How LeRobot Hackathon Winners Built AI-Robots With NVIDIA Isaac GR00T

NVIDIA Omniverse
49K subscribers

Subscribe

145 | Share | Ask | Save

4,426 views Streamed live on Jul 31, 2025

Join us for a special livestream featuring two winning teams from the LeRobot Worldwide Hackathon: Spice Terminator and LeDetective DaVinci. See how these teams used GR00T N1.5 and the SO-101 robot to build AI-powered systems that understand natural language and perform complex real-world tasks. From robotic cleaning driven by vision-language-action models to AI agents solving mysteries like a detective, this session is full of innovation, creativity, and real-world applications of physical AI.

Vision Language Action Models for Lerobot

Who are we?

Sanan Garayev
MS in Robotics and AI @UCL
Former AI Engineer @Baykar Technologies

Ton Hoang (Bill) Nguyen
Contributor @Google Deepmind
Software & AI Engineer @Hammer Missions

Pulkit Gera
Applied Scientist Intern @Flawless

Atharva Deshmukh
Research Engineer @Epic Games

William W. Whatley
Fractional CTO / CPO @Catalyst Technologies

What did we do?

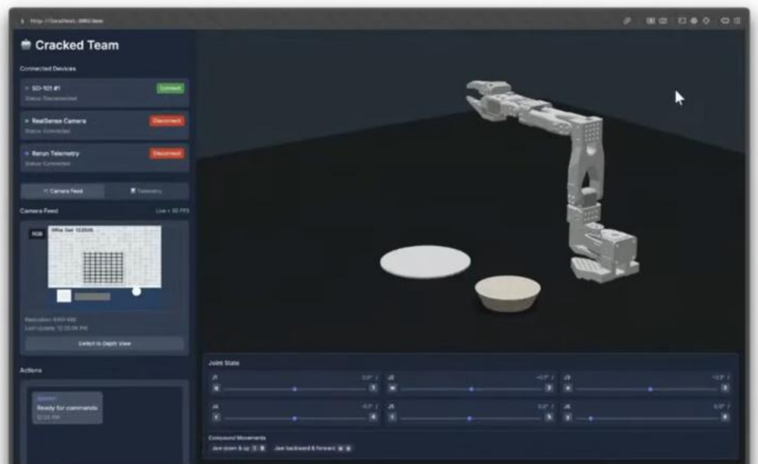
🏆 Novelty Price in the local **Embodied AI Hackathon** organized by **Society for Technological Advancement (SoTA)** in London

🏆 28th place out of 250 submissions in the **Global Lerobot Hackathon** organized by **HuggingFace**

We achieved these with our **Spice Terminator** 🦾 and Web UI.

Spice Terminator

Web User Interface

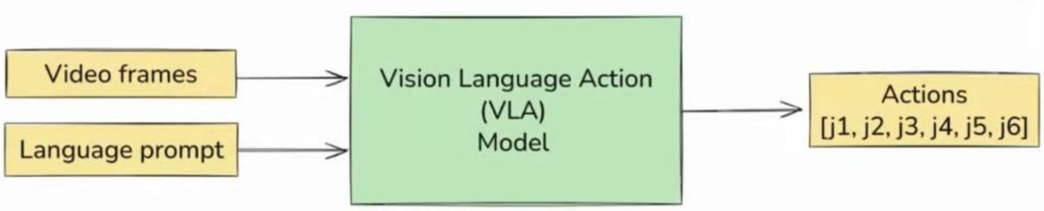


Overview

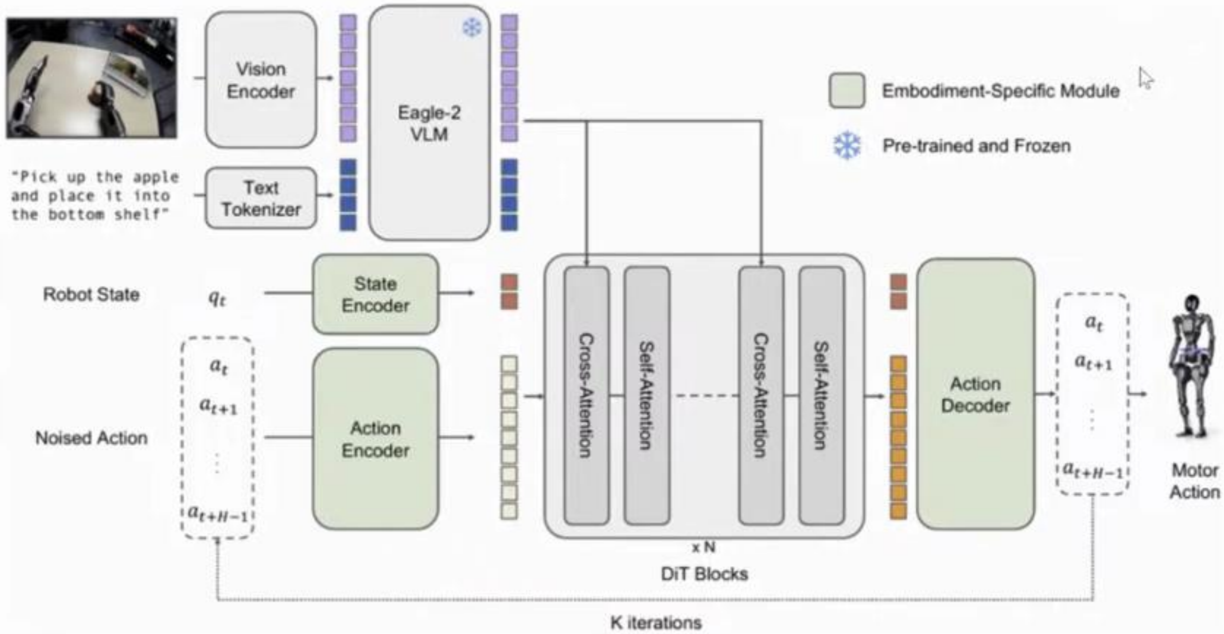
1. Introduction to Vision Language Action (VLA) Models
2. SO101 Arm and Lerobot configurations
3. Experiment setup and dataset collection process
4. Model fine-tuning workflow
5. Running inference on the real robot

Vision Language Action (VLA) Models

Models that combine visual understanding, natural language processing, and action generation to control physical agents (robots) from high-level instructions.

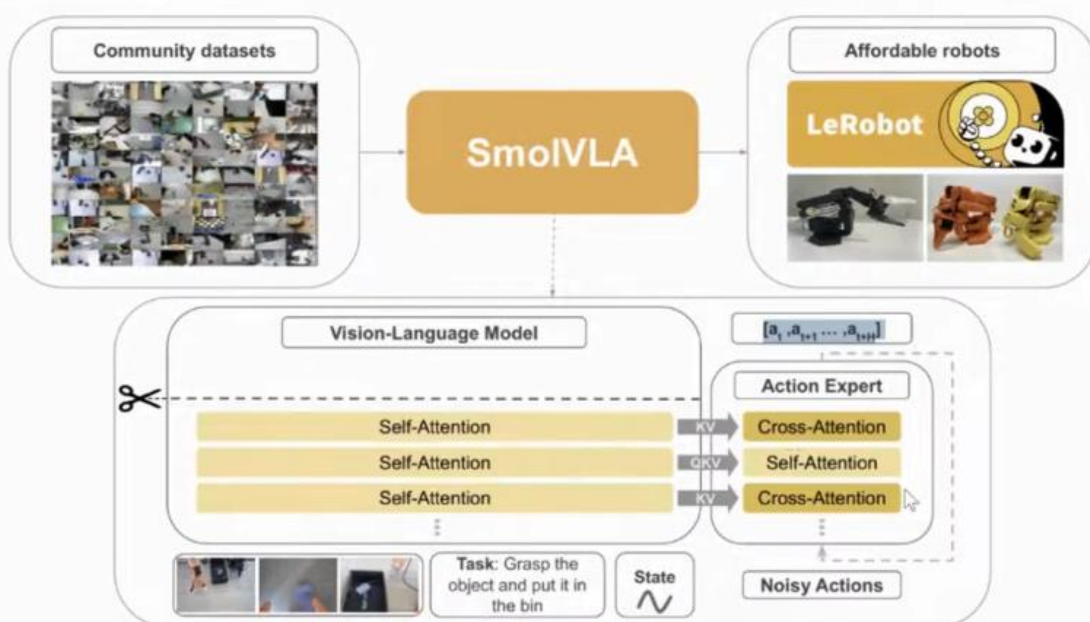


GR00T N1 Model Architecture



NVIDIA et al. (2025). GR00T N1: An open foundation model for generalist humanoid robots.

SmolVLA Model Architecture



Shukor, M., et al. (2025). SmolVLA: A vision-language-action model for affordable and efficient robotics.

VLA Model Comparison

Feature	SmolVLA	GR00T N1.5
Goal	Lightweight, fast	High-capacity, generalist
Model Size	Small (edge-friendly) 450M	Medium/Large 3B
Use Case	Simple table-top tasks	Complex household tasks

LLM Model Training

01 Pre-training phase

- Learning how to predict next tokens
- Uses large-scale internet data
- Model is trained for so long time
- Regardless of cleaning, the datasets involve too much noise

02 Fine-tuning

- Specializing the model to specific tasks (e.g., question-answering in ChatGPT)
- Clearly collected less noisy data
- Few-shot ability for further fine tuning

03 Reinforcement Learning from Human Feedback

- Reducing the risk of hallucinations
- Human feedback collection
- Further model training on the collected human feedback

VLA Model Training

01 Pre-training phase

- Learning how to predict action tokens
- Uses large-scale internet video data
- Uses large-scale cross-embodiment data
- Model is trained for so long time
- Regardless of cleaning, the datasets involve too much noise

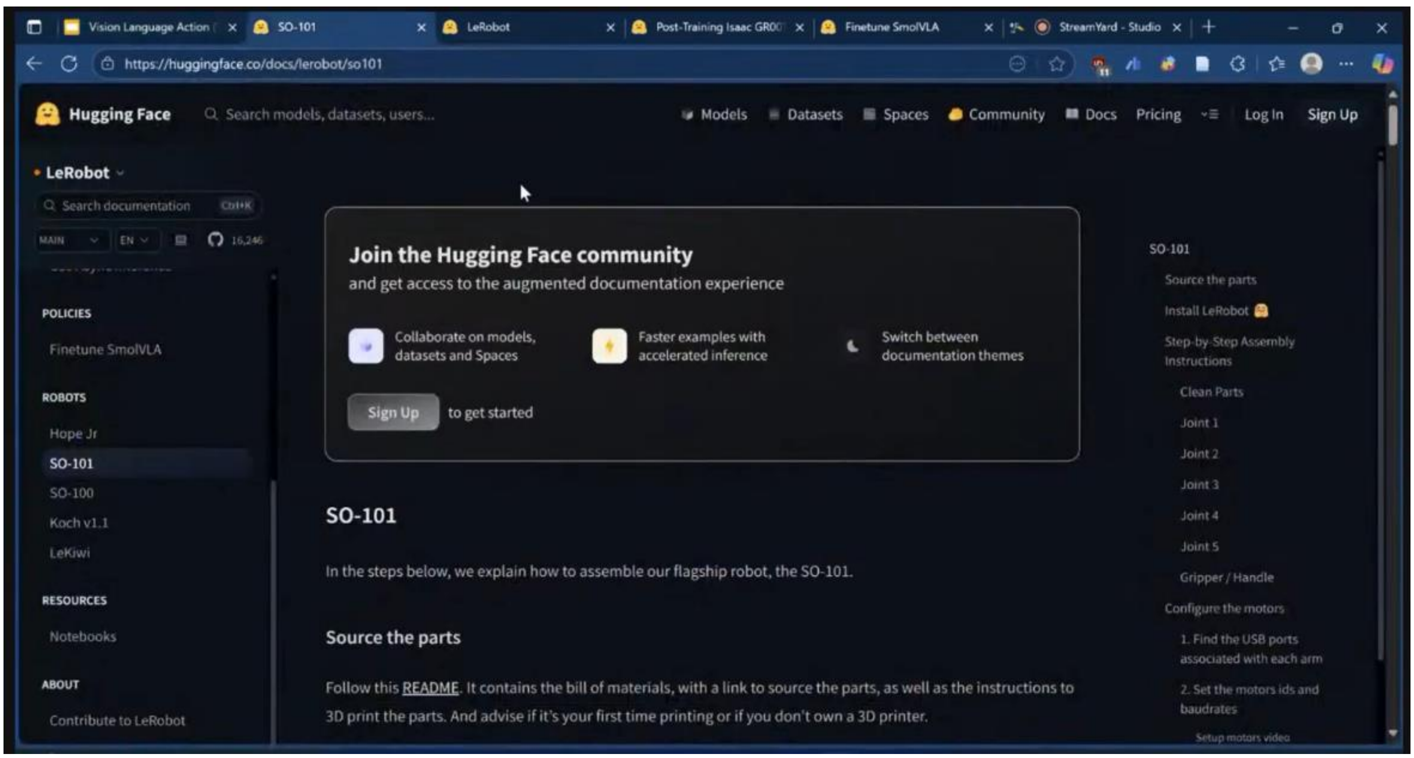
02 Fine-tuning

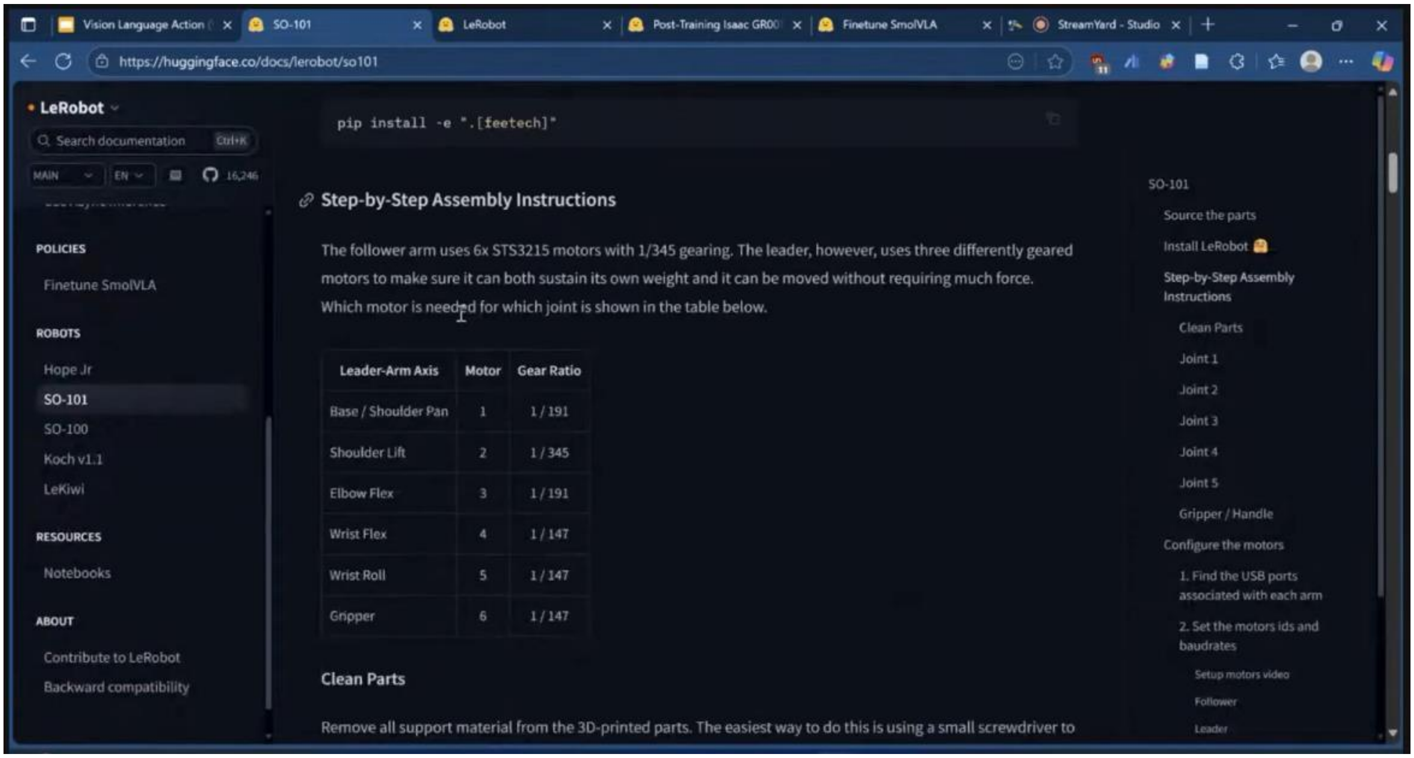
- Specializing the model to specific tasks (e.g., cleaning kitchen, cutting apples, putting fruit to bags)
- Clearly collected less noisy data
- Simulation benchmark datasets to improve the success rates

Section 2: SO101 Arm and Lerobot Environment Configuration

SO101 Arm Configuration

01	Physical setup
02	Motor setup
03	Arm joint calibration
04	Camera setup (dual camera setup)
05	Full teleoperation with camera





https://huggingface.co/docs/lerobot/so101

```
pip install -e ".[feetech]"
```

Step-by-Step Assembly Instructions

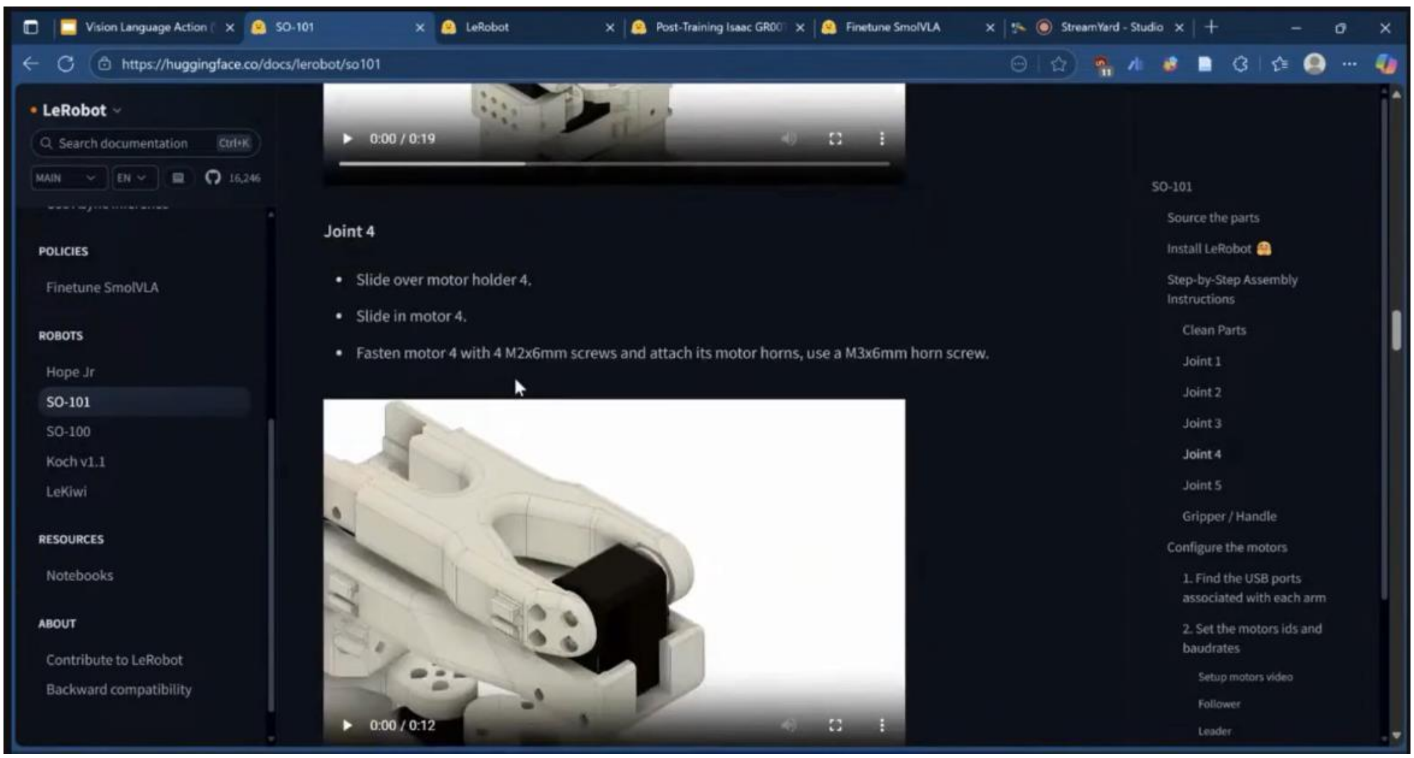
The follower arm uses 6x STS3215 motors with 1/345 gearing. The leader, however, uses three differently geared motors to make sure it can both sustain its own weight and it can be moved without requiring much force. Which motor is needed for which joint is shown in the table below.

Leader-Arm Axis	Motor	Gear Ratio
Base / Shoulder Pan	1	1 / 191
Shoulder Lift	2	1 / 345
Elbow Flex	3	1 / 191
Wrist Flex	4	1 / 147
Wrist Roll	5	1 / 147
Gripper	6	1 / 147

Clean Parts

Remove all support material from the 3D-printed parts. The easiest way to do this is using a small screwdriver to

SO-101
Source the parts
Install LeRobot
Step-by-Step Assembly Instructions
Clean Parts
Joint 1
Joint 2
Joint 3
Joint 4
Joint 5
Gripper / Handle
Configure the motors
1. Find the USB ports associated with each arm
2. Set the motors ids and baudrates
Setup motors video
Follower
Leader



https://huggingface.co/docs/lerobot/so101

0:00 / 0:19

Joint 4

- Slide over motor holder 4.
- Slide in motor 4.
- Fasten motor 4 with 4 M2x6mm screws and attach its motor horns, use a M3x6mm horn screw.

0:00 / 0:12

Configure the motors

1. Find the USB ports associated with each arm

To find the port for each bus servo adapter, connect MotorBus to your computer via USB and power. Run the following script and disconnect the MotorBus when prompted:

```
python -m lerobot.find_port
```

Mac Linux

Example output:

```
Finding all available ports for the MotorBus.
['/dev/tty.usbmodem575E0032081', '/dev/tty.usbmodem575E0031751']
Remove the USB cable from your MotorsBus and press Enter when done.

[...Disconnect corresponding leader or follower arm and press Enter...]
```


Example output:

```
Finding all available ports for the MotorBus.
['/dev/tty.usbmodem575E0032081', '/dev/tty.usbmodem575E0031751']
Remove the USB cable from your MotorsBus and press Enter when done.

[...Disconnect corresponding leader or follower arm and press Enter...]
```

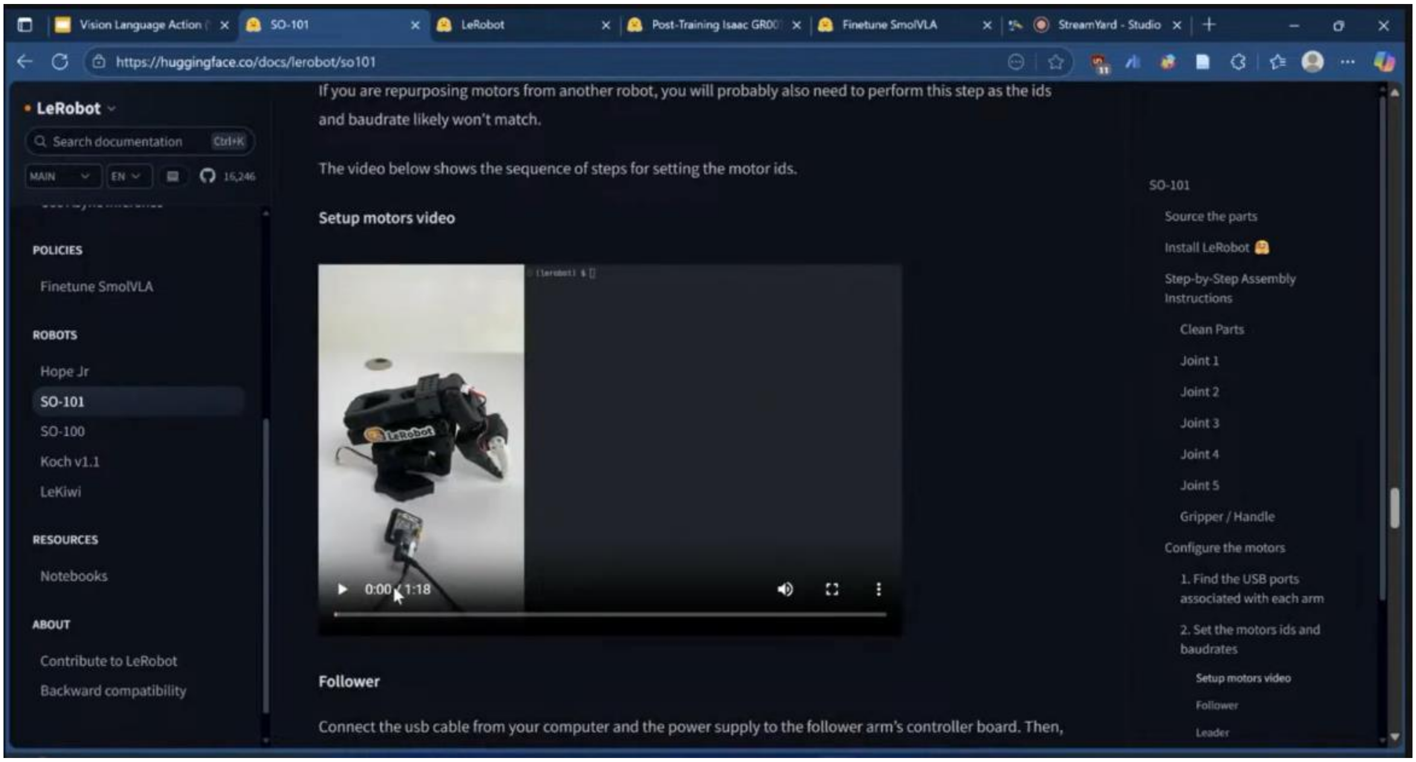

```
The port of this MotorsBus is /dev/tty.usbmodem575E0032081
Reconnect the USB cable.
```

Where the found port is: `/dev/tty.usbmodem575E0032081` corresponding to your leader or follower arm.

2. Set the motors ids and baudrates

Each motor is identified by a unique id on the bus. When brand new, motors usually come with a default id of `1`. For the communication to work properly between the motors and the controller, we first need to set a unique, different id to each motor. Additionally, the speed at which data is transmitted on the bus is determined by the baudrate. In order to talk to each other, the controller and all the motors need to be configured with the same baudrate.

To that end, we first need to connect to each motor individually with the controller in order to set these. Since we will write these parameters in the non-volatile section of the motors' internal memory (EEPROM), we'll only need to do this once.



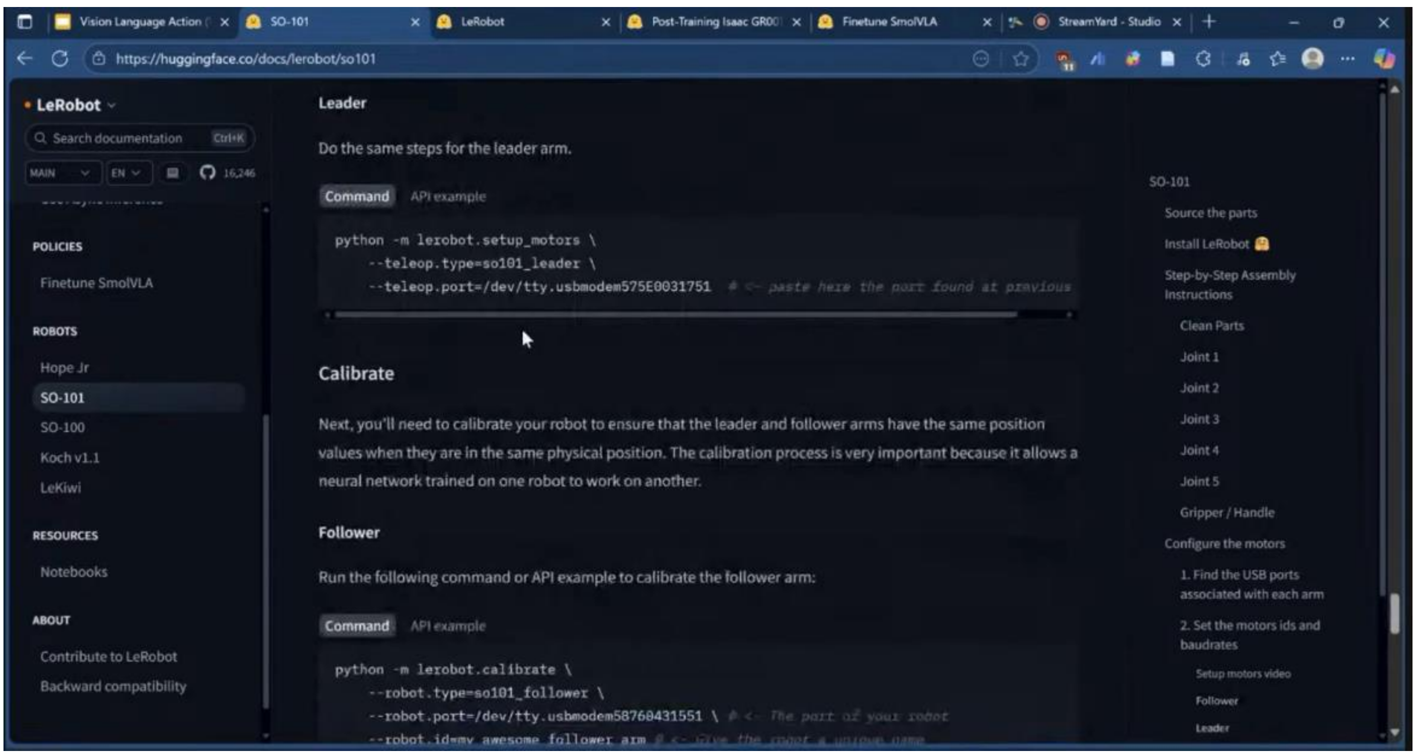
If you are repurposing motors from another robot, you will probably also need to perform this step as the ids and baudrate likely won't match.

The video below shows the sequence of steps for setting the motor ids.

Setup motors video

Follower

Connect the usb cable from your computer and the power supply to the follower arm's controller board. Then,



Leader

Do the same steps for the leader arm.

Command | API example

```
python -m lerobot.setup_motors \
    --teleop.type=so101_leader \
    --teleop.port=/dev/tty.usbmodem575E0031751 # <- paste here the port found at previous
```

Calibrate

Next, you'll need to calibrate your robot to ensure that the leader and follower arms have the same position values when they are in the same physical position. The calibration process is very important because it allows a neural network trained on one robot to work on another.

Follower

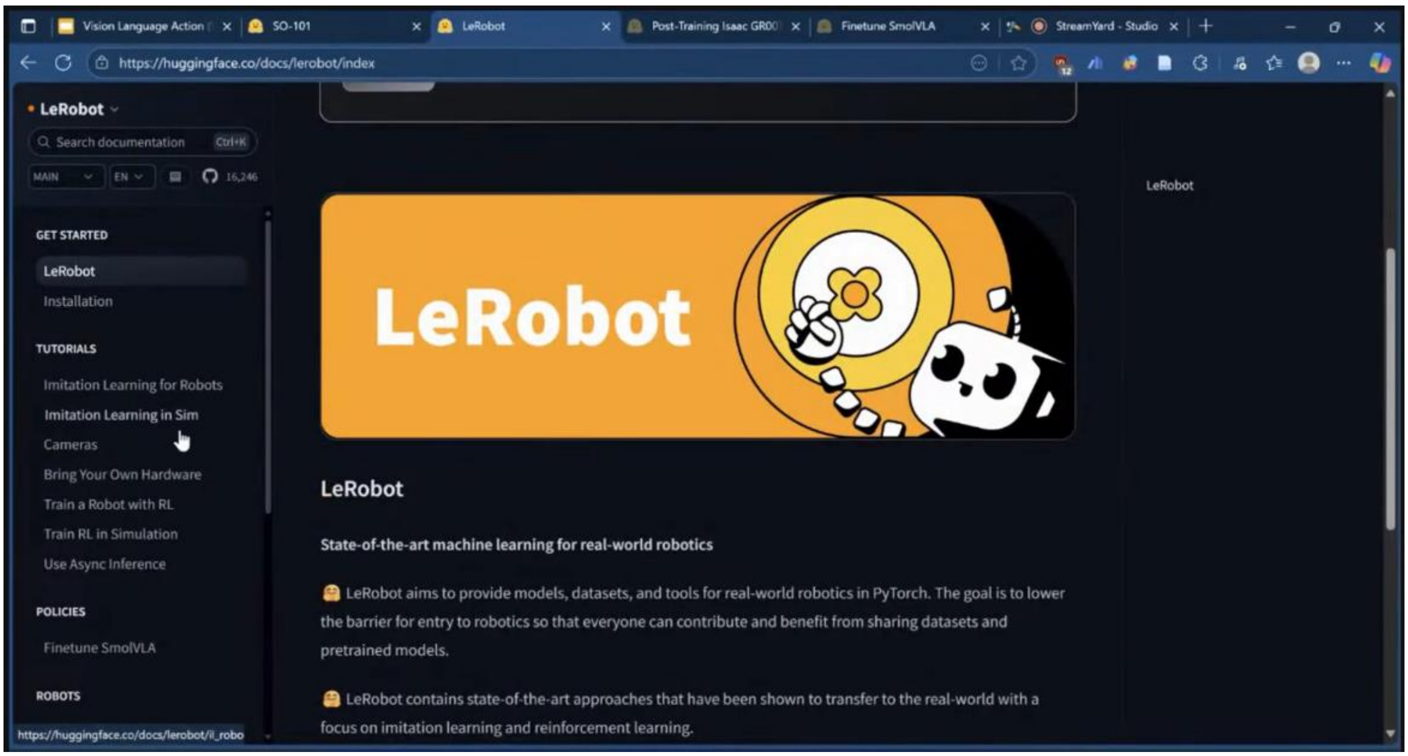
Run the following command or API example to calibrate the follower arm:

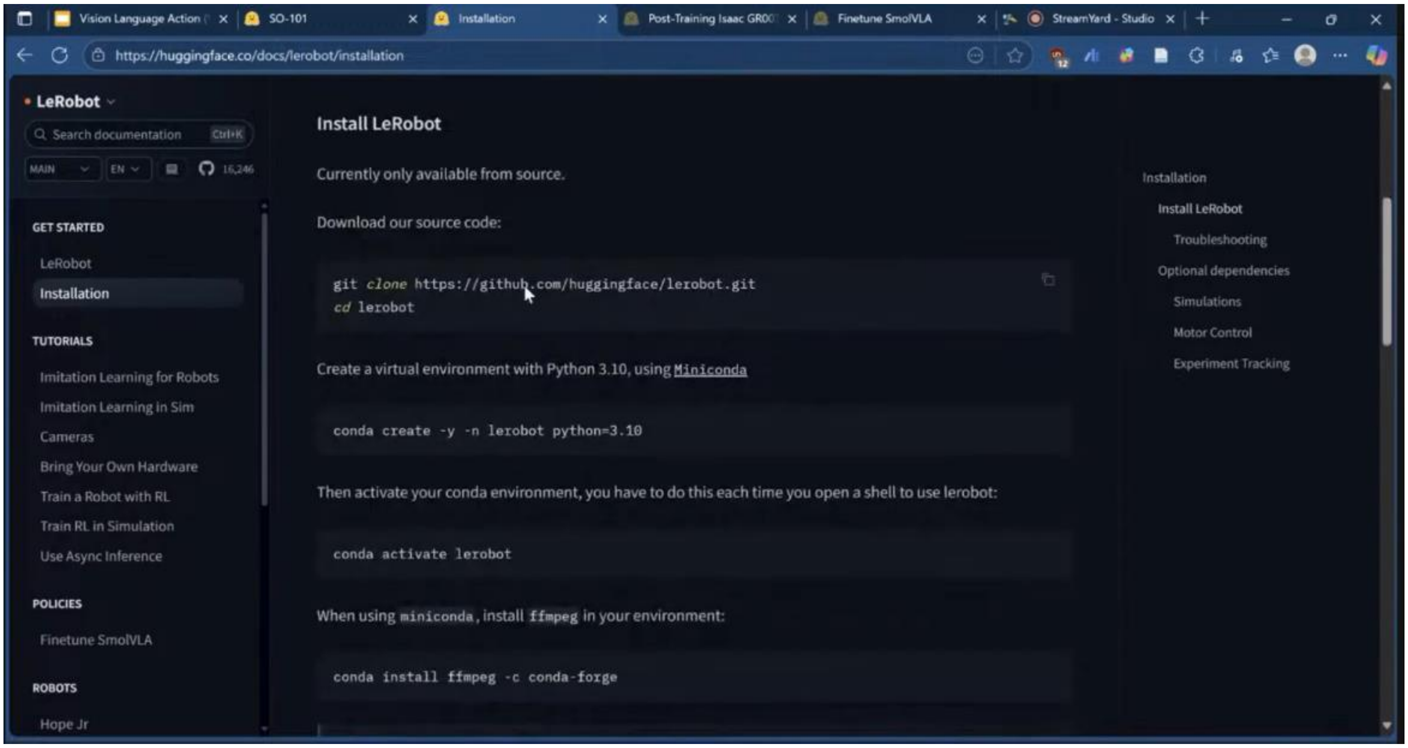
Command | API example

```
python -m lerobot.calibrate \
    --robot.type=so101_follower \
    --robot.port=/dev/tty.usbmodem58760431551 \ # <- The port of your robot
    --robot.id=my_awesome_follower_arm # <- Give the robot a unique name
```

Lerobot Configuration

01	Conda installation
02	Lerobot installation/pip install .
03	User interface arrangement (ffmpeg, rerun sdk)
04	Login to wandb
05	Install smolVLA (optional)





Install LeRobot

Currently only available from source.

Download our source code:

```
git clone https://github.com/huggingface/lerobot.git
cd lerobot
```

Create a virtual environment with Python 3.10, using Miniconda

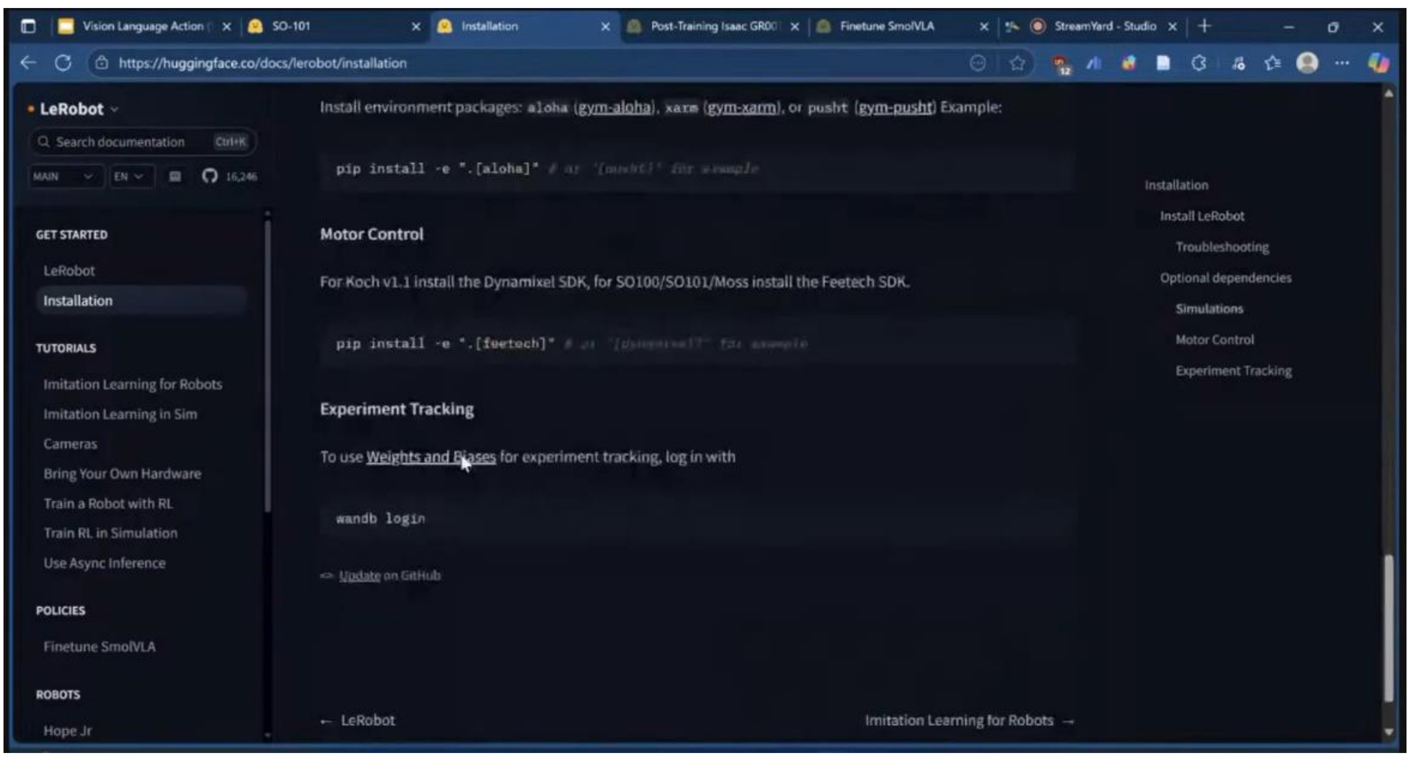
```
conda create -y -n lerobot python=3.10
```

Then activate your conda environment, you have to do this each time you open a shell to use lerobot:

```
conda activate lerobot
```

When using `miniconda`, install `ffmpeg` in your environment:

```
conda install ffmpeg -c conda-forge
```



Install environment packages: `aloha` (gym-aloha), `xarm` (gym-xarm), or `pusht` (gym-pusht) Example:

```
pip install -e ".[aloha]"
```

Motor Control

For Koch v1.1 install the Dynamixel SDK, for SO100/SO101/Moss install the Feetech SDK.

```
pip install -e ".[feetech]"
```

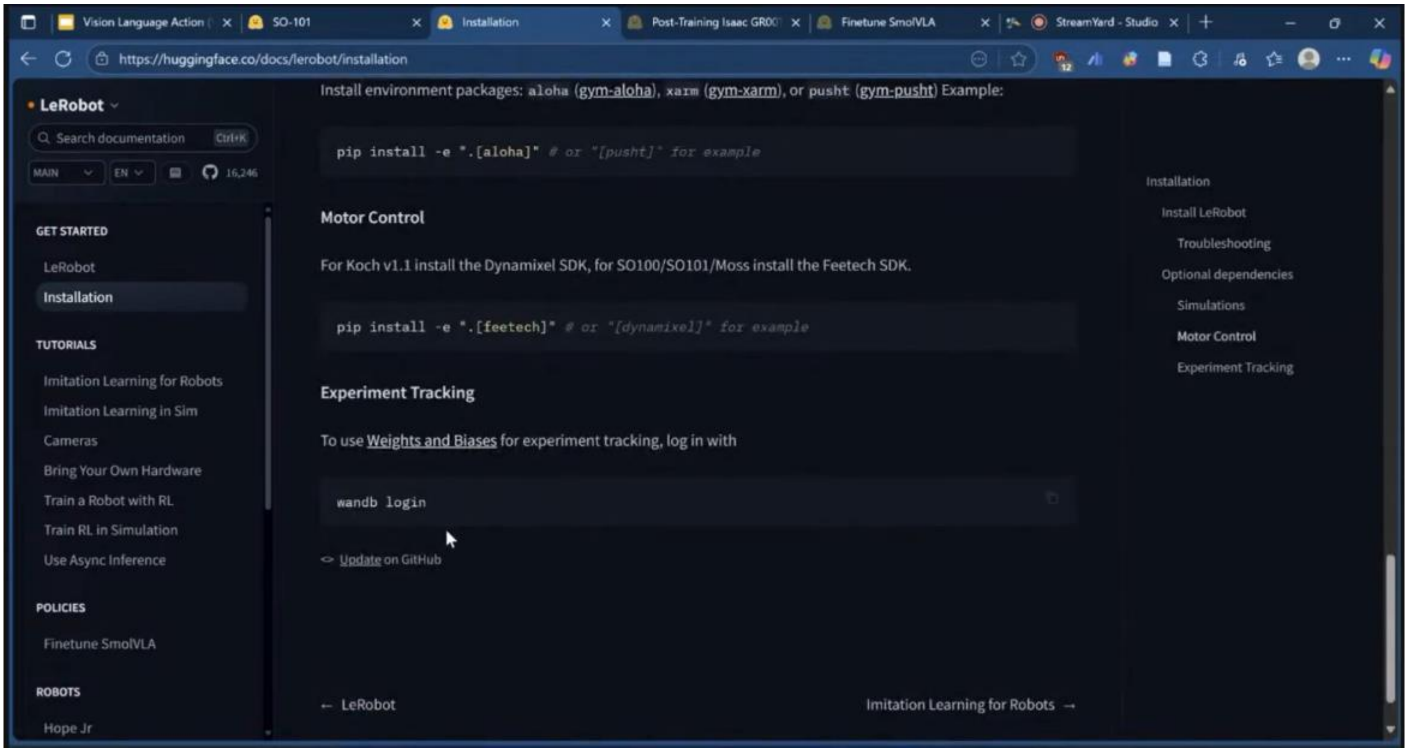
Experiment Tracking

To use Weights and Biases for experiment tracking, log in with

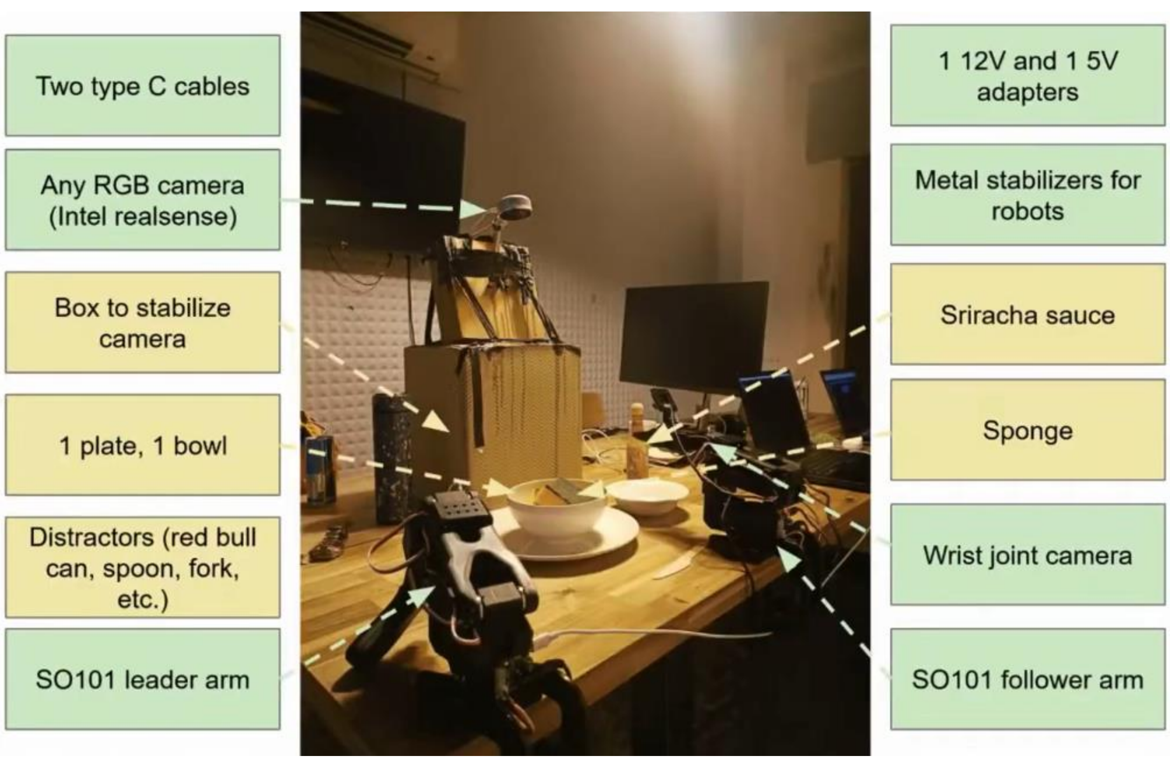
```
wandb login
```

Update on GitHub

← LeRobot   Imitation Learning for Robots →



Section 3: Experiment setup and dataset collection process



Data Collection - Core Task

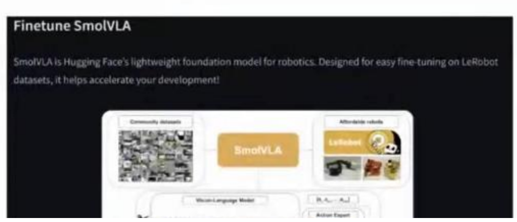
- Collect episodes using the leader arm.
- We collect 75 episodes. Each episode is 25 seconds long
- Prompt: "Wipe red sauce from the plate using sponge."
- Create lot of examples with obstructions for generalization
- Tip: Calibrate robot arm carefully
- Tip: Easier to control with left arm
- Prepared using phosphobot
- Dataset Link

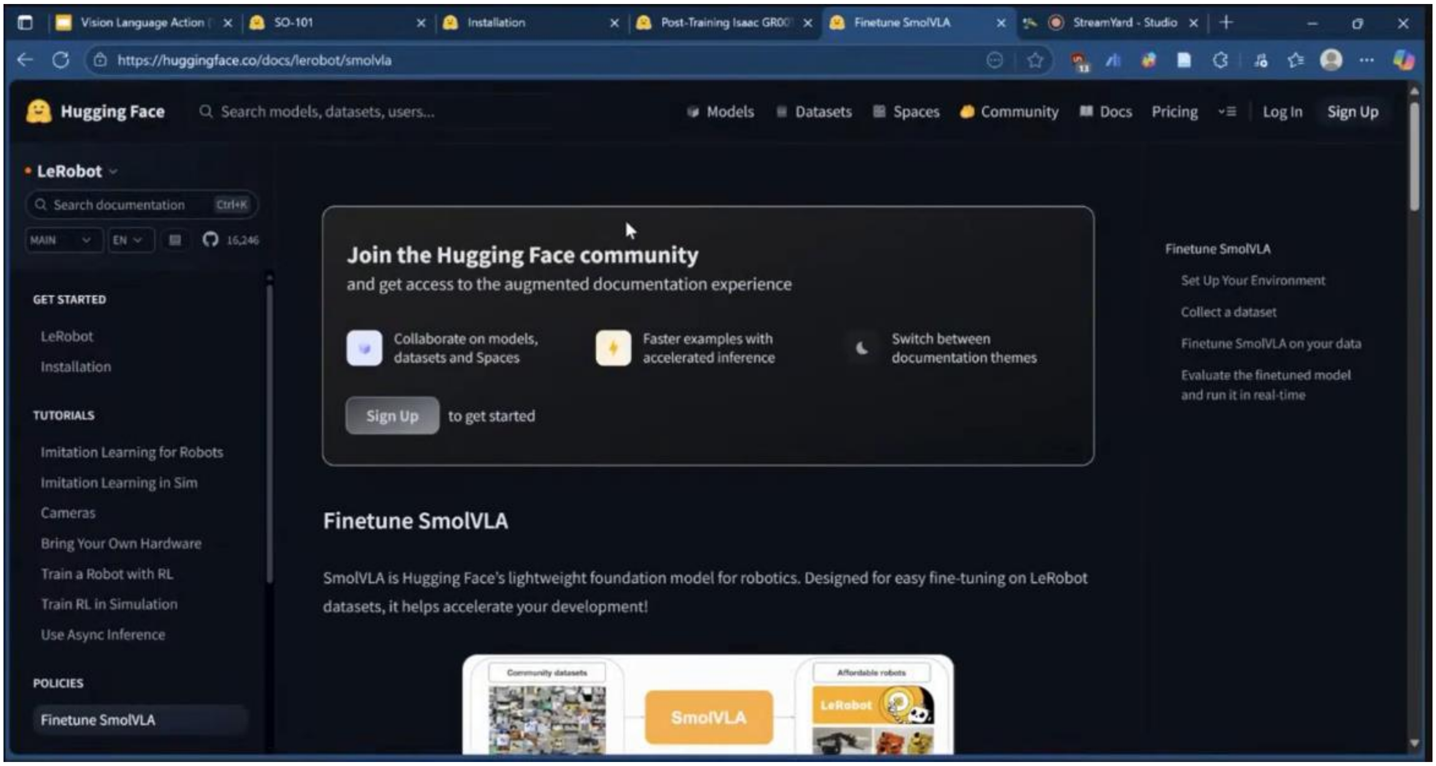
Data Collection - Distributed Samples

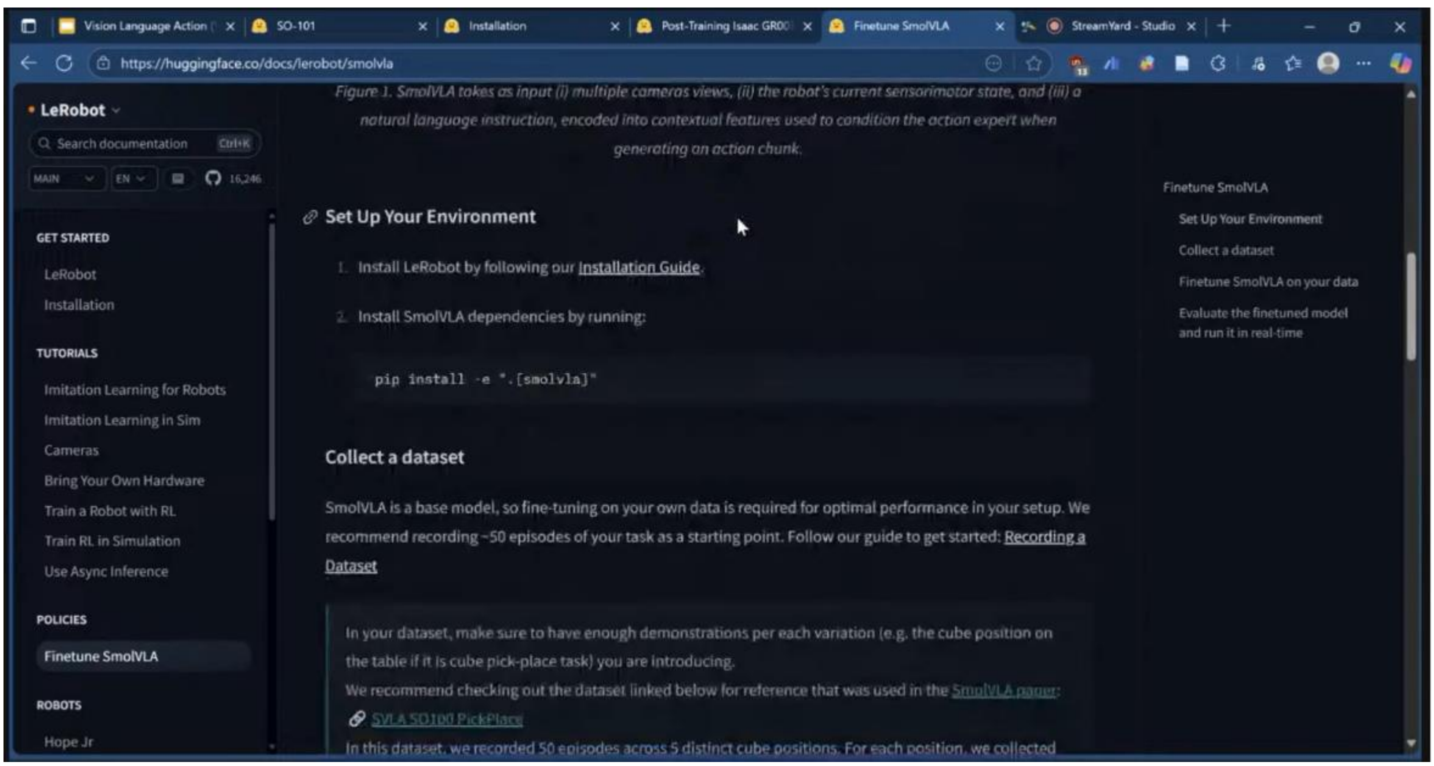
Section 4: Model fine-tuning workflow

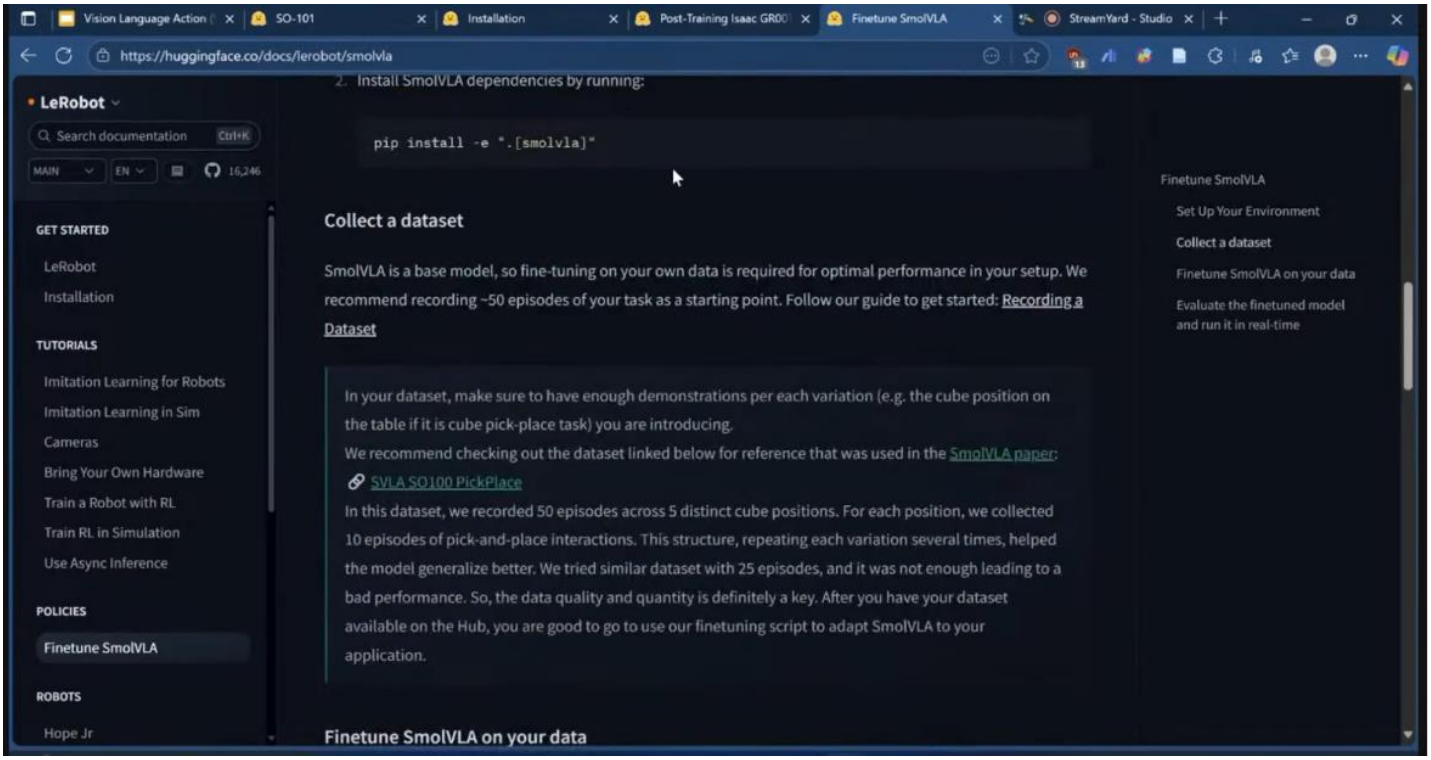
SmolVLA fine-tuning

- 400-500 mB VRAM space needed
- Used default settings (LORA adapter, freezed VLM)
- Worked at commercial laptop with RTX4060
- Took 3-4 hours for 25 samples
- Checkpoints are saved in each 5000 steps
- Total of 20000 steps are trained




Vision Language Action
SO-101
Installation
Post-Training Isaac GR00
Finetune SmolVLA
StreamYard - Studio
https://huggingface.co/docs/lerobot/smolvla
Hugging Face
Search models, datasets, users...
Models
Datasets
Spaces
Community
Docs
Pricing
Log In
Sign Up
LeRobot
Search documentation
Ctrl+K
MAIN
EN
16,246
GET STARTED
LeRobot
Installation
TUTORIALS
Imitation Learning for Robots
Imitation Learning in Sim
Cameras
Bring Your Own Hardware
Train a Robot with RL
Train RL in Simulation
Use Async Inference
POLICIES
Finetune SmolVLA
Join the Hugging Face community
and get access to the augmented documentation experience
Collaborate on models, datasets and Spaces
Faster examples with accelerated inference
Switch between documentation themes
Sign Up
to get started
Finetune SmolVLA
SmolVLA is Hugging Face's lightweight foundation model for robotics. Designed for easy fine-tuning on LeRobot datasets, it helps accelerate your development!
Community datasets
SmolVLA
Affordable robots
LeRobot
Finetune SmolVLA
Set Up Your Environment
Collect a dataset
Finetune SmolVLA on your data
Evaluate the finetuned model and run it in real-time


Vision Language Action
SO-101
Installation
Post-Training Isaac GR00
Finetune SmolVLA
StreamYard - Studio
https://huggingface.co/docs/lerobot/smolvla
LeRobot
Search documentation
Ctrl+K
MAIN
EN
16,246
GET STARTED
LeRobot
Installation
TUTORIALS
Imitation Learning for Robots
Imitation Learning in Sim
Cameras
Bring Your Own Hardware
Train a Robot with RL
Train RL in Simulation
Use Async Inference
POLICIES
Finetune SmolVLA
ROBOTS
Hope Jr
Figure 1. SmolVLA takes as input (i) multiple cameras views, (ii) the robot's current sensorimotor state, and (iii) a natural language instruction, encoded into contextual features used to condition the action expert when generating an action chunk.
Set Up Your Environment
1. Install LeRobot by following our Installation Guide.
2. Install SmolVLA dependencies by running:
pip install -e ".[smolvla]"
Collect a dataset
SmolVLA is a base model, so fine-tuning on your own data is required for optimal performance in your setup. We recommend recording ~50 episodes of your task as a starting point. Follow our guide to get started: Recording a Dataset
In your dataset, make sure to have enough demonstrations per each variation (e.g. the cube position on the table if it is cube pick-place task) you are introducing.
We recommend checking out the dataset linked below for reference that was used in the SmolVLA paper:
SVLA SO100 PickPlace
In this dataset, we recorded 50 episodes across 5 distinct cube positions. For each position, we collected
Finetune SmolVLA
Set Up Your Environment
Collect a dataset
Finetune SmolVLA on your data
Evaluate the finetuned model and run it in real-time



2. Install SmolVLA dependencies by running:

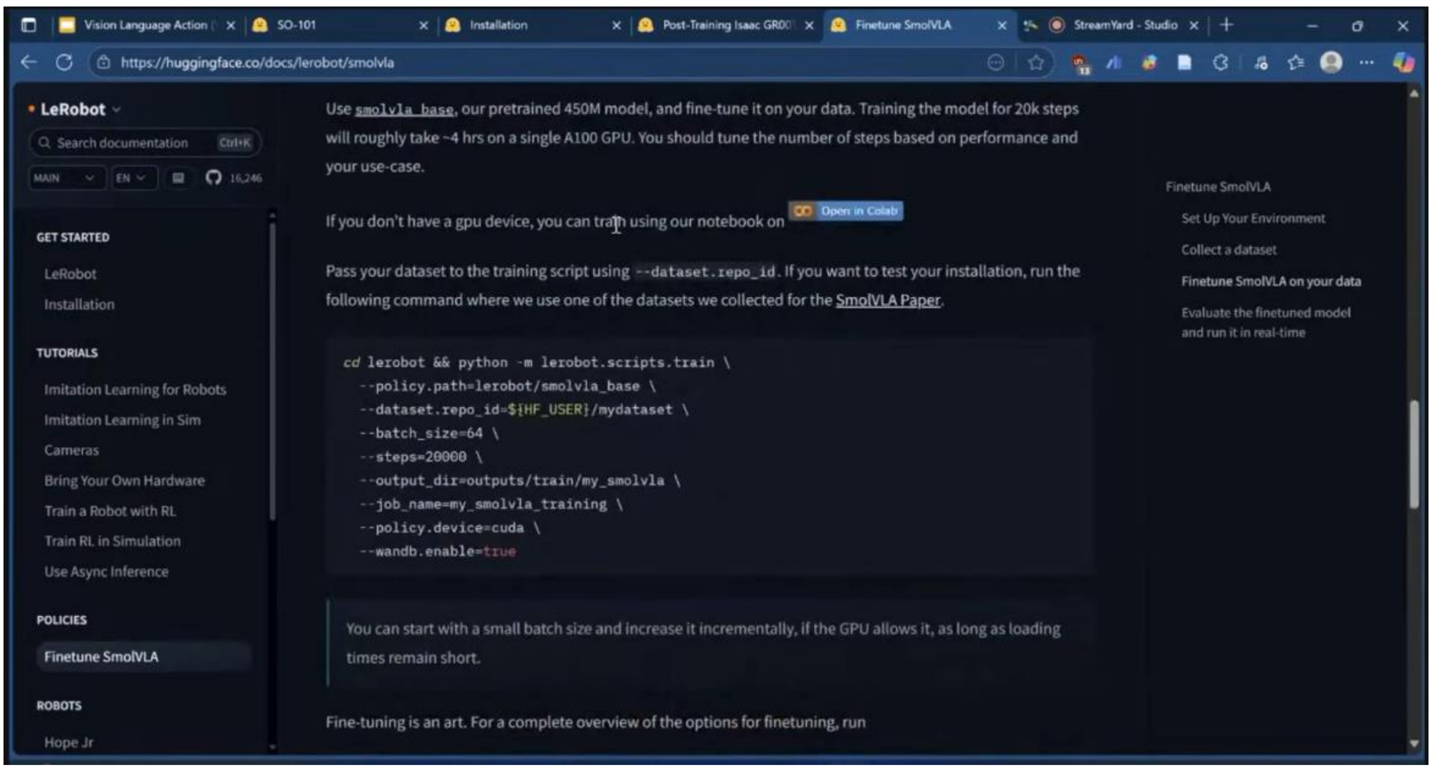
```
pip install -e ".[smolvla]"
```

Collect a dataset

SmolVLA is a base model, so fine-tuning on your own data is required for optimal performance in your setup. We recommend recording ~50 episodes of your task as a starting point. Follow our guide to get started: Recording a Dataset

> In your dataset, make sure to have enough demonstrations per each variation (e.g. the cube position on the table if it is cube pick-place task) you are introducing.
> We recommend checking out the dataset linked below for reference that was used in the SmolVLA paper:
> 🔗 SVLA SO100 PickPlace
> In this dataset, we recorded 50 episodes across 5 distinct cube positions. For each position, we collected 10 episodes of pick-and-place interactions. This structure, repeating each variation several times, helped the model generalize better. We tried similar dataset with 25 episodes, and it was not enough leading to a bad performance. So, the data quality and quantity is definitely a key. After you have your dataset available on the Hub, you are good to go to use our finetuning script to adapt SmolVLA to your application.

Finetune SmolVLA on your data



Use `smolvla_base`, our pretrained 450M model, and fine-tune it on your data. Training the model for 20k steps will roughly take ~4 hrs on a single A100 GPU. You should tune the number of steps based on performance and your use-case.

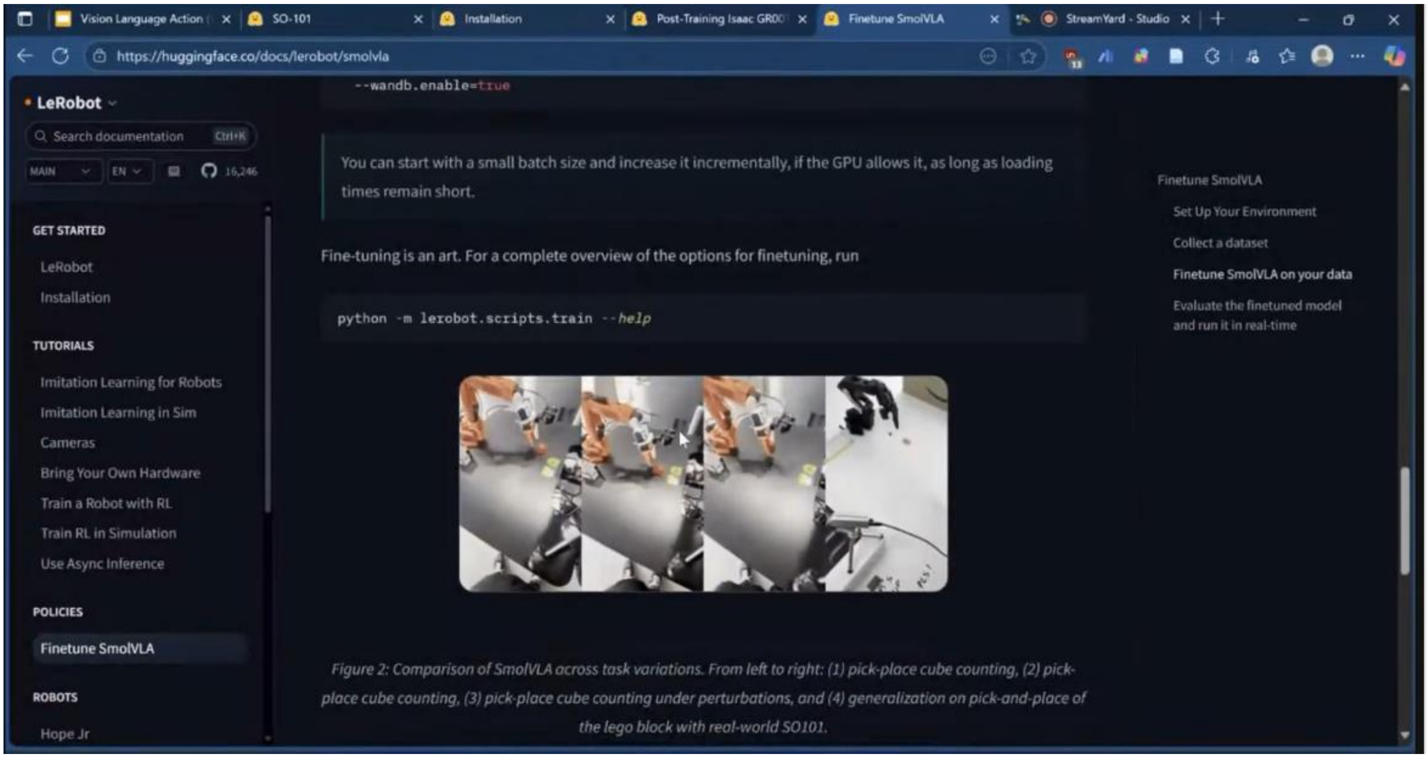
If you don't have a gpu device, you can train using our notebook on Open in Colab

Pass your dataset to the training script using `--dataset.repo_id`. If you want to test your installation, run the following command where we use one of the datasets we collected for the SmolVLA Paper.

```
cd lerobot && python -m lerobot.scripts.train \
  --policy.path=lerobot/smolvla_base \
  --dataset.repo_id=${HF_USER}/mydataset \
  --batch_size=64 \
  --steps=20000 \
  --output_dir=outputs/train/my_smolvla \
  --job_name=my_smolvla_training \
  --policy.device=cuda \
  --wandb.enable=true
```

> You can start with a small batch size and increase it incrementally, if the GPU allows it, as long as loading times remain short.

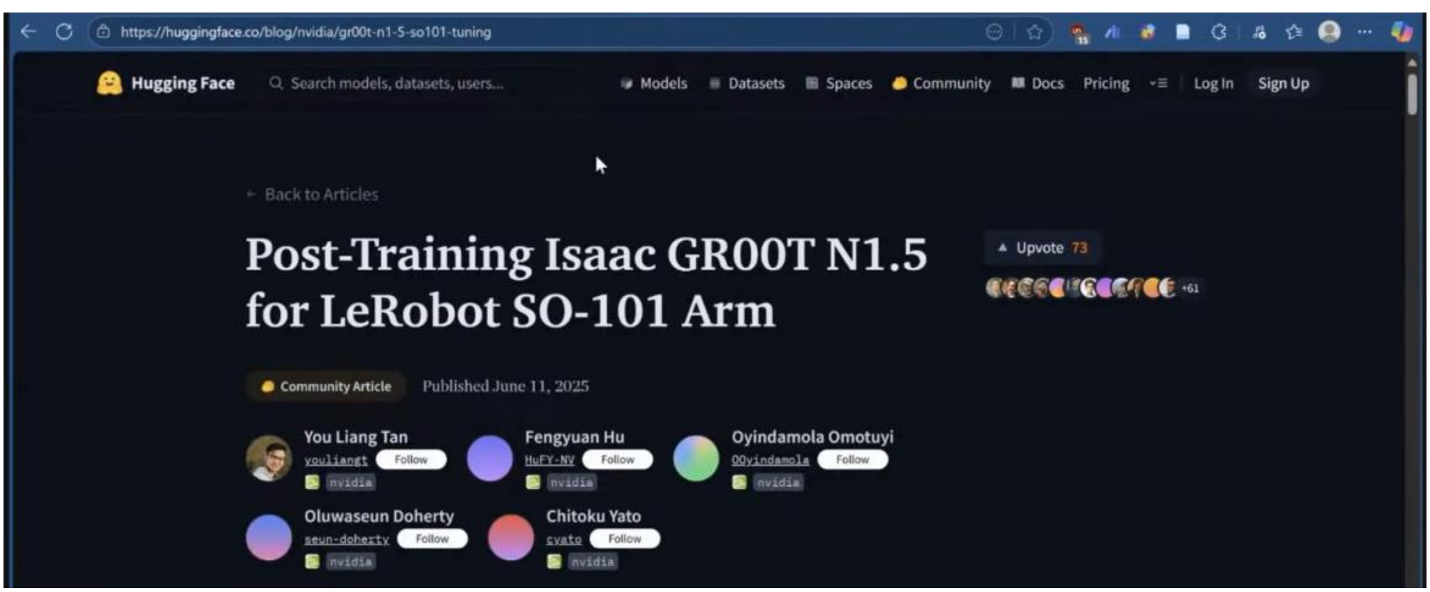
Fine-tuning is an art. For a complete overview of the options for finetuning, run

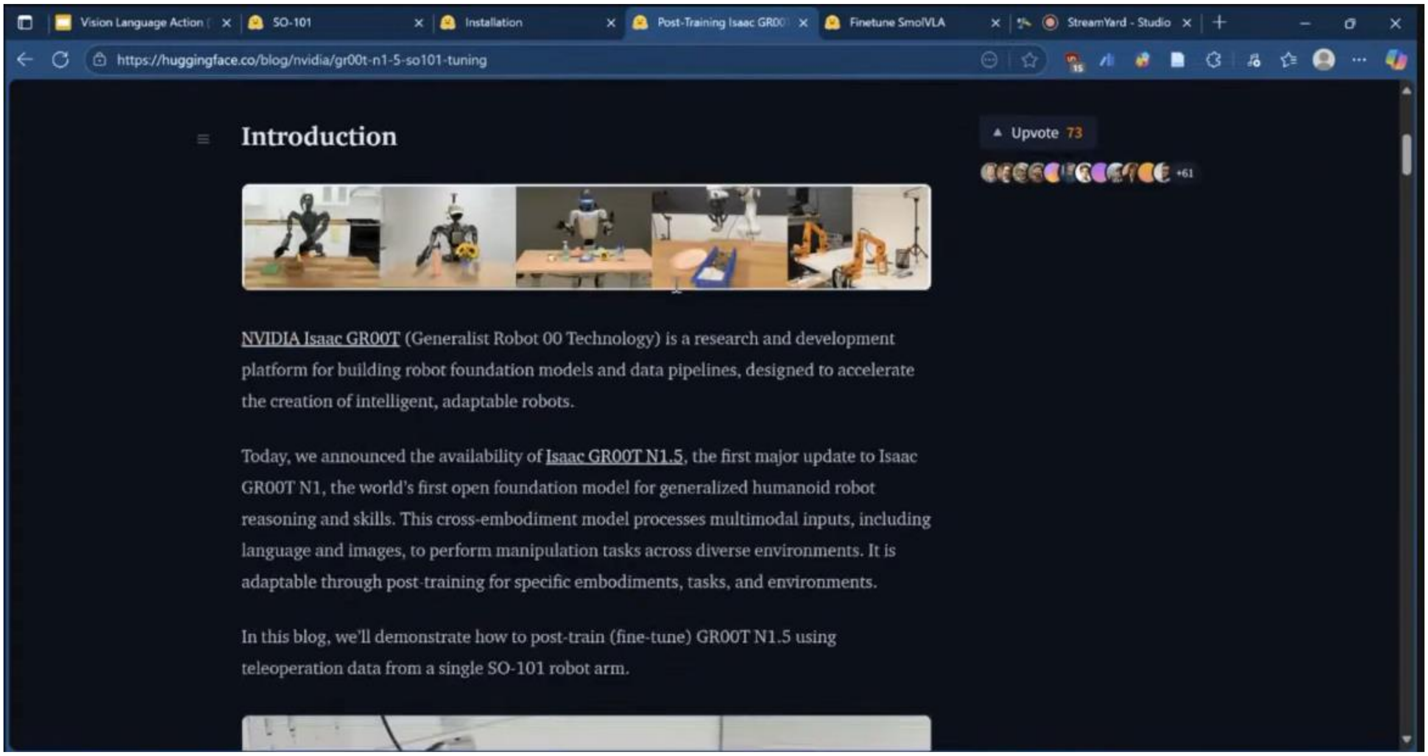


GR00T-N1.5 Model Fine Tuning

- ~20GB VRAM needed!
- Ori provided free credits to train the model in their clusters
- Took 1 hour with H100 GPU
- Used default settings, LORA adapter
- Used new embodiment setting
- Model trained on 20000 steps
- Checkpoints are saved in each 1000 step

1 skill for less than 3 dollars





Vision Language Action | SO-101 | Installation | Post-Training Isaac GR00 | Finetune SmolVLA | StreamYard - Studio

https://huggingface.co/blog/nvidia/gr00t-n1-5-so101-tuning

Upvote 73

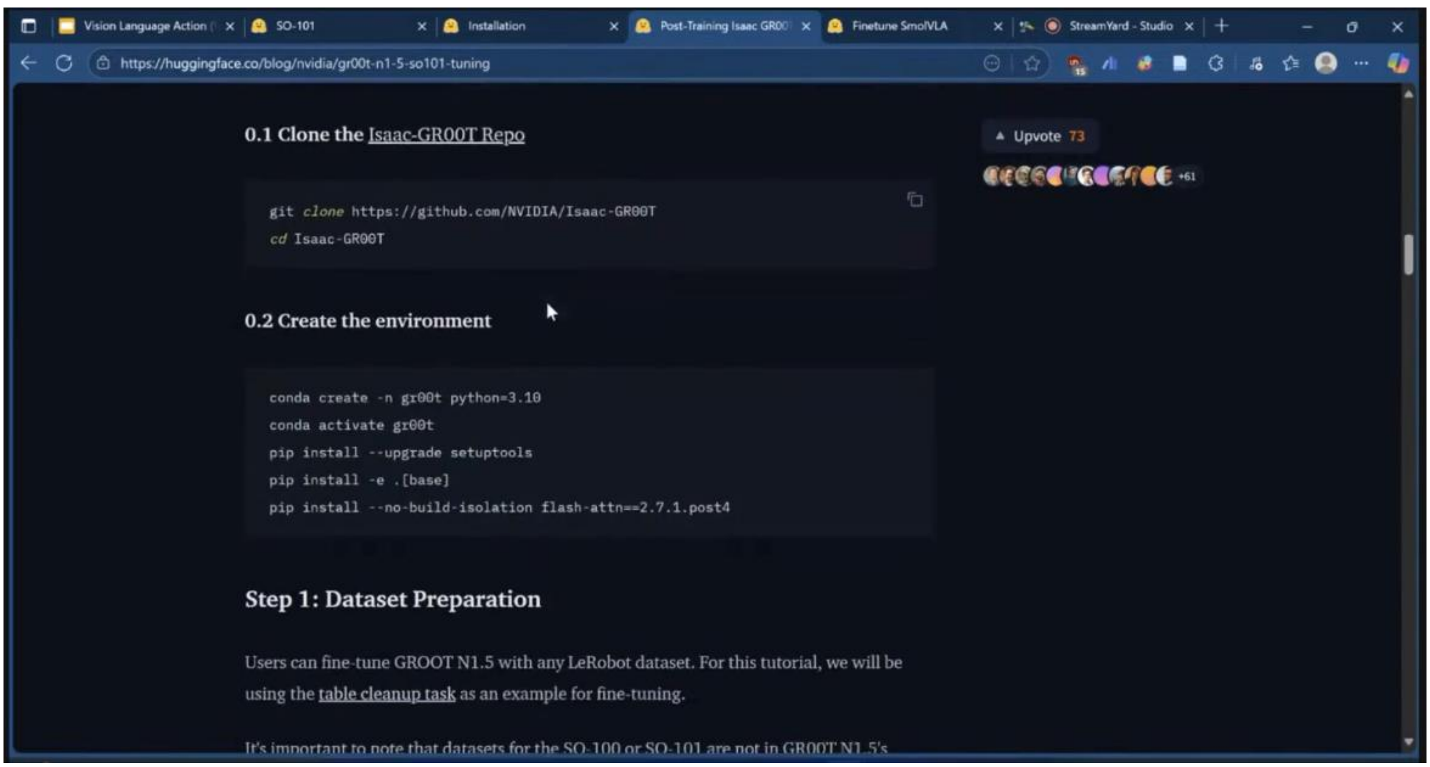
+61

Introduction

[NVIDIA Isaac GR00T](#) (Generalist Robot 00 Technology) is a research and development platform for building robot foundation models and data pipelines, designed to accelerate the creation of intelligent, adaptable robots.

Today, we announced the availability of [Isaac GR00T N1.5](#), the first major update to Isaac GR00T N1, the world's first open foundation model for generalized humanoid robot reasoning and skills. This cross-embodiment model processes multimodal inputs, including language and images, to perform manipulation tasks across diverse environments. It is adaptable through post-training for specific embodiments, tasks, and environments.

In this blog, we'll demonstrate how to post-train (fine-tune) GR00T N1.5 using teleoperation data from a single SO-101 robot arm.



Vision Language Action | SO-101 | Installation | Post-Training Isaac GR00 | Finetune SmolVLA | StreamYard - Studio

https://huggingface.co/blog/nvidia/gr00t-n1-5-so101-tuning

Upvote 73

+61

0.1 Clone the [Isaac-GR00T Repo](#)

```
git clone https://github.com/NVIDIA/Isaac-GR00T
cd Isaac-GR00T
```

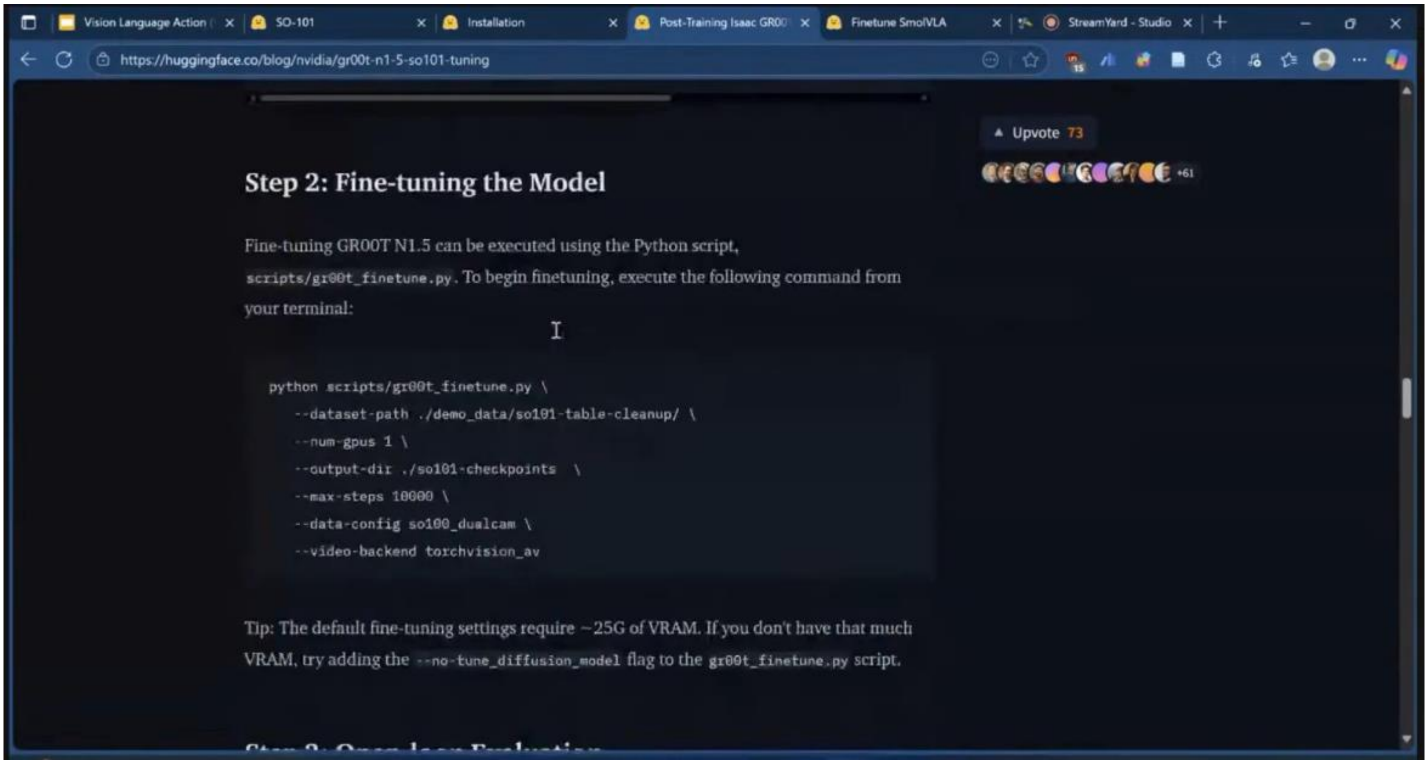
0.2 Create the environment

```
conda create -n gr00t python=3.10
conda activate gr00t
pip install --upgrade setuptools
pip install -e .[base]
pip install --no-build-isolation flash-attn==2.7.1.post4
```

Step 1: Dataset Preparation

Users can fine-tune GROOT N1.5 with any LeRobot dataset. For this tutorial, we will be using the [table cleanup task](#) as an example for fine-tuning.

It's important to note that datasets for the SO-100 or SO-101 are not in GR00T N1.5's



Step 2: Fine-tuning the Model

Fine-tuning GR00T N1.5 can be executed using the Python script, `scripts/gr00t_finetune.py`. To begin finetuning, execute the following command from your terminal:

```
python scripts/gr00t_finetune.py \
   --dataset-path ./demo_data/so101-table-cleanup/ \
   --num-gpus 1 \
   --output-dir ./so101-checkpoints  \
   --max-steps 10000 \
   --data-config so100_dualcam \
   --video-backend torchvision_av
```

Tip: The default fine-tuning settings require ~25G of VRAM. If you don't have that much VRAM, try adding the `--no-tune_diffusion_model` flag to the `gr00t_finetune.py` script.

Section 5: Running inference on the real robot

GR00T-N1.5 Model Inference and Model Evaluation

01	Give cloud computers access to the local ports
02	Run the inference with the latest checkpoint
03	Test different prompts
04	Test the generalization prompts
05	Evaluate the success rate

Cloud-Based GR00T Inference with 16-Action Window

Setup

- Inference Location: Ori Cloud H100 GPU (GR00T model)
- Action Window: 16 predicted actions per inference
- Cloud inference adds ~300–500 ms latency
- 16-step prediction reduces stop-and-wait behavior
- Allows robot to move continuously between cloud updates

Inference for the core task

Wrist joint camera view

Real-time performance of GR00T-N1.5
deployed on SO101 follower

Model's generalization ability

Struggles of Spice Terminator

Async vs Sync Inference (Le Robot + GR00T)

Synchronous

Stop → Think → Act

Robot waits for GR00T prediction before moving

✓ Stable & deterministic

✗ Slow, high latency → jerky if cloud-based

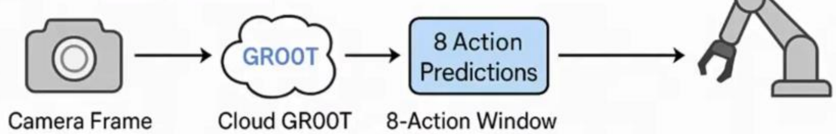
Asynchronous

Keep moving while inference runs in background

✓ Smoother, lower perceived latency

✗ Can act on stale observations, harder to 'ebug

SO-101 Arm Cloud Setup



Section 6: Web UI Integration