# Tutorial of Lerobot on Jetson series device

Hi, everyone! I'm Peter, a robot engineer of Robify. In this tutorial, I will assist you in setting up the environment for the large-scale open-source robotic arm project Lerobot on the Jetson series devices, as well as deploying your first model trained using the ACT algorithm! Come and keep up with my pace!

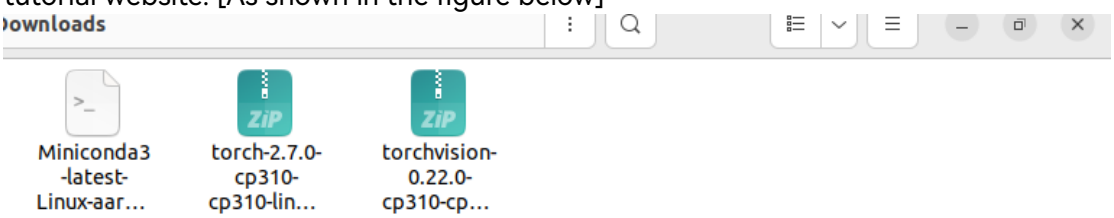
## Preparation

Jetson Orin NX Developer Kit (the recommended hard disk memory is at least 256 GB. It should also be equipped with the Ubuntu 22.04 LTS system image, CUDA 12.6, jetpack 6.0+, Pytorch and other necessary software.)
SO-ARM100
Two cameras
A number of related cables (such as HDMI to DP data cable, USB to Type-C data cable)
[Notice: If you purchase the development equipments related to this tutorial from our company's website, they will include all the necessary development equipment and cables mentioned above. ]

## Development Environment Setting

Generally speaking, if you purchase development board equipment from our company, the hard drive included with the development board has already been pre-installed with a system that contains all the necessary environments required for this development tutorial. However, if you are willing to set up your own development environment and learn some development knowledge related to the Linux system, or during your learning process with the development board for the Lerobot project, you encountered some fatal operations that you didn't understand and caused the system to crash, making it unusable, then you need to re-flash the original NVIDIA system image and set up the environment, you could complete the setup of the development environment according to the following steps. [Users who purchased the development board from our company can directly proceed to the "Development in Progress" module for learning.]

Above all, I highly recommend that you first install Anaconda or Miniconda on your system to isolate the Python development environment of your Lerobot project from the host system. This will prevent any potential conflicts with the Python software modules required by your other projects. You can download the software installation packages needed for the upcoming tutorials in advance from our company's robotic tutorial website. [As shown in the figure below]



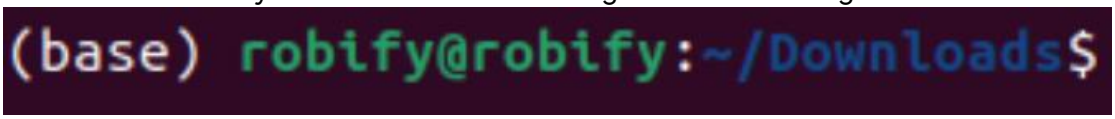
Then, open the terminal window on your system and enter the following commands one by one (press Enter after typing each command and wait for the system to complete the command before entering the next one, do as the same in the subsequent tutorials).

```
bash
  cd ~/Downloads
  sudo chmod +x Miniconda3-latest-Linux-aarch64.sh   ##This is a command, and the next line is another command.##
  ./Miniconda3-latest-Linux-aarch64.sh
```

After executing all the above commands one by one, you can see the installation process of Miniconda in your system terminal window. It will ask you to input "accept" and press the Enter key, or directly press the Enter key to confirm. Or you can input "yes" and then press the Enter key. For all the inquiries during the installation process, just follow what is mentioned above. (Unless you are a seasoned Linux engineer with own requirements or preferences for the installation path of system software.) When the process is completed, please execute the following command to make the Miniconda software run on your system.

```
bash
  source ~/.bashrc
```

You would see that your terminal window changes to the following:




This indicates that you have successfully completed the installation and operation of Miniconda!

Next, let's use the exclusive commands of Miniconda to create an isolated Python development environment. Please input the commands below one by one to your system terminal and execute:

```
bash
    conda create -y -n lerobot python=3.10
    conda activate lerobot
    cd ~/
    sudo apt-get install git
    git clone https://github.com/bladewin-smith/Jetson_Orin_NX-Lerobot.git
```

In order to successfully deploy the trained model in the subsequent development tutorials, we need to install some necessary Python modules for this environment (if your system does not have CUDA installed, please click the link to follow the NVIDIA official tutorial to install CUDA 12.6). I assume that before this, you have already downloaded the software installation package from our company's website as I instructed. So, let's proceed with the following steps:

```
bash
    cd ~/Downloads
    sudo apt-get install python3-pip libopenblas-base libopenmpi-dev libomp-dev
    pip3 install 'Cython<3'
    pip3 install numpy
    pip3 install torch-2.7.0-cp310-cp310-linux_aarch64.whl torchvision-0.22.0-cp310-cp310-linux_aarch64.whl
```

```
conda install ffmpeg -c conda-forge
cd ~/Jetson_Orin_NX-Lerobot && pip install -e ".[feetech]"
conda install -y -c conda-forge "opencv>=4.10.0.84"
conda remove opencv
pip3 install opencv-python==4.10.0.84
conda install -y -c conda-forge ffmpeg
conda uninstall numpy
pip3 install numpy==1.26.0
```

During the above installation process, if the terminal outputs some red-font warnings, please ignore them first. Generally, these warnings will disappear when we complete all the above steps. However, sometimes some warnings do not disappear. You can check which necessary Python module packages are missing in your environment by entering the command "pip check" in the terminal window, and then install them one by one using "pip install".

Eventually, let's check through the following command whether the key module Pytorch, which is necessary for the operation of the Lerobot project, has been installed successfully, and whether it can be used to accelerate model training and evaluating with CUDA:

```
bash
python ##After entering this command, you can access the interactive interface of the Python language. During the interaction, you can input "exit()" to exit the interface.##
import torch
print(torch.cuda.is_available())
```

If you can see "True" printed out in the above Python interactive interface, then congratulations! You have successfully completed the setup of the basic environment for the Lerobot project development!!! Let's move on to the next stage of our study!

## Development in Progress

Entering this module, I assume that you have already prepared the environment necessary for project development. Now, let's first follow the instructions in the manual to complete the wiring and fixation of the robotic arm, then calibrate the main arm and the slave arm of the SO-ARM100 robotic arm to ensure they can operate smoothly in the subsequent remote operation and data collection operations. Please enter the following commands in the terminal one by one and follow the linked video for the operation:

```
bash
sudo chmod 777 /dev/tty/ACM0
lerobot-calibrate --robot.type=so100_follower --robot.port=/dev/ttyACM0 --robot.id=so100_follower
lerobot-calibrate --teleop.type=so100_leader --teleop.port=/dev/ttyACM0 --teleop.id=so100_leader
```

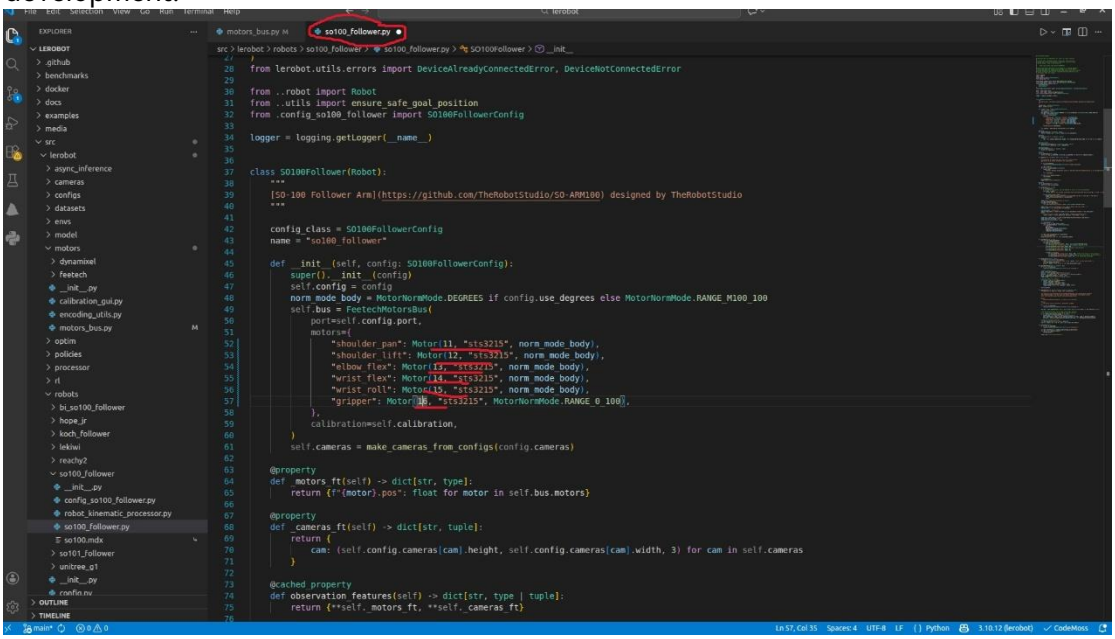
If the calibration is successful, you will see the information shown in the picture on your system terminal window:

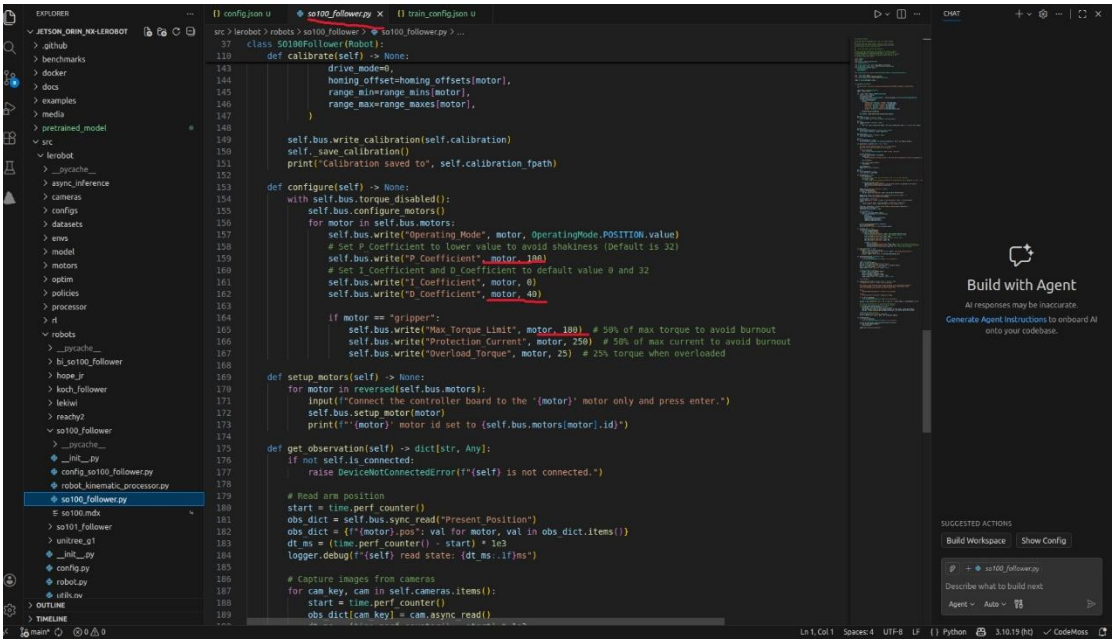
```
(lerobot) robify@ubuntu:~/lerobot$ lerobot-calibrate --robot.type=so100_follower --robot.port=/dev/ttyACM0 --robot.id=so100_follower
INFO 2026-01-20 10:14:48 calibrate.py:74 {'robot': {'calibration_dir': None,
          'cameras': {},
          'disable_torque_on_disconnect': True,
          'id': 'so100_follower',
          'max_relative_target': None,
          'port': '/dev/ttyACM0',
          'use_degrees': False},
 'teleop': None}
INFO 2026-01-20 10:14:49 follower.py:104 so100_follower SO100Follower connected.
INFO 2026-01-20 10:14:49 follower.py:121
Running calibration of so100_follower SO100Follower
Move so100_follower SO100Follower to the middle of its range of motion and press ENTER....
Move all joints except 'wrist_roll' sequentially through their entire ranges of motion.
Recording positions. Press ENTER to stop...

-------------------------------------------
-------------------------------------------
NAME            |    MIN |    POS |    MAX
shoulder_pan    |    834 |   2097 |   3457
shoulder_lift   |    797 |    798 |   3334
elbow_flex      |    799 |   3005 |   3024
wrist_flex      |    763 |   2908 |   3224
gripper         |   2039 |   2047 |   3464
Calibration saved to /home/robify/.cache/huggingface/lerobot/calibration/robots/so100_follower/so100_follower.json
INFO 2026-01-20 10:16:19 follower.py:232 so100_follower SO100Follower disconnected.
(lerobot) robify@ubuntu:~/lerobot$ lerobot-calibrate --teleop.type=so100_leader --teleop.port=/dev/ttyACM0 --teleop.id=so100_leader
INFO 2026-01-20 10:18:01 calibrate.py:74 {'robot': None,
 'teleop': {'calibration_dir': None,
            'id': 'so100_leader',
            'port': '/dev/ttyACM0'}}
INFO 2026-01-20 10:18:02 00_leader.py:81 so100_leader SO100Leader connected.
INFO 2026-01-20 10:18:02 00_leader.py:98
Running calibration of so100_leader SO100Leader
Move so100_leader SO100Leader to the middle of its range of motion and press ENTER....
Move all joints except 'wrist_roll' sequentially through their entire ranges of motion.
Recording positions. Press ENTER to stop...

-------------------------------------------
-------------------------------------------
NAME            |    MIN |    POS |    MAX
shoulder_pan    |    772 |   2025 |   3500
shoulder_lift   |    788 |    791 |   3290
elbow_flex      |    894 |   3076 |   3003
wrist_flex      |    838 |   2652 |   3131
gripper         |   2040 |   2050 |   3287
Calibration saved to /home/robify/.cache/huggingface/lerobot/calibration/teleoperators/so100_leader/so100_leader.json
INFO 2026-01-20 10:19:43 0_leader.py:159 so100_leader SO100Leader disconnected.
(lerobot) robify@ubuntu:~/lerobot$
```

One important point to note is that if you were previously familiar with the Lerobot project, you will find that in the calibration commands we provided, the angle data of the robotic arm's servo is obtained from the same ttyACM port. This is because we have made the following improvements to the algorithm as shown in the picture, in order to further reduce the cost of the robotic arm application! Subsequently, you can refer to the improvements we have made in the following figures to carry out further development.





After calibrating the robotic arm, we need to first find the names of the two camera ports connected to your Jetson device. They are usually attached to the /dev/video* ports. Run the following command in the terminal (it is strongly recommended that only the USB ports on the Jetson device are connected to the two cameras required by this tutorial when running):

```bash
python src/lerobot/scripts/lerobot_find_cameras.py
```

```
(lerobot) jetson@robify:~/Jetson_Orin_NX-Lerobot$ python src/lerobot/scripts/lerobot_find_cameras.py
ERROR:__main__:Error finding RealSense cameras: name 'rs' is not defined

--- Detected Cameras ---
Camera #0:
  Name: OpenCV Camera @ /dev/video0
  Type: OpenCV
  Id: /dev/video0
  Backend api: V4L2
  Default stream profile:
    Format: 16.0
    Fourcc: YUYV
    Width: 640
    Height: 480
    Fps: 30.0
-------------------
Camera #1:
  Name: OpenCV Camera @ /dev/video2
  Type: OpenCV
  Id: /dev/video2
  Backend api: V4L2
  Default stream profile:
    Format: 16.0
    Fourcc: YUYV
    Width: 640
    Height: 480
    Fps: 30.0
-------------------

Finalizing image saving...
Image capture finished. Images saved to outputs/captured_images
```

If everything goes well, you will see an output similar to the one above. Please make sure to remember the port numbers of these two cameras, as they are crucial for our subsequent remote operation, data collection, and the deployment of the trained model!

Nextly, let's start the remote operation tutorial! Please enter the following command into your system terminal. [Notice: Before running this command, make sure that the camera port parameters you have are consistent with the actual ones. If not, please modify them according to the actual port numbers printed out by your system previously.]

```bash
lerobot-teleoperate --robot.type=so100_follower --robot.port=/dev/ttyACM0 --robot.id=so100_follower --teleop.type=so100_leader --teleop.port=/dev/ttyACM0 --teleop.id=so100_follower --display_data=true --robot.cameras="{ front: {type: opencv, index_or_path: /dev/video0, width: 640, height: 480, fps: 30}, wrist: {type: opencv, index_or_path: /dev/video2, width: 640, height: 480, fps: 30}}"
```

[Parameter Explanation]

“--robot.type”: The type of your follower robotic arm, and the type we are using here is from the official Huggingface developer kit. Just fill in the parameters as indicated in the above command. Of course, if you use other robotic arms in your future independent development, you must modify this parameter. The configuration file of the robotic arm needs to exist in your workspace. It can be other brands of robotic arms that are already available in the Huggingface repository, or it can be your own written configuration file for the robotic arm (this requires you to have strong independent development capabilities).

“--robot.port”: The port number of your device connected via the follower arm interface. It is usually one of the ports in the /dev/ttyACM* group. Before running the above command, please make sure to use the command "sudo chmod 777" to add read and write permissions to the port. Otherwise, during the running process, the system will report an error that the mechanical arm interface port cannot be found, resulting in the failure of the operation.

“--robot.id”: The name of your follower robotic arm. To modify this parameter, you can find the following line in the “so100_follower.py” file and make your changes.

```
43        name = "so100_follower"
```

“--teleop.type”: The type of your leader robotic arm. The usage method and modification suggestions are basically the same as explaination for the follower robotic arm that mentioned above.

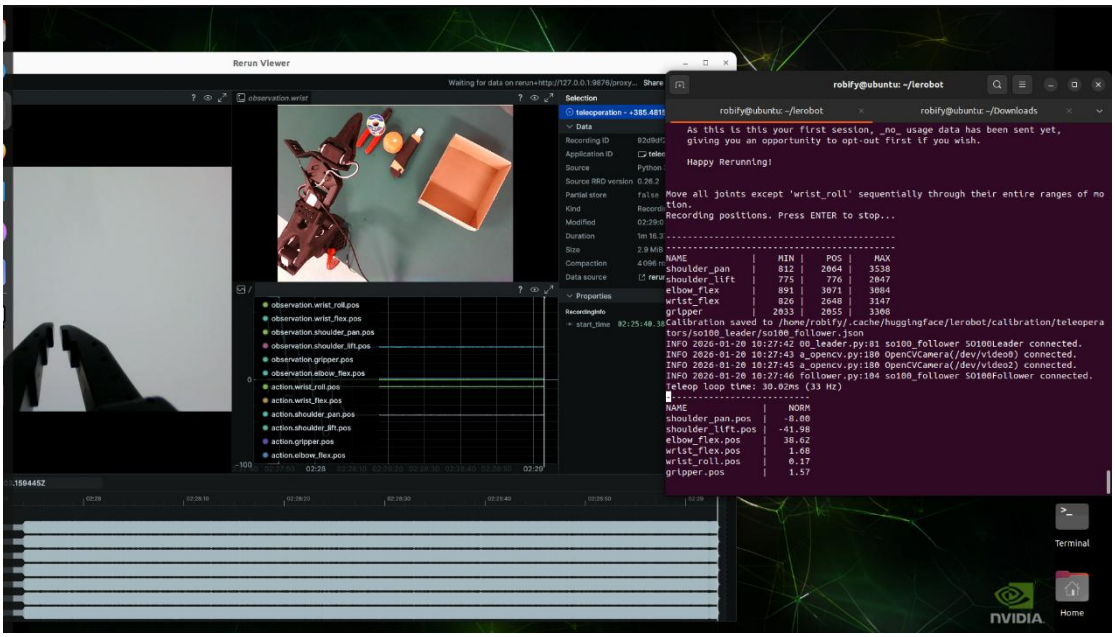
“--teleop.port”: The port number of your device connected via the leader arm interface. The usage method and modification suggestions are basically the same as explaination for the follower robotic arm that mentioned above.

“--teleop.id”: The name of your leader robotic arm. The usage method and modification suggestions are basically the same as explaination for the follower robotic arm that mentioned above.

“--display_data”: The parameter indicating whether the image data transmitted by the cameras is displayed through the "rerun" software, and it can be either "true" or "false".

“--robot.cameras”: The camera parameters of the robotic arm system generally do not need to be modified. However, if during your previous execution of the Python script for searching the camera, the system printed out different information from the parameters mentioned above, then you should make the modifications according to the actual situation by referring to them.

OK, if the above command is executed successfully, you will see a screen similar to the one shown in the picture below on your monitor.




After completing the tutorial on the remote operation of the robotic arm, let's proceed to remotely operate the robotic arm and simultaneously collect data! The system terminal commands for data collection are roughly the same as those for remote operation, except that few modifications have been made and several additional parameters have been added. As follows:

bash

```
lerobot-record --robot.type=so100_follower --robot.port=/dev/ttyACM0 --robot.cameras="{ front: {type: opencv, index_or_path: /dev/video0, width: 640, height: 480, fps: 30}, wrist: {type: opencv, index_or_path: /dev/video2, width: 640, height: 480, fps: 30}}" --robot.id=so100_follower --teleop.type=so100_leader --teleop.port=/dev/ttyACM0 --teleop.id=so100_leader --dataset.push_to_hub=false --dataset.num_episodes=50 --dataset.root="/home/jetson/lerobot/robify/data1" --dataset.episode_time_s=300 --dataset.reset_time_s=30 --display_data=true --dataset.repo_id=${HF_USER}/record_data1 --dataset.single_task="Sort out things"
```

[Parameter Explanation]

Now, let me explain the several newly added parameters.

“--dataset.push_to_hub”: Whether the representative will upload the dataset you collected to the cloud data repository of your account on the Huggingface official website. If this parameter is set to true, you will need to perform more complex operations such as logging in to the official website and uploading your dataset. These operations will be affected by your network, and in the worst case, it may lead to the failure of dataset collection. Therefore, I generally suggest that you save the dataset locally like me, that is, set this parameter to false.

“--dataset.num_episodes”: The parameter representing the number of data groups contained in the dataset you want to collect. I suggest you do not set this parameter to a very large number, because collecting data is a very tiring work. Later, I will tell you how to combine multiple small-sized datasets into a larger-sized dataset.

“--dataset.root”: The parameter that represents where you want to store the collected data set. I suggest you use the absolute path of the storage location like I do, because when I was trying to create the dataset, I often encountered system errors

related to the path. Therefore, using the absolute path method when passing in the parameters can avoid many of the errors I encountered before. Another important point to note is that if you feel the size of the dataset you have collected is a bit small, you can run the above command to collect more data. However, do not forget to modify this parameter. You only need to change the last "data1" to "data2" and make the same modification to the value of the "--dataset.repo_id" parameter. Then you can collect the dataset for the same action task again.

“--dataset.episode_time_s”: This parameter is used to set the time limit representing the period during which you collect a set of data. You can adjust this parameter according to the complexity of the actual operations that you want the robotic arm to perform during the training process. I suggest you set this parameter to a relatively large value so that you have more time to familiarize yourself with how to operate the main robotic arm to collect the data set. Of course, if you are really unsure about how to set this time limit value, you can arbitrarily set a value far exceeding the time required for your designed action task to complete, and then during the data collection process, when you have completed one data collection but the time has not reached the limit value, you can press the right mouse button "→" to exit the background data collection time limit timing loop and prematurely terminate the current data collection process without worrying about data loss. This operation also applies to the parameter "--dataset.reset_time_s" to exit the background reset environment time limit timing loop. In addition, the official Huggingface has also defined the left mouse button "←", that is, to cancel the current data recording and re-collect. Pressing the "Esc" key can prematurely exit the global data recording loop and save the collected data to the data set. Completing this data set collection work (note: the number of data in the data set at this time depends on the number of data you have collected!).

“--dataset.reset_time_s”: This parameter represents the time limit for restoring the environment in which the data was collected when you are collecting the dataset. You can set this parameter to be larger as well, just like the previous parameter, to give yourself sufficient time to restore the environment.

“--dataset.repo_id”: The name of your dataset in the cache storage space. You can modify it according to your actual needs. Also, you may notice that I have passed the parameter ${HF_USER} here. This is a very convenient way to modify an important global parameter throughout the development environment. You can do this by entering the following commands one by one in the terminal, and adding a global parameter like "HF_USER=robify" at the end of the “~/.bashrc” file.

| bash |
| --- |
| sudo vim ~/.bashrc ##Then, you will then enter the interface for editing the ~/.bashrc file. First, move the mouse to scroll the interface to the bottom. Then press the 'i' key, add the parameters, and press the 'esc' key on the keyboard to exit the file editing. Finally, press the 'shift+:' key, 'w', 'q', and 'Enter' keys in sequence to save your modifications to the file. ##<br>source ~/.bashrc ##This operation will make the modifications you made to the ~/.bashrc file take effect throughout the entire development environment. However, please note that this operation will also cause you to exit the conda environment you have set up specifically for Lerobot development. Therefore, when following the tutorial and continuing with the subsequent steps, you should first run the command |

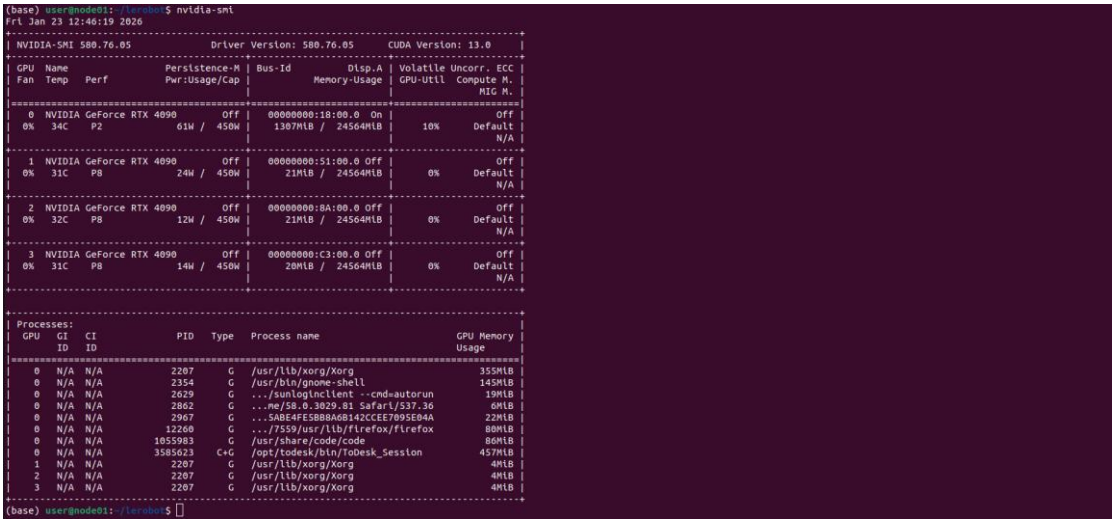
"conda activate lerobot" to return to the Lerobot development environment.##

“--dataset.single_task”: This parameter represents the name you give to the action tasks that the robotic arm needs to perform. You can modify it according to your needs, but please remember to add double quotation marks!

During the data collection process, there are some important points that you must pay attention to. Otherwise, the quality of the data set you collect will be poor, which will ultimately affect the performance of the model you train in completing the tasks you expect: Firstly, you must be aware that there should be no other items besides the necessary ones for the designed action task within the global camera's view. Even the leader robotic arm cannot appear within the global camera's view during the data collection process. However, you can place a fixed reference item at a fixed position to facilitate the restoration of the data collection environment. For example, a utility knife. Of course, you can also use a table with holes to facilitate the data collection. Moreover, the follower robotic arm must be within the global camera's view. Otherwise, it will be difficult for the model to learn the posture of the follower robotic arm when completing the designed action task. Secondly, you should try to avoid any displacement of the global camera during the data collection process. This is also an important factor affecting the quality of the data set. Finally, you should try to ensure that the environment where you collect the data set (including lighting conditions, etc.) is consistent with the environment when using the trained model for inference and verification. Otherwise, the model's inference and verification will not achieve a good effect.

Having arrived here, I believe you have successfully completed the data collection for training the model. Now, let's move on to the tutorial on model training.

First of all, I need to clarify one point: Training the model can be done on your Jetson series devices. However, during my own process of model training before writing this tutorial, I found that even on the best device in my possession, the Jetson Orin NX 16GB, and under the Super mode officially launched by NVIDIA for model training, the speed of training the model was still unsatisfactory. Therefore, you need to adjust the method of training the model according to your actual needs. I suggest that you rent a server with greater computing power or use your laptop (or desktop computer) with an RTX series graphics card (at least the 30 series, preferably the 40 series or above, of course, you can also use other brands of graphics cards to complete the task of model training. However, I have not tried it and do not know if the ecosystem of other brands of graphics cards has methods similar to NVIDIA's for accelerating model training using CUDA. If you have innovative practices in your independent development and want to share them in our company's robot enthusiasts community, please do! ) for model training. This can greatly increase your development speed. By the way, the selection of the CPU is also very important for model training. You can refer to the relevant hardware information of my server in the picture below to choose or build the computer you will use to train the model. Last but not least, if you choose to train the model on a computer or server other than the Jetson series devices, it is recommended to use the Linux system. In this case, you can follow the previous steps to set up the environment. However, one thing to note is that the CUDA and Pytorch modules should be selected in the x86_64 architecture version instead of the aarch64 architecture version.


In the previous tutorial on collecting data, I mentioned that collecting a large dataset in one go is a very tiring task. Therefore, I suggest collecting small batches of data first and then merging them into a larger dataset. Now, I will teach you how to merge the datasets. First, transfer the dataset you collected on the Jetson series devices to the device where you are training the model. Then, enter the following command in the system terminal and execute it:

| bash |
| --- |
| lerobot-edit-dataset --repo_id robify/data --operation.type merge --operation.repo_ids "['/home/user/lerobot/robify/data2', '/home/user/lerobot/robify/data3', '/home/user/lerobot/robify/data4', '/home/user/lerobot/robify/data5', '/home/user/lerobot/robify/data6', '/home/user/lerobot/robify/data7']" |

[Parameter Explanation]

“--repo_id”: The name of the folder where your merged dataset is stored is located at the path "/home/user/.cache/huggingface/lerobot". You can modify it as needed by yourself.

“--operation.type”: The type of operation you need to perform on the dataset, the parameter we have passed in here is "merge".

“--operation.repo_ids”: The name of the dataset you want to merge. Here, we pass in the absolute path where the dataset is stored, which can avoid many unnecessary unexpected situations.

For the usage of commands for performing other operations on the dataset, please refer to the file /home/user/lerobot/src/lerobot/datasets/dataset_tools.py.

The above command is what I used before training the model to combine the 6 data sets I collected. After using this command to merge them into one data set, the final large data set will be stored in the path “/home/user/.cache/huggingface/lerobot/robify/data”. We don't need to move it to another location for viewing. During the model training process, you can simply pass this path as the dataset path parameter when training the model. After the data set merging is successful, it will be similar to the following figure:


After obtaining the appropriate dataset, it's time to train the target model. This tutorial takes the ACT algorithm for model training as an example, showing you how to train an ACT model. If you want to use other algorithms to train the model, you can refer to the usage cases in the official tutorials of Huggingface for training or follow our subsequent Demo tutorial updates. Next, you can input the following command into your system terminal to train the model:

| bash |
| --- |
| lerobot-train --dataset.repo_id=robify/data --policy.type=act --output_dir=outputs/train/act_robify --job_name=sort_out_thing --policy.device=cuda --batch_size=8 --steps=150000 --wandb.enable=false --policy.push_to_hub=false --dataset.root="/home/user/.cache/huggingface/lerobot/robify/data" |

[Parameter Explanation]

“--policy.type”: The type of algorithm used in training the model is the ACT algorithm.

“--output_dir”: The path for saving the output of the trained model can be modified according to your own requirements.

“--job_name”: The name of the task completed by the model. This is a relatively unimportant parameter. You can modify it according to your own needs, and this will not affect the subsequent deployment of the model.

“--policy.device”: The device used to accelerate the training process when training the model. Here, we use CUDA to speed up the model training. Please modify this parameter according to your specific situation.

“--batch_size”: This parameter represents the number of samples in the dataset used for each parameter update during the model training process. Here, you can simply input the value 8. If you have a deep and unique understanding of deep learning training, you can adjust it by yourself.

“--steps”: The training step parameter indicates the minimum loss value you want your model to achieve (the lower the loss value, the better the deployment effect of the trained model). However, it's important to note that this parameter doesn't necessarily need to be set to the maximum value. Because depending on the difficulty level of the action tasks that the model is supposed to perform to drive the robotic arm, the loss

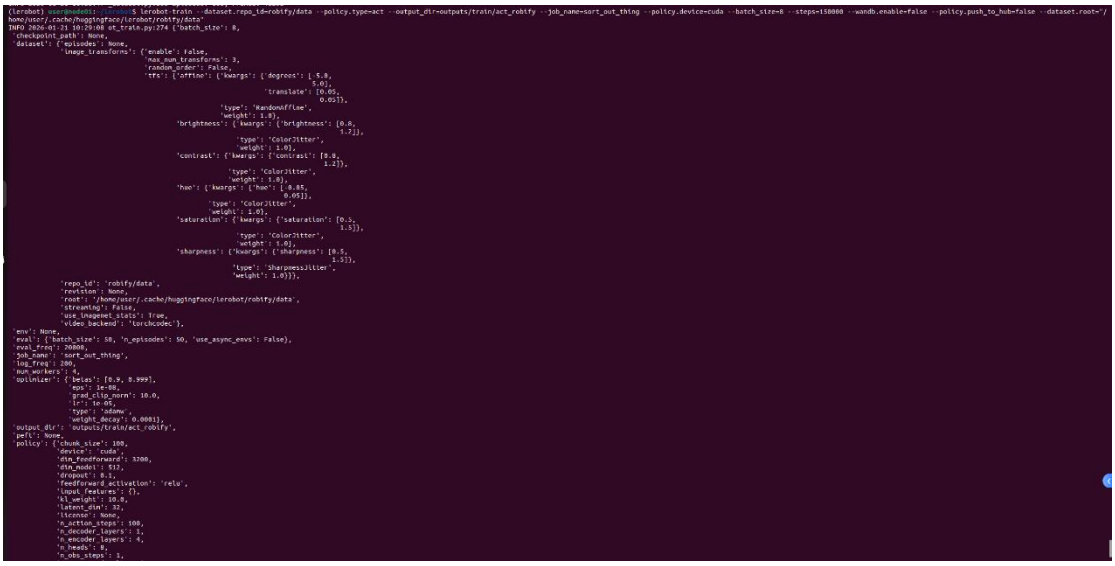
value will converge at a certain appropriate number of training steps. At this point, even if you increase the training steps further, there won't be much practical improvement in the deployment effect.

“--wandb.enable”: This parameter indicates whether you want to use the Weights and Biases tool to visualize the training chart. If you want to use it, you need to pass the parameter "true" and make sure you have logged into your own account via wandb login.

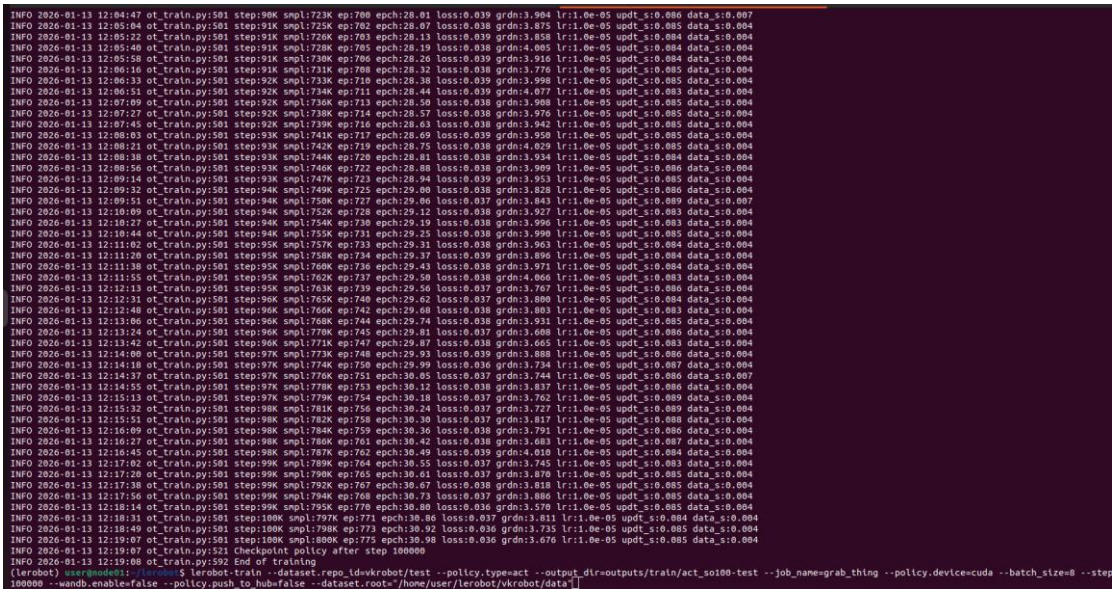
“--policy.push_to_hub”: Whether to upload the trained model file to the cloud database of Huggingface, this parameter can be set to true or false depending on your actual needs.

“--dataset.root”: The storage path for the dataset used in model training. Here, we can simply pass in the parameter based on the dataset storage path after the previous data merging process.

OK. When you see the same information as shown in the picture below on your system terminal, it means that the model training has begun.



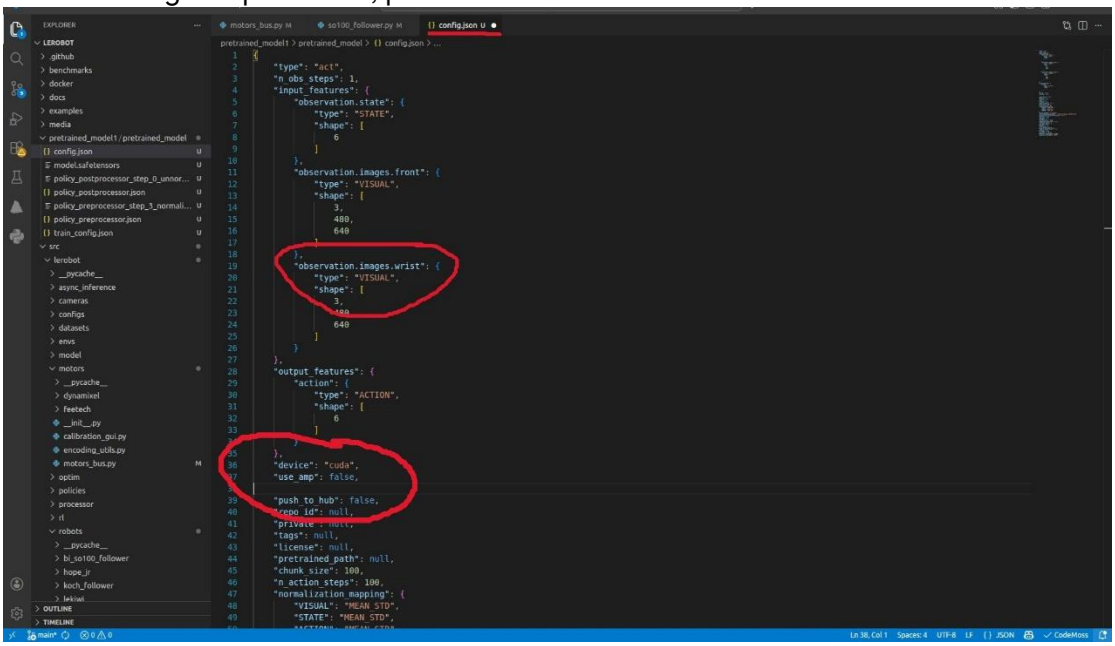
Please be patient and wait for the training results to be displayed. If everything goes smoothly, you will see the same system terminal output result as this one.



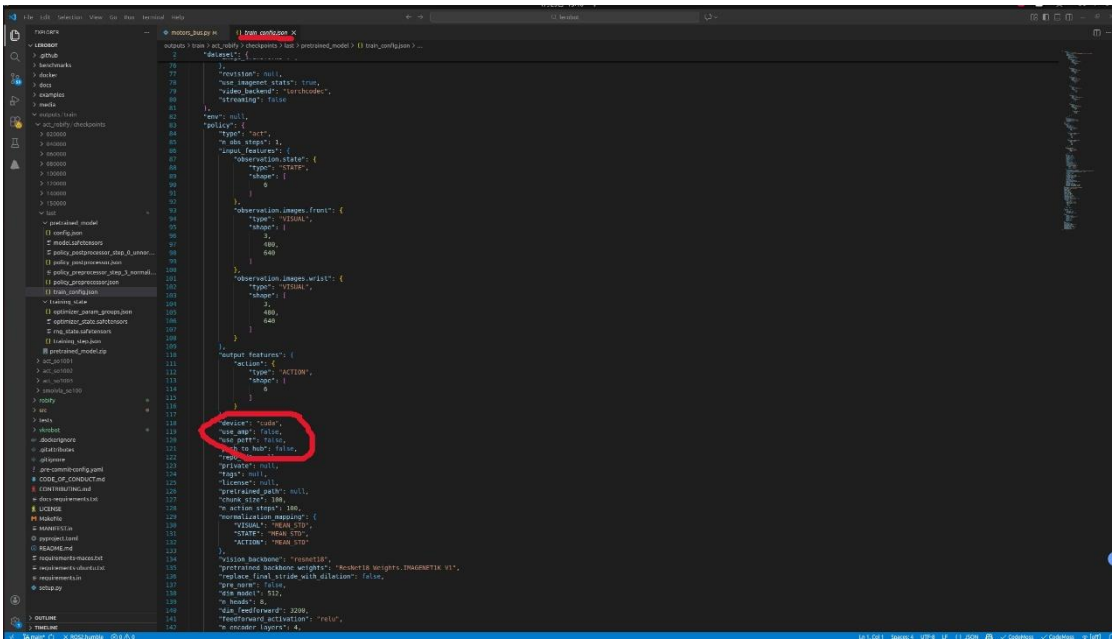
In the folder pointed to by the model output path parameter you passed in when running the training model command above, you can see many model files generated when the training steps reach a certain stage. Generally, we only need to click on the folder pointed to by the "last" folder, find the "pretrained_model" folder inside it, and compress it before transferring it to our Jetson series device to proceed with the next step of model deployment and learning.

Now, let's move on to the final step of the tutorial - using the trained model for inference verification!

Before starting the verification of the reasoning, we need to make some modifications to the configuration file of the trained model. Otherwise, during the running process, it will report many uncomfortable running failure errors, which took me a lot of time to troubleshoot. Here, I would like to emphasize this point particularly so that you can remember it well and avoid acting like a lost baton-wielding insect when independently developing in the future. Please look at the red circled areas in the following picture. You need to find the position of the parameters in the picture and modify them to be consistent with the parameters in the following picture. Additionally, in the second red circle, I deleted a parameter "use_peft" and its value, so there is an extra blank line. The blank line here is to inform you of the position of this parameter. After deleting this parameter, please be sure to delete this blank line as well!




It is necessary to point out again that "use_peft" and its value will also appear in the train_config.json file under the pretrained_model folder. You need to locate it and delete it.



Well, I'm sure that with the guidance of this tutorial, you have successfully completed all the preparatory work for model inference verification and have carried out the corresponding learning and practice. Now, please enter the command in your terminal and run it:

| bash |
| --- |
| lerobot-record --robot.type=so100_follower --robot.port=/dev/ttyACM0 --robot.cameras="{ front: {type: opencv, index_or_path: /dev/video0, width: 640, height: 480, fps: 30}, wrist: {type: opencv, index_or_path: /dev/video2, width: 640, height: 480, fps: 30}}" --robot.id=so100_follower --display_data=true --dataset.repo_id=${HF_USER}/eval_so100_test --dataset.single_task="Sort out things" --dataset.episode_time_s=3000 --dataset.num_episodes=10 --policy.path=/home/jetson/lerobot/pretrained_model |

[Parameter Explanation]

You may notice that this command is quite similar to the one used when collecting data. Therefore, here I will only explain the new meanings of the three parameters.

“--dataset.repo_id”: The name of the storage folder for the dataset in the path "/home/user/.cache/huggingface/lerobot/" is easily identifiable. It can be observed that the folder names where the data recorded during the inference verification are stored all have an additional prefix "eval_", which indicates that this dataset was collected during the inference verification process. This is the official naming convention of Huggingface. In the future, when you conduct your own model inference verification, please keep this principle in mind! Last but not least, if you want to conduct another model inference verification after exiting the previous one, please make sure to first input and run the command "sudo rm -rf /home/jetson/.cache/huggingface/lerobot/vkrobot/eval_so100_test" in your system terminal to delete the data recorded during the previous inference verification of the model. Alternatively, you can choose to modify the name of the folder where the data is saved in this parameter value, and then conduct the new model inference verification.

“--dataset.episode_time_s”: During the process of reasoning verification, the model-driven robotic arm is required to complete the designated action tasks within a

certain time limit. This parameter can be appropriately adjusted based on the difficulty level of the designed action tasks and the optimal loss value obtained from previous model training. At the same time, make sure to leave sufficient time for the model to complete the reasoning.

“--dataset.num_episodes”: Model inference verification refers to the number of data groups to be recorded. You can adjust it according to your actual needs.

Now, if after running the above command, you can see an interface similar to the one used for creating the dataset on your monitor, and your follower robotic arm moves and completes the action tasks you designed under the control of the model, then congratulations! You have initially mastered the application of the Lerobot project! Next, you can try to refer to the above steps to design more complex action tasks and conduct training. Welcome you to join our community of robot development enthusiasts and share your wonderful development experiences! See you in our community!

Peter Smith

January 23, 2026