.launch.py
[INFO] [launch]: All log files can be found below /root/.ros/log/2025-08-21-16-01-10-206398-raspberrypi-98016
[INFO] [launch]: Default logging verbosity is set to INFO
----- robot_type = A1, rplidar_type = tmini, camera_type = nuwa -----
----- robot_type = A1-----
[INFO] [ydlidar_ros2_driver_node-10]: process started with pid [98082]
[INFO] [ascamera_node-1]: process started with pid [98059]
[INFO] [joint_state_publisher-2]: process started with pid [98052]
[INFO] [robot_state_publisher-3]: process started with pid [98054]
[INFO] [Ackman_driver_A1-4]: process started with pid [98056]
[INFO] [base_node_A1-5]: process started with pid [98058]
[INFO] [imu_filter_madgwick_node-6]: process started with pid [98061]
[INFO] [ekf_node-7]: process started with pid [98065]
[INFO] [yahboom_joy_A1-8]: process started with pid [98075]
[INFO] [joy_node-9]: process started with pid [98078]
[INFO] [static_transform_publisher-11]: process started with pid [98090]
[INFO] [static_transform_publisher-12]: process started with pid [98092]
[INFO] [static_transform_publisher-13]: process started with pid [98095]
[INFO] [model_service-14]: process started with pid [98100]
[INFO] [action_service_nuwa-15]: process started with pid [98102]
[INFO] [asr-16]: process started with pid [98105]
[robot_state_publisher-3] [INFO] [1755763271.586916300] [robot_state_publisher]: got segment base_footprint
[robot_state_publisher-3] [INFO] [1755763271.587125040] [robot_state_publisher]: got segment base_link
[robot_state_publisher-3] [INFO] [1755763271.587149058] [robot_state_publisher]: got segment camera_link
[robot_state_publisher-3] [INFO] [1755763271.587160632] [robot_state_publisher]: got segment imu_link
[robot_state_publisher-3] [INFO] [1755763271.587171336] [robot_state_publisher]: got segment laser
[robot_state_publisher-3] [INFO] [1755763271.587180410] [robot_state_publisher]: got segment left_front_wheel_joint
[robot_state_publisher-3] [INFO] [1755763271.587189632] [robot_state_publisher]: got segment left_rear_wheel_hinge
[robot_state_publisher-3] [INFO] [1755763271.587199426] [robot_state_publisher]: got segment left_steering_hinge_joint
[robot_state_publisher-3] [INFO] [1755763271.587208447] [robot_state_publisher]: got segment right_front_wheel_joint
[robot_state_publisher-3] [INFO] [1755763271.587217132] [robot_state_publisher]: got segment right_rear_wheel_hinge
[robot_state_publisher-3] [INFO] [1755763271.587226428] [robot_state_publisher]: got segment right_steering_hinge_joint

```

On the virtual machine, create a new terminal and start.

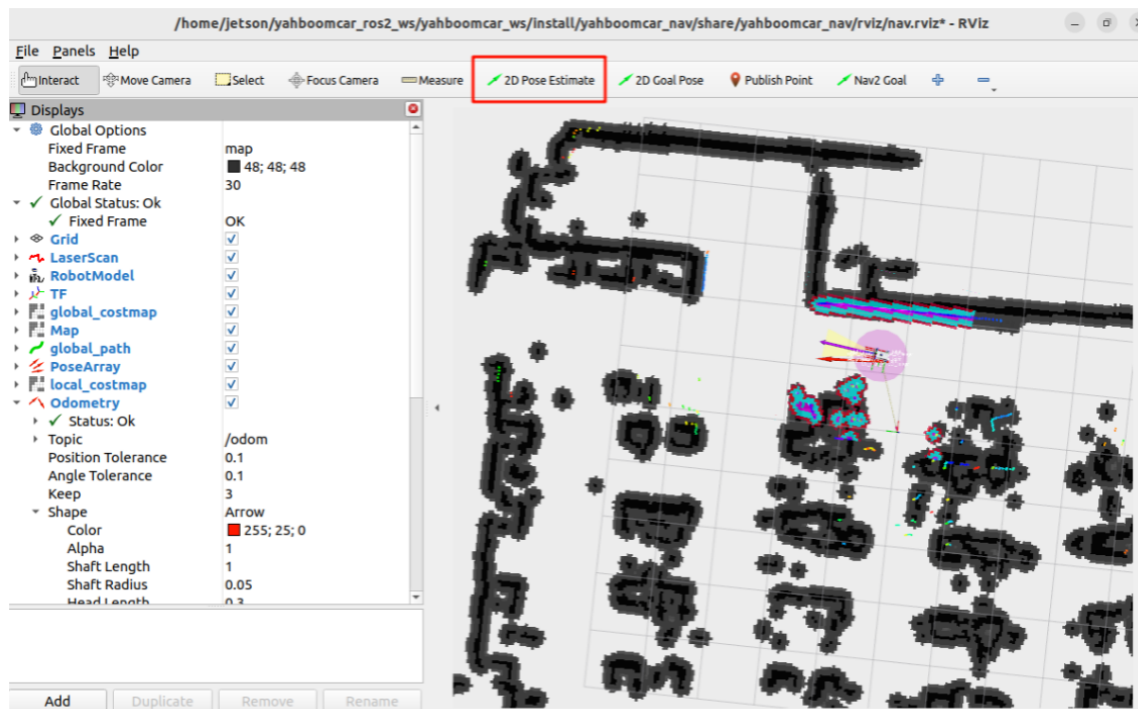
```
ros2 launch yahboomcar_nav display_nav_launch.py
```

Wait for the navigation algorithm to start before displaying the map. Then open a VM terminal and enter:

```
ros2 launch yahboomcar_nav navigation_teb_launch.py
```

Note: When running the car's mapping function, you must enter the save map command on the VM to save the navigation map.

After that, follow the navigation function startup process to initialize positioning. The rviz2 visualization interface will open. Click **2D Pose Estimate** in the upper toolbar to select it. Roughly mark the robot's position and orientation on the map. After initializing positioning, preparations are complete.



4.1.1 Raspberry Pi 5 and ORIN board startup steps:

For the Raspberry Pi 5, you need to first enter the Docker container; the ORIN board does not require this.

Open a terminal in Docker and enter the following command:

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

```
root@raspberrypi: ~
File Edit Tabs Help

root@raspb... x root@raspb... x root@raspb... x root@raspb... x

root@raspberrypi:~# ros2 launch largemodel largemodel_control.launch.py text_chat_mode:=True
[INFO] [launch]: All log files can be found below /root/.ros/log/2025-08-20-15-20-59-679615-raspberrypi-87377
[INFO] [launch]: Default logging verbosity is set to INFO

----- robot_type = A1, rplidar_type = tmini, camera_type = nuwa -----

-----robot_type = A1-----
[INFO] [ydlidar_ros2_driver_node-10]: process started with pid [87437]
[INFO] [ascamera_node-1]: process started with pid [87409]
[INFO] [joint_state_publisher-2]: process started with pid [87411]
[INFO] [robot_state_publisher-3]: process started with pid [87413]
[INFO] [Ackman_driver_A1-4]: process started with pid [87415]
[INFO] [base_node_A1-5]: process started with pid [87417]
[INFO] [imu_filter_madgwick_node-6]: process started with pid [87420]
[INFO] [ekf_node-7]: process started with pid [87425]
[INFO] [yahboom_joy_A1-8]: process started with pid [87432]
[INFO] [joy_node-9]: process started with pid [87435]
[INFO] [static_transform_publisher-11]: process started with pid [87440]
[INFO] [static_transform_publisher-12]: process started with pid [87449]
[INFO] [static_transform_publisher-13]: process started with pid [87451]
[INFO] [model_service-14]: process started with pid [87454]
[INFO] [action_service_nuwa-15]: process started with pid [87457]
[ascamera_node-1] [INFO] [1755674461.008407074] [ascamera_hp60c.camera_publisher]: get depth_width 640
[ascamera_node-1] [INFO] [1755674461.008622721] [ascamera_hp60c.camera_publisher]: get depth_height 480
[ascamera_node-1] [INFO] [1755674461.008647609] [ascamera_hp60c.camera_publisher]: get rgb_width 640
[ascamera_node-1] [INFO] [1755674461.008662739] [ascamera_hp60c.camera_publisher]: get rgb_height 480
[ascamera_node-1] [INFO] [1755674461.008674739] [ascamera_hp60c.camera_publisher]: get set_fps 25
[ascamera_node-1] [INFO] [1755674461.008685739] [ascamera_hp60c.camera_publisher]: get pub_tftree 1
[ascamera_node-1] [INFO] [1755674461.008696609] [ascamera_hp60c.camera_publisher]: hello world angstrong camera ros2 node
[ascamera_node-1] [INFO] [1755674461.008797201] [ascamera_hp60c.camera_publisher]: 2025-08-20 15:21:01[INFO] [CameraSrv.cpp] [35] [CameraSrv] Angstrong camera server
[ascamera_node-1] [INFO] [1755674461.008831553] [ascamera_hp60c.camera_publisher]: 2025-08-20 15:21:01[INFO] [CameraSrv.cpp] [45] [CameraSrv] Angstrong camera
sdk version:v1.2.28.20241021
[static_transform_publisher-11] [WARN] [1755674461.049432038] [1]: Old-style arguments are deprecated; see --help for new-style arguments
[static_transform_publisher-12] [WARN] [1755674461.070104001] [2]: Old-style arguments are deprecated; see --help for new-style arguments
```

Create a new terminal on the virtual machine and start the command.(For Raspberry Pi and Jetson Nano, it is recommended to run the visualization in the virtual machine)

```
ros2 launch yahboomcar_nav display_nav_launch.py
```

Wait for the navigation algorithm to start and the image will be displayed. Then open the Docker terminal and enter

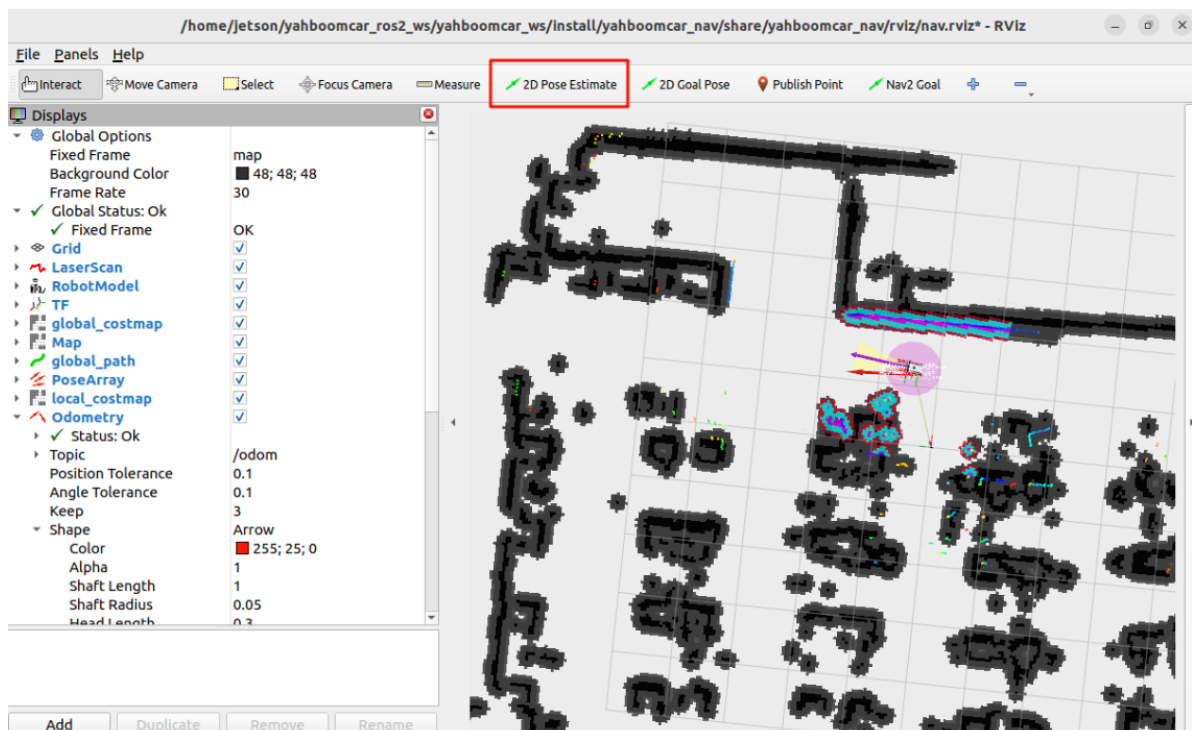
```
# Choose one of the two navigation algorithms (Raspberry Pi 5 and Jetson Nano
only have standard navigation)
# Standard navigation
ros2 launch yahboomcar_nav navigation_teb_launch.py

#Fast relocalization navigation (Orin board)
ros2 launch yahboomcar_nav localization_imu_odom.launch.py --use_rviz:=true
load_state_filename:=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/m
aps/yahboomcar.pbstream

ros2 launch yahboomcar_nav navigation_cartodwb_launch.py
maps:=/root/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_nav/maps/yahboomcar.
yaml
```

Note: yahboomcar.yaml and yahboomcar.pbstream must be mapped simultaneously, meaning they are the same map. Refer to the cartograph mapping algorithm for saving maps.

After that, follow the navigation function startup process to initialize positioning. The rviz2 visualization interface will open. Click **2D Pose Estimate** in the upper toolbar to enter the selection state. Roughly mark the robot's position and orientation on the map. After initializing positioning, preparations are complete.



Log into the same docker run from multiple terminals and start it.

```
ros2 run text_chat text_chat
```

```
root@raspberrypi: /
File Edit Tabs Help
*root@raspb... x *root@raspb... x root@raspb... x root@raspb... x
root@raspberrypi:~/# ros2 run text_chat text_chat
user input: 
```

4.2 Test Cases

Here are some test cases for reference; users can create their own dialogue commands.

- I'm in the office and feeling thirsty.

4.2.1 Example 1

Type "I'm in the office now, feeling a little thirsty" in the terminal. The terminal prints the following information:

```
root@raspberrypi: ~
File Edit Tabs Help
root@raspb... x root@raspb... x root@raspb... x pi@raspberr... x
root@raspberrypi:~/# ros2 run text_chat text_chat
user input: 我现在办公区，我有点口渴了
okay☺, let me think for a moment... -[INFO] [1755692529.223522879] [text_chat_node]: 决策层AI规划:1. 导航前往厨房区域
2. 观察是否存在矿泉水
3. 如果观察到矿泉水，反馈矿泉水和你之间的距离；否则前进0.5米后退0.5米
4. 导航返回办公区
user input: [INFO] [1755692530.957639015] [text_chat_node]: "action": ['navigation(G)'], "response": 好的，我这就去厨房给你找矿泉水，就像一个贴心的小助手一样
[INFO] [1755692557.989323529] [text_chat_node]: "action": ['seewhat()'], "response": 我已经到达厨房了，现在用我的火眼金睛寻找矿泉水
[INFO] [1755692560.396245271] [text_chat_node]: "action": ['get_dist(240,210)'], "response": 我看到前面有一个矿泉水瓶，正在测量它和我的距离
[INFO] [1755692564.370085347] [text_chat_node]: "action": ['navigation(F)'], "response": 我已经测到矿泉水离我1.004米，现在准备返回办公区啦
[INFO] [1755692590.463255325] [text_chat_node]: "action": ['finishtask()'], "response": 我已经完成全部任务啦，矿泉水在1.004米处，现在安全返回办公区了

```

The decision-making model outputs the planned task steps:

```
okay☺, let me think for a moment... -[INFO] [1755692529.223522879] [text_chat_node]: 决策层AI规划:1. 导航前往厨房区域
2. 观察是否存在矿泉水
3. 如果观察到矿泉水，反馈矿泉水和你之间的距离；否则前进0.5米后退0.5米
4. 导航返回办公区
```

The execution model then executes the task steps:

```
user input: [INFO] [1755692530.957639015] [text_chat_node]: "action": ['navigation(G)'], "response": 好的，我这就去厨房给你找矿泉水，就像一个贴心的小助手一样
[INFO] [1755692557.989323529] [text_chat_node]: "action": ['seewhat()'], "response": 我已经到达厨房了，现在用我的火眼金睛寻找矿泉水
[INFO] [1755692560.396245271] [text_chat_node]: "action": ['get_dist(240,210)'], "response": 我看到前面有一个矿泉水瓶，正在测量它和我的距离
[INFO] [1755692564.370085347] [text_chat_node]: "action": ['navigation(F)'], "response": 我已经测到矿泉水离我1.004米，现在准备返回办公区啦
[INFO] [1755692590.463255325] [text_chat_node]: "action": ['finishtask()'], "response": 我已经完成全部任务啦，矿泉水在1.004米处，现在安全返回办公区了
```

The screenshot displays a ROS2 environment. On the left, a terminal window shows a text chat interface with the following text:

```
root@raspberrypi1:~# ros2 run text_chat
user input: 我现在在办公区，我有点口渴了
okay☺️, let me think for a moment... - [
观察是否存在矿泉水
如果观察到矿泉水，反馈矿泉水和你之间的
导航返回办公区
user input: [INFO] [1755692530.957639015
[INFO] [1755692557.989323529] [text_chat
```

On the right, a camera feed window titled "Saved Image" shows a view of an office. A person is standing in the background, and a clear plastic water bottle is in the foreground. The camera feed includes a toolbar with navigation and zoom controls. At the bottom of the camera feed, the coordinates and bounding box are displayed: `(x=537, y=225) ~ R:24 G:30 B:28`.

```
File Edit Tabs Help
* root@raspb... × root@raspb... × root@raspb... × pi@raspberr... ×

root@raspberrypi1:~# ros2 run text_chat text_chat
user input: 我现在在办公区，我有点口渴了
okay☺, let me think for a moment... -[INFO] [1755692529.223522879] [text_chat_node]: 决策层AI规划:1. 导航前往厨房区域
2. 观察是否存在矿泉水
3. 如果观察到矿泉水，反馈矿泉水和你之间的距离；否则前进0.5米后退0.5米
4. 导航返回办公区
user input: [INFO] [1755692530.957639015] [text_chat_node]: "action": ['navigation(G)'], "response": 好的，我就去厨房给你找矿泉水，就像一个贴心的小助手一样
[INFO] [1755692557.989323529] [text_chat_node]: "action": ['seewhat()'], "response": 我已经到达厨房了，现在用我的火眼金睛寻找矿泉水
[INFO] [1755692560.396245271] [text_chat_node]: "action": ['get_dist(240,210)'], "response": 我看到前面有一个矿泉水瓶，正在测量它和我的距离
[INFO] [1755692564.370085347] [text_chat_node]: "action": ['navigation(F)'], "response": 我已经测到矿泉水离我1.004米，现在准备返回办公区啦
[INFO] [1755692590.46325325] [text_chat_node]: "action": ['finishtask()'], "response": 我已经完成全部任务啦，矿泉水在1.004米处，现在安全返回办公区了

user input: 结束当前任务
okay☺, let me think for a moment... -[INFO] [1755692770.129807838] [text_chat_node]: "action": ['finish_dialogue()'], "response": 好的，任务已经结束了，有需要再叫我哦
user input: █
```

Jetson Nano, Raspberry Pi Host:

You need to first enter Docker

```
#NUWA Camera User
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/largemodel/largemodel/action_service_
nuwa.py
#USB Camera User
/root/yahboomcar_ros2_ws/yahboomcar_ws/src/largemodel/largemodel/action_service_
usb.py
```

5.1 Example 1

action_service.py :

This example uses the **seewhat**, **navigation**, **load_target_points**, and **get_dist** methods in the **CustomActionServer** class. **seewhat**, **navigation**, **load_target_points**, and **get_dist** are described in the previous section [2.Multimodal visual understand], [6.Multimodal visual understand + Depth Camera Distance Question Answering], [7.Multimodal visual understand+SLAM navigation] have all been explained, and there are no new procedures in this chapter.