## Pi0 - generalist Vision Language Action policy for robots (VLA Series Ep.2)

**Ilia**
3.63K subscribers

350 | Share | Ask | Save

**8,708 views Sep 16, 2025** #Robotics #transformers #AI
The second video in the series about Visual Language Action policies for robotics!

This time, I will focus on one of the most notable open-source VLA models, Pi0 from Physical Intelligence.

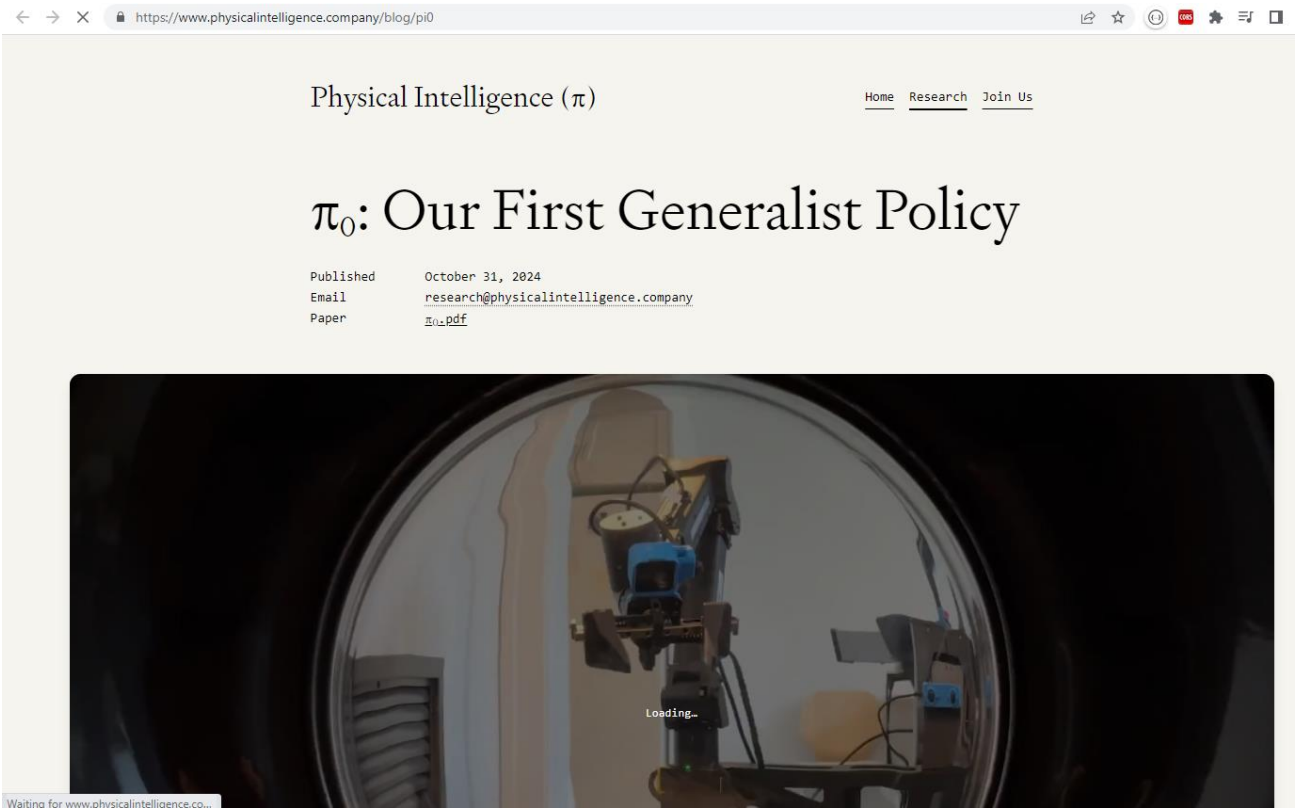
What You'll Learn in This Episode:
The model architecture and the overview
Fine-tuning on your own robot
FAST tokenization vs. Flow Matching
LoRA vs Full fine-tuning
Practical demo with LeKiwi and LeRobot

Coming Up in This Series:
- Pi-0.5 - the more advanced version of Pi 0
- SmolVLA from HuggingFace LeRobot
- Gr00t N1.5 from NVIDIA
- And other related topics

0:00 Intro
3:44 Model Architecture
8:21 Action Expert Details
22:30 Fine-tuning with OpenPI
52:01 Experiments with My Robot
1:14:12 Outro

Series Playlist: • VLA policies in robotics

Resources & Code:
- Pi-0 Blog post: https://www.physicalintelligence.comp...
- Pi-0 Paper: https://www.physicalintelligence.comp...
- OpenPi Github: https://github.com/Physical-Intellige...



https://github.com/Physical-Intelligence/openpi

Platform Solutions Resources Open Source Enterprise Pricing

Search or jump to... Sign in Sign up

Physical-Intelligence / openpi Public

Notifications Fork 1.3k Star 9.4k

Code Issues 178 Pull requests 32 Discussions Actions Projects Security Insights

main 6 Branches 0 Tags Go to file Code

kpertsch Multi-RLDS Datasets (Cotraining) + PolaRiS DROID jointpos policy conf... bd70b8f · 3 days ago 186 Commits

| | | |
|---|---|---|
| .github | Upgrade LeRobot | 7 months ago |
| .vscode | Initial commit | 11 months ago |
| docs | broken link fix | 2 months ago |
| examples | Fix typo in comment for read_camera method | 2 weeks ago |
| packages/openpi-client | feat: add wss support | 2 months ago |
| scripts | Fix typo in comment about sudo requirement | 2 weeks ago |
| src/openpi | fix ruff formatting | last week |
| third_party | Initial commit | 11 months ago |
| .dockerignore | Initial commit | 11 months ago |

About

*No description, website, or topics provided.*

Readme
Apache-2.0 license
Contributing
Activity
Custom properties
9.4k stars
78 watching
1.3k forks
Report repository

Releases

No releases published

Packages

Physical Intelligence (π) Home Research Join Us

physicalintelligence.company

Physical Intelligence is bringing general-purpose AI into the physical world. We are a group of engineers, scientists, roboticists, and company builders developing foundation models and learning algorithms to power the robots of today and the physically-actuated devices of the future.

- **Real-Time Action Chunking with Large Models** June 9, 2025
  A real-time system for large VLAs that maintains precision and speed in the face of high latency.
- **VLAs that Train Fast, Run Fast, and Generalize Better** May 28, 2025
  A method to train vision-language-action models that train quickly, maintain internet-scale knowledge, have high quality inference properties, and generalize well.
- **$\pi_{0.5}$: a VLA with Open-World Generalization** April 22, 2025
  Our latest generalist policy, π0.5, extends π0 and enables open-world generalization. Our new model can control a mobile manipulator to clean up an entirely new kitchen or bedroom.
- **Teaching Robots to Listen and Think Harder** February 26, 2025
  A method for robots to think through complex tasks step by step, incorporating human-in-the-loop feedback.
- **Open Sourcing $\pi_0$** February 4, 2025
  We are releasing the weights and code for π0 as well as our new π0-FAST autoregressive model.
- **FAST: Efficient Robot Action Tokenization** January 16, 2025
  A new robot action tokenizer that allows us to train generalist policies 5x faster than previous models.

Physical Intelligence (π)

Home Research Join Us

# $\pi_0$: Our First Generalist Policy

Published October 31, 2024
Email research@physicalintelligence.company
Paper $\pi_0$.pdf

https://www.physicalintelligence.company/download/pi0.pdf

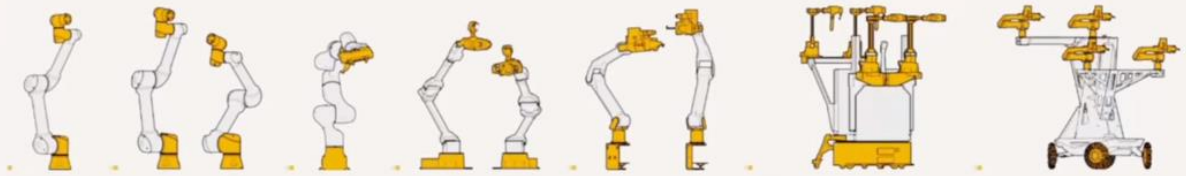
INPUTS

1 Open X Embodiment Dataset.

2 Internet-Scale pre-training.

3 π Dataset: Multiple dexterous robots.

UR5e | Bimanual UR5e | Franka | Bimanual Trossen | Bimanual Arx | Mobile Trossen | Mobile Fibocom



4 $\pi_0$

The model can then perform a wide variety of tasks, via either direct prompting or fine-tuning.

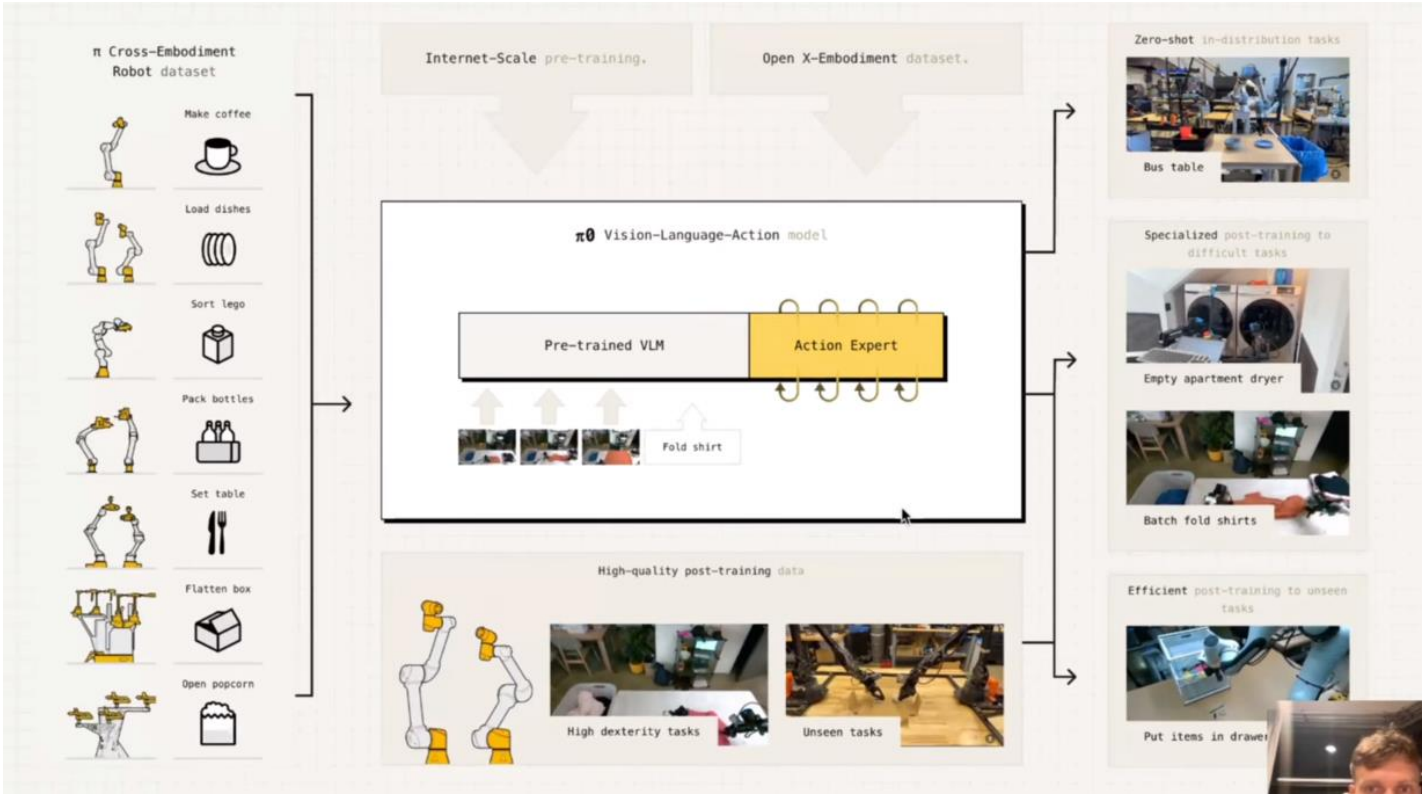
4

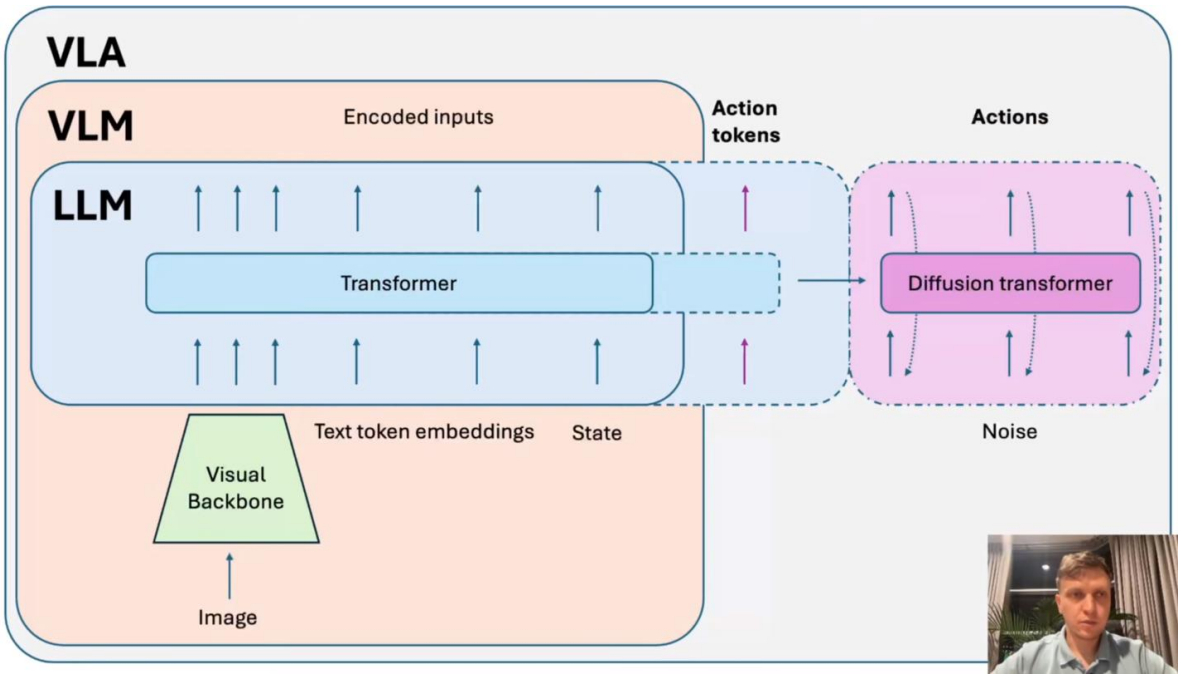
# $\pi_0$

The model can then perform a wide variety of tasks, via either direct prompting or fine-tuning.

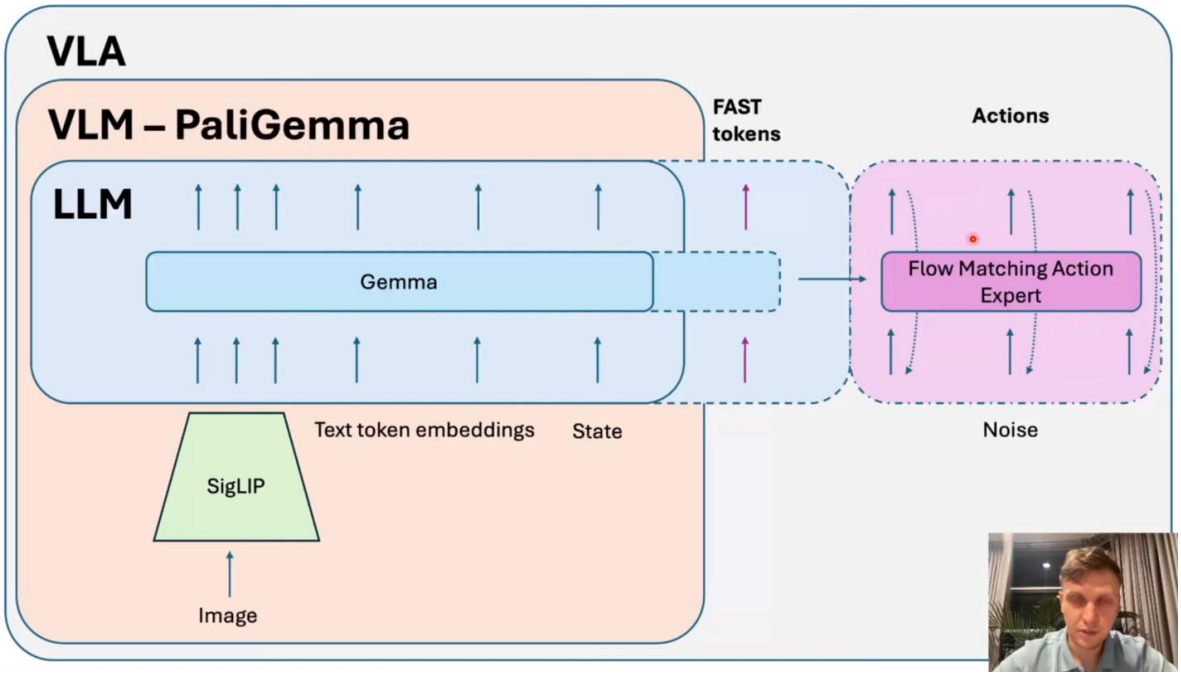
OUTPUT


Our first prototype generalist robot policy is trained on the largest robot interaction dataset to date. The full training mixture includes both open-source data and a large and diverse dataset of dexterous tasks that we collected across 8 distinct robots.

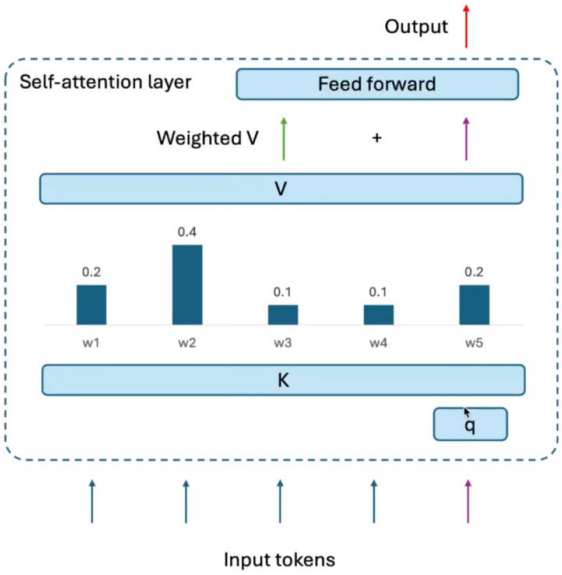




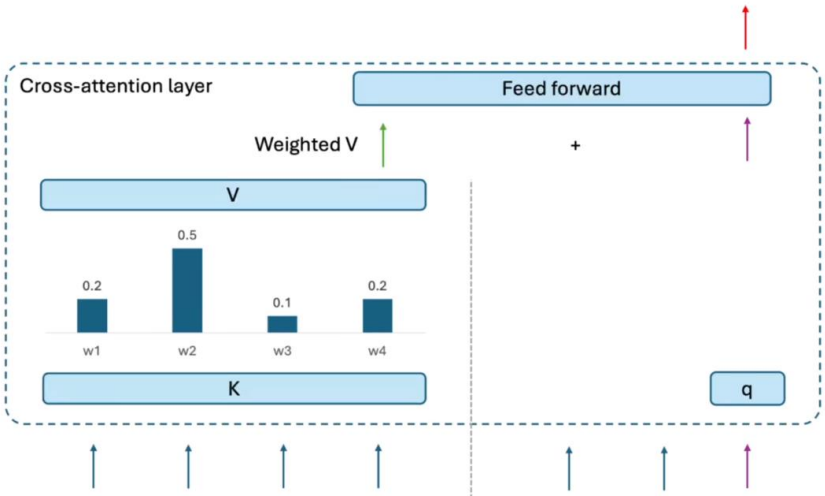

We can now identify the specific models used in the **Pi0 VLA architecture** below

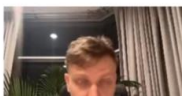


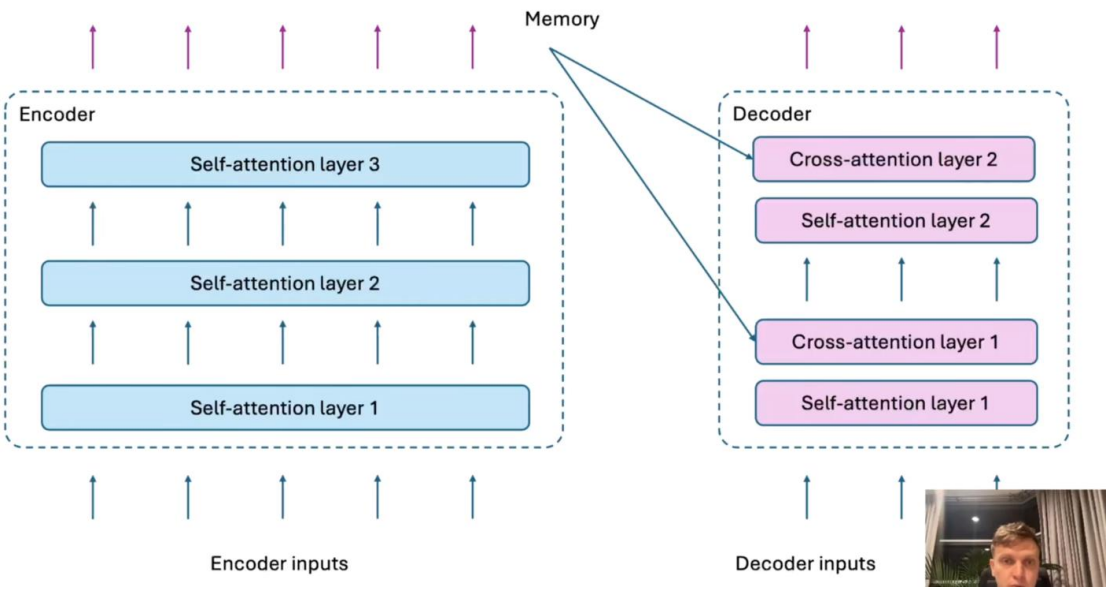

Gemma + SigLiP = PaliGemma model. FAST is the way of predicting Action tokens directly from the VLM developed by Physical Intelligence using image compression principles.

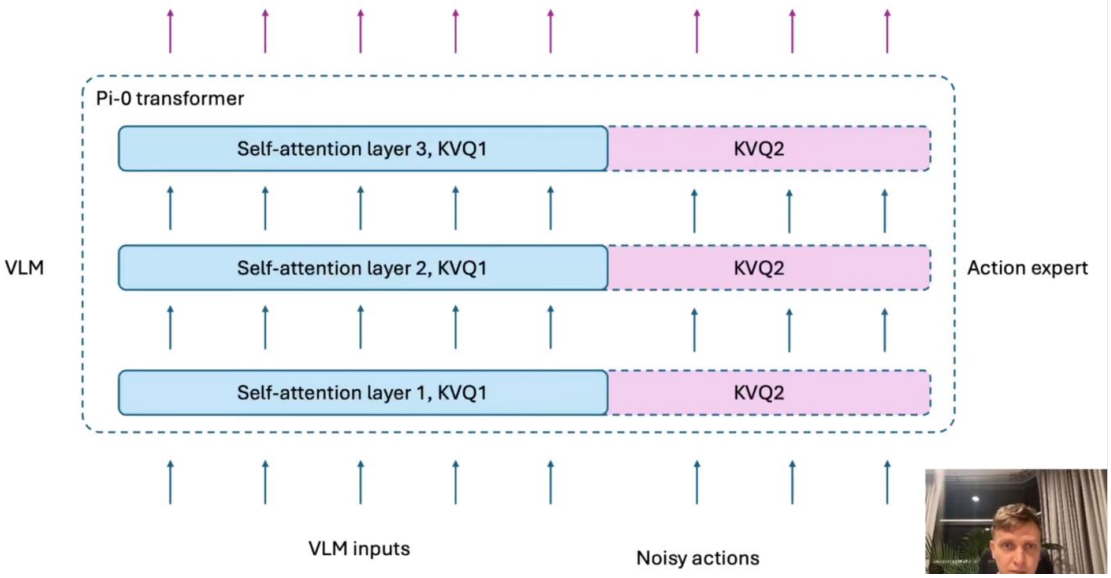



Let us see how Pi0 generates Action tokens from the Query, Key, Attention Weights, Values and Outputs. The Self-Attention layer looks at all the **Input data** tokens while the Cross-Attention only looks at the **Memory** tokens.

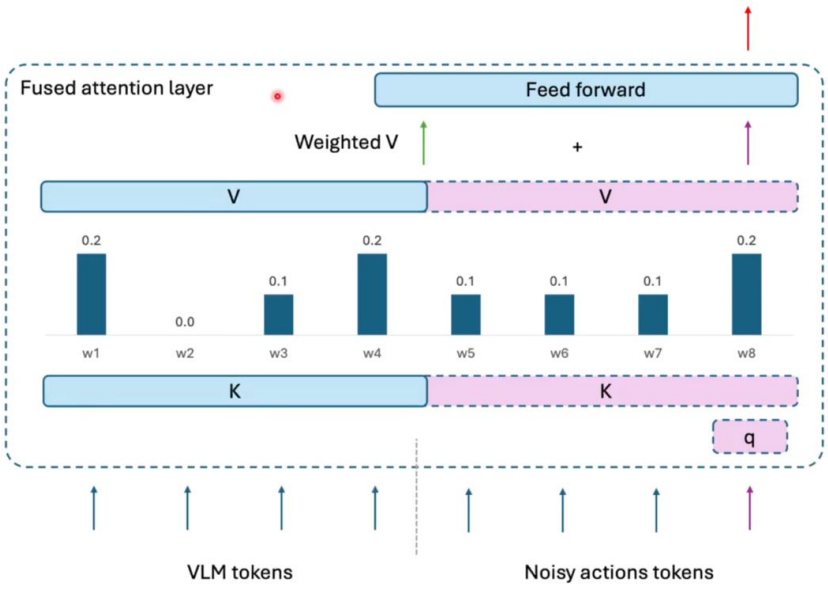


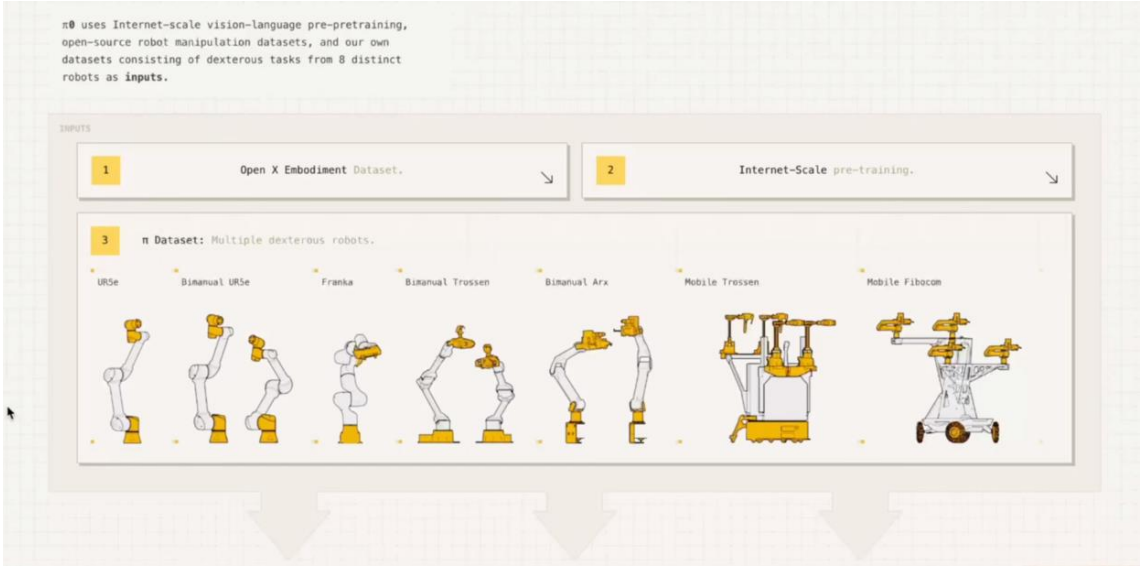



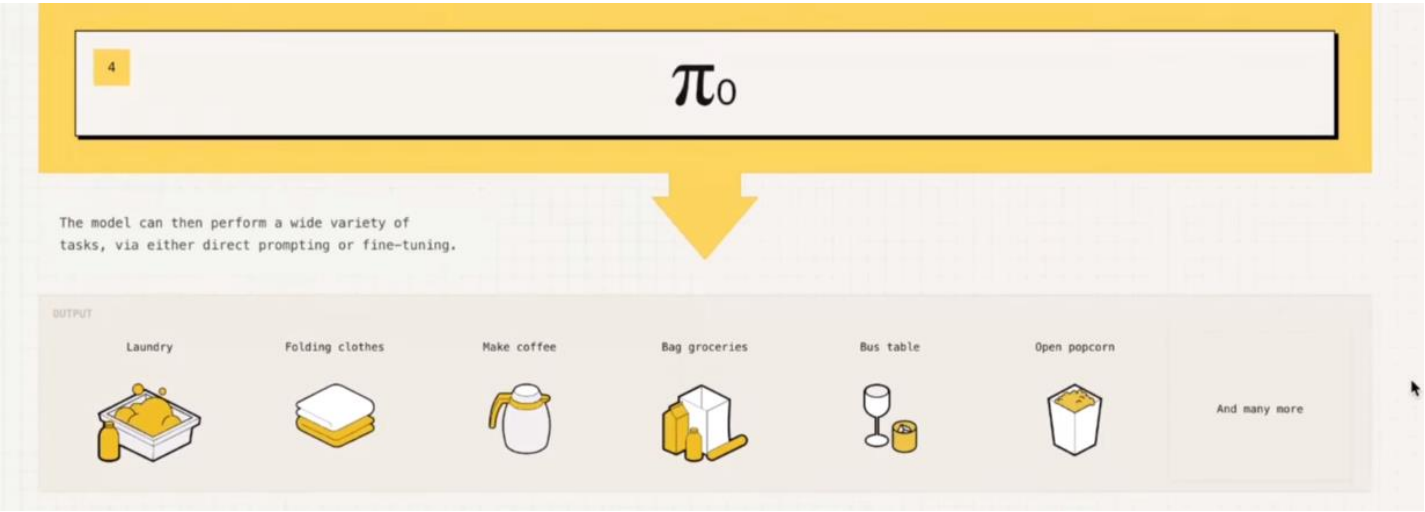





This is what we actually have in the Pi0 transformer, it uses only self-attention layers and no cross-attention layers at all. The VLM part is blue and the Action expert part is pink that tries to denoise using flow matching approach.

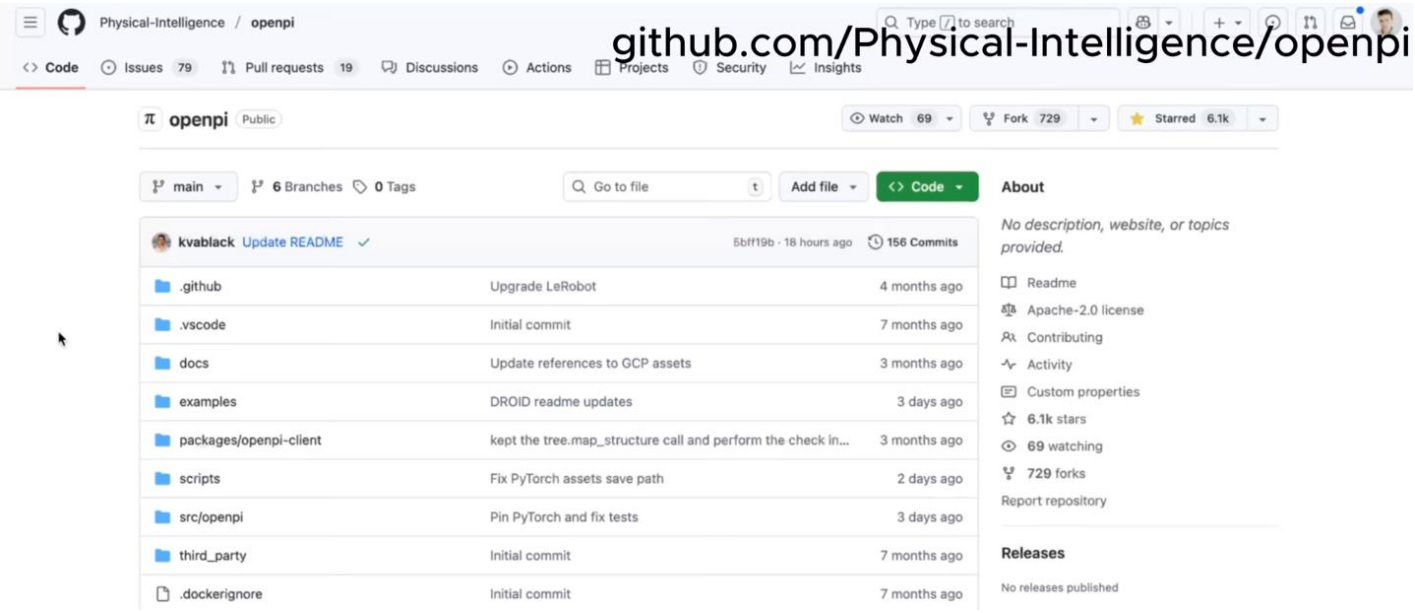





This policy doesn't have any robot-specific action tokens, meaning it is robust enough to work on any robot as long as the model was fine-tuned for the robot at startup.

The currently existing VLAs are not supposed to be used Zero-shot, you are expected to fine-tune the model. You take the pretrained Pi0 model, then collect some small dataset using your particular robot from when your robot completes the particular tasks that you need, like pick and place some objects. Then do some extra training of the model of the policy on your dataset like 15 minutes of data for 55 examples in total.



README Contributing Apache-2.0 license

# openpi

openpi holds open-source models and packages for robotics, published by the Physical Intelligence team.

Currently, this repo contains three types of models:

- the $\pi_0$ model, a flow-based vision-language-action model (VLA).
- the $\pi_0$-FAST model, an autoregressive VLA, based on the FAST action tokenizer.
- the $\pi_{0.5}$ model, an upgraded version of $\pi_0$ with better open-world generalization trained with knowledge insulation. Note that, in this repository, we currently only support the flow matching head for both $\pi_{0.5}$ training and inference.

For all models, we provide *base model* checkpoints, pre-trained on 10k+ hours of robot data, and examples for using them out of the box or fine-tuning them to your own datasets.

This is an experiment: $\pi_0$ was developed for our own robots, which differ from the widely used platforms such as ALOHA and DROID, and though we are optimistic that researchers and practitioners will be able to run creative new experiments adapting $\pi_0$ to their own platforms, we do not expect every such attempt to be successful. All this is to say: $\pi_0$ may or may not work for you, but you are welcome to try it and see!

## Updates

- [Sept 2025] We released PyTorch support in openpi.
- [Sept 2025] We released pi05, an upgraded version of pi0 with better open-world generalization.
- [Sept 2025]: We have added an improved idle filter for DROID training.
- [Jun 2025]: We have added instructions for using `openpi` to train VLAs on the full DROID dataset. This is an approximate open-source implementation of the training pipeline used to train pi0-FAST-DROID.

README Contributing Apache-2.0 license

your own data. We will explain three steps:

1. Convert your data to a LeRobot dataset (which we use for training)
2. Defining training configs and running training
3. Spinning up a policy server and running inference

### 1. Convert your data to a LeRobot dataset

We provide a minimal example script for converting LIBERO data to a LeRobot dataset in `examples/libero/convert_libero_data_to_lerobot.py`. You can easily modify it to convert your own data! You can download the raw LIBERO dataset from here, and run the script with:

```
uv run examples/libero/convert_libero_data_to_lerobot.py --data_dir /path/to/your/libero/data
```

**Note:** If you just want to fine-tune on LIBERO, you can skip this step, because our LIBERO fine-tuning configs point to a pre-converted LIBERO dataset. This step is merely an example that you can adapt to your own data.

### 2. Defining training configs and running training

To fine-tune a base model on your own data, you need to define configs for data processing and training. We provide example configs with detailed comments for LIBERO below, which you can modify for your own dataset:

- `LiberoInputs` and `LiberoOutputs`: Defines the data mapping from the LIBERO environment to the model and vice versa. Will be used for both, training and inference.
- `LeRobotLiberoDataConfig`: Defines how to process raw LIBERO data from LeRobot dataset for training.
- `TrainConfig`: Defines fine-tuning hyperparameters, data config, and weight loader.

We provide example fine-tuning configs for $\pi_0$, $\pi_0$-FAST, and $\pi_{0.5}$ on LIBERO data.

Before we can run training, we need to compute the normalization statistics for the training data. Run the script below with the name of your training config:

```
uv run scripts/compute_norm_stats.py --config-name pi05_libero
```

You can follow the instructions to finetune the model.

## Requirements

To run the models in this repository, you will need an NVIDIA GPU with at least the following specifications. These estimations assume a single GPU, but you can also use multiple GPUs with model parallelism to reduce per-GPU memory requirements by configuring `fsdp_devices` in the training config. Please also note that the current training script does not yet support multi-node training.

| Mode | Memory Required | Example GPU |
|---|---|---|
| Inference | > 8 GB | RTX 4090 |
| Fine-Tuning (LoRA) | > 22.5 GB | RTX 4090 |
| Fine-Tuning (Full) | > 70 GB | A100 (80GB) / H100 |

The repo has been tested with Ubuntu 22.04, we do not currently support other operating systems.

## Installation

When cloning this repo, make sure to update submodules:

```
git clone --recurse-submodules git@github.com:Physical-Intelligence/openpi.git

# Or if you already cloned the repo:
git submodule update --init --recursive
```

We use uv to manage Python dependencies. See the uv installation instructions to set it up. Once uv is installed, run the following to set up the environment:

```
GIT_LFS_SKIP_SMUDGE=1 uv sync
GIT_LFS_SKIP_SMUDGE=1 uv pip install -e .
```

NOTE: `GIT_LFS_SKIP_SMUDGE=1` is needed to pull LeRobot as a dependency.

IliaLarchenko / lerobot_random

Type / to search

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main lerobot_random / scripts / setup_remote.sh

Go to file t

IliaLarchenko Added setup script 648aaaf · 20 minutes ago History

Code Blame 63 lines (54 loc) · 2.21 KB Raw

```
#!/usr/bin/env bash

# I use this script to setup a remote machine to use LeRobot and other tools
# It installs different dependencies that can be missing
# The list can be redundant in some cases and not complete in others
# But it covers most of the cases I faced when worked with LeRobot
# It can take around 10 minutes to install all the dependencies
# Just run `bash setup_remote.sh` to install all the dependencies

set -euo pipefail

# 1) Apt deps
# Some dependencies can be redundant but I included all that caused me issues in the past
SUDO=""
if [ "${EUID:-$(id -u)}" -ne 0 ]; then SUDO="sudo"; fi
export DEBIAN_FRONTEND=noninteractive
$SUDO apt-get update
$SUDO apt-get install -y \
  curl htop git-lfs gcc build-essential python3-dev linux-libc-dev \
  libgl1-mesa-glx libgl1-mesa-dev libglfw3 libglfw3-dev libglew-dev xorg-dev \
  libsvtav1-dev
git lfs install || true

# 2) Ensure conda is available and usable in this script
if ! command -v conda >/dev/null 2>&1; then
  echo "Conda not found, installing Miniconda to \$HOME/miniconda3..."
  tmpd="$(mktemp -d)"
  curl -fsSL https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -o "$tmpd/miniconda.sh"
```

You can use this script if you want to train your model.

README Contributing Apache-2.0 license

## Installation

When cloning this repo, make sure to update submodules:

```
git clone --recurse-submodules git@github.com:Physical-Intelligence/openpi.git

# Or if you already cloned the repo:
git submodule update --init --recursive
```

We use uv to manage Python dependencies. See the uv installation instructions to set it up. Once uv is installed, run the following to set up the environment:

```
GIT_LFS_SKIP_SMUDGE=1 uv sync
GIT_LFS_SKIP_SMUDGE=1 uv pip install -e .
```

NOTE: `GIT_LFS_SKIP_SMUDGE=1` is needed to pull LeRobot as a dependency.

**Docker**: As an alternative to uv installation, we provide instructions for installing openpi using Docker. If you encounter issues with your system setup, consider using Docker to simplify installation. See Docker Setup for more details.

## Model Checkpoints

### Base Models

We provide multiple base VLA model checkpoints. These checkpoints have been pre-trained on 10k+ hours of robot data, and can be used for fine-tuning.

| Model | Use Case | Description | Checkpoint Path |
|---|---|---|---|
| $\pi_0$ | Fine-Tuning | Base $\pi_0$ model for fine-tuning | `gs://openpi-assets/checkpoints/pi0_base` |

After setting up the machine, you need to set up OpenPi using git clone and install uv, then follow the instructions.

README Contributing Apache-2.0 license

## Model Checkpoints

### Base Models

We provide multiple base VLA model checkpoints. These checkpoints have been pre-trained on 10k+ hours of robot data, and can be used for fine-tuning.

| Model | Use Case | Description | Checkpoint Path |
|---|---|---|---|
| $\pi_0$ | Fine-Tuning | Base $\pi_0$ model for fine-tuning | `gs://openpi-assets/checkpoints/pi0_base` |
| $\pi_0$-FAST | Fine-Tuning | Base autoregressive $\pi_0$-FAST model for fine-tuning | `gs://openpi-assets/checkpoints/pi0_fast_base` |
| $\pi_{0.5}$ | Fine-Tuning | Base $\pi_{0.5}$ model for fine-tuning | `gs://openpi-assets/checkpoints/pi05_base` |

### Fine-Tuned Models

We also provide "expert" checkpoints for various robot platforms and tasks. These models are fine-tuned from the base models above and intended to run directly on the target robot. These may or may not work on your particular robot. Since these checkpoints were fine-tuned on relatively small datasets collected with more widely available robots, such as ALOHA and the DROID Franka setup, they might not generalize to your particular setup, though we found some of these, especially the DROID checkpoint, to generalize quite broadly in practice.

| Model | Use Case | Description | Checkpoint Path |
|---|---|---|---|
| $\pi_0$-FAST-DROID | Inference | $\pi_0$-FAST model fine-tuned on the DROID dataset: can perform a wide range of simple table-top manipulation tasks 0-shot in new scenes on the DROID | `gs://openpi-assets/checkpoints/pi0_fast_droid` |

README Contributing Apache-2.0 license

| Model | Use Case | Description | Checkpoint Path |
|---|---|---|---|
| $\pi_0$-FAST-DROID | Inference | $\pi_0$-FAST model fine-tuned on the DROID dataset: can perform a wide range of simple table-top manipulation tasks 0-shot in new scenes on the DROID robot platform | `gs://openpi-assets/checkpoints/pi0_fast_droid` |
| $\pi_0$-DROID | Fine-Tuning | $\pi_0$ model fine-tuned on the DROID dataset: faster inference than $\pi_0$-FAST-DROID, but may not follow language commands as well | `gs://openpi-assets/checkpoints/pi0_droid` |
| $\pi_0$-ALOHA-towel | Inference | $\pi_0$ model fine-tuned on internal ALOHA data: can fold diverse towels 0-shot on ALOHA robot platforms | `gs://openpi-assets/checkpoints/pi0_aloha_towel` |
| $\pi_0$-ALOHA-tupperware | Inference | $\pi_0$ model fine-tuned on internal ALOHA data: can unpack food from a tupperware container | `gs://openpi-assets/checkpoints/pi0_aloha_tupperware` |
| $\pi_0$-ALOHA-pen-uncap | Inference | $\pi_0$ model fine-tuned on public ALOHA data: can uncap a pen | `gs://openpi-assets/checkpoints/pi0_aloha_pen_uncap` |
| $\pi_{0.5}$-LIBERO | Inference | $\pi_{0.5}$ model fine-tuned for the LIBERO benchmark: gets state-of-the-art performance (see LIBERO README) | `gs://openpi-assets/checkpoints/pi05_libero` |
| | | $\pi_{0.5}$ model fine-tuned on the | |

You can also use a finetuned model if you have one of the above robots

README Contributing Apache-2.0 license

## Fine-Tuning Base Models on Your Own Data

We will fine-tune the $\pi_{0.5}$ model on the LIBERO dataset as a running example for how to fine-tune a base model on your own data. We will explain three steps:

1. Convert your data to a LeRobot dataset (which we use for training)
2. Defining training configs and running training
3. Spinning up a policy server and running inference

### 1. Convert your data to a LeRobot dataset

We provide a minimal example script for converting LIBERO data to a LeRobot dataset in `examples/libero/convert_libero_data_to_lerobot.py`. You can easily modify it to convert your own data! You can download the raw LIBERO dataset from here, and run the script with:

```
uv run examples/libero/convert_libero_data_to_lerobot.py --data_dir /path/to/your/libero/data
```

**Note:** If you just want to fine-tune on LIBERO, you can skip this step, because our LIBERO fine-tuning configs point to a pre-converted LIBERO dataset. This step is merely an example that you can adapt to your own data.

### 2. Defining training configs and running training

To fine-tune a base model on your own data, you need to define configs for data processing and training. We provide example configs with detailed comments for LIBERO below, which you can modify for your own dataset:

- `LiberoInputs` and `LiberoOutputs`: Defines the data mapping from the LIBERO environment to the model and vice versa. Will be used for both, training and inference.
- `LeRobotLiberoDataConfig`: Defines how to process raw LIBERO data from LeRobot dataset for training.
- `TrainConfig`: Defines fine-tuning hyperparameters, data config, and weight loader.

We provide example fine-tuning configs for $\pi_0$, $\pi_0$-FAST, and $\pi_{0.5}$ on LIBERO data.

Before we can run training, we need to compute the normalization statistics for the training data. Run the script

README Contributing Apache-2.0 license

## 2. Defining training configs and running training

To fine-tune a base model on your own data, you need to define configs for data processing and training. We provide example configs with detailed comments for LIBERO below, which you can modify for your own dataset:

- `LiberoInputs` and `LiberoOutputs` : Defines the data mapping from the LIBERO environment to the model and vice versa. Will be used for both, training and inference.
- `LeRobotLiberoDataConfig` : Defines how to process raw LIBERO data from LeRobot dataset for training.
- `TrainConfig` : Defines fine-tuning hyperparameters, data config, and weight loader.

We provide example fine-tuning configs for $\pi_0$, $\pi_0$-FAST, and $\pi_{0.5}$ on LIBERO data.

Before we can run training, we need to compute the normalization statistics for the training data. Run the script below with the name of your training config:

```
uv run scripts/compute_norm_stats.py --config-name pi05_libero
```

Now we can kick off training with the following command (the `--overwrite` flag is used to overwrite existing checkpoints if you rerun fine-tuning with the same config):

```
XLA_PYTHON_CLIENT_MEM_FRACTION=0.9 uv run scripts/train.py pi05_libero --exp-name=my_experimen
```

The command will log training progress to the console and save checkpoints to the `checkpoints` directory. You can also monitor training progress on the Weights & Biases dashboard. For maximally using the GPU memory, set `XLA_PYTHON_CLIENT_MEM_FRACTION=0.9` before running training -- this enables JAX to use up to 90% of the GPU memory (vs. the default of 75%).

**Note:** We provide functionality for *reloading* normalization statistics for state / action normalization from pre-training. This can be beneficial if you are fine-tuning to a new task on a robot that was part of our pre-training mixture. For more details on how to reload normalization statistics, see the norm_stats.md file.

## 3. Spinning up a policy server and running inference

Physical-Intelligence / openpi

Type / to search

Code Issues 79 Pull requests 19 Discussions Actions Projects Security Insights

main openpi / src / openpi / policies / libero_policy.py

Go to file t

kvablack Fix normalize/padding ordering 59d1371 · 3 days ago History

Code Blame 100 lines (81 loc) · 4.22 KB

Raw

```python
import dataclasses

import einops
import numpy as np

from openpi import transforms
from openpi.models import model as _model


def make_libero_example() -> dict:
    """Creates a random input example for the Libero policy."""
    return {
        "observation/state": np.random.rand(8),
        "observation/image": np.random.randint(256, size=(224, 224, 3), dtype=np.uint8),
        "observation/wrist_image": np.random.randint(256, size=(224, 224, 3), dtype=np.uint8),
        "prompt": "do something",
    }


def _parse_image(image) -> np.ndarray:
    image = np.asarray(image)
    if np.issubdtype(image.dtype, np.floating):
        image = (255 * image).astype(np.uint8)
    if image.shape[0] == 3:
        image = einops.rearrange(image, "c h w -> h w c")
    return image

```

Symbols

Find definitions and references for functions and other symbols in this file by clicking a symbol below or in the code.

Filter symbols r

- func make_libero_example
- func _parse_image
- class LiberoInputs
  - const model_type
  - func __call__
- class LiberoOutputs
  - func __call__

Create this config file for your robot if it doesn't exist yet. Read the comments to know what is going on.

IliaLarchenko / lerobot_random

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main lerobot_random / vla / pi / lekiwi_policy.py

IliaLarchenko Added useful files for pi0 training and inference 4df3d2e · 20 minutes ago History

Code Blame 92 lines (72 loc) · 3.28 KB Raw

```python
# Based on https://github.com/Physical-Intelligence/openpi/src/openpi/policies/libero_policy.py

import dataclasses

import einops
import numpy as np

from openpi import transforms
from openpi.models import model as _model


def make_libero_example() -> dict:
    """Creates a random input example for the Libero policy."""
    return {
        "observation/state": np.random.rand(9),
        "observation/images/front": np.random.randint(256, size=(640, 360, 3), dtype=np.uint8),
        "observation/images/wrist": np.random.randint(256, size=(640, 360, 3), dtype=np.uint8),
        "observation/images/top": np.random.randint(256, size=(640, 360, 3), dtype=np.uint8),
        "prompt": "do something",
    }


def _parse_image(image) -> np.ndarray:
    image = np.asarray(image)
    if np.issubdtype(image.dtype, np.floating):
        image = (255 * image).astype(np.uint8)
    if image.shape[0] == 3:
        image = einops.rearrange(image, "c h w -> h w c")
```

This is my own example policy file.

main lerobot_random / vla / pi / lekiwi_policy.py ↑ Top

Code Blame 92 lines (72 loc) · 3.28 KB Raw

```python
class LeKiwiInputs(transforms.DataTransformFn):
    action_dim: int

    # Determines which model will be used.
    # Do not change this for your own dataset.
    model_type: _model.ModelType = _model.ModelType.PI0

    def __call__(self, data: dict) -> dict:
        state = transforms.pad_to_dim(data["observation/state"], self.action_dim)

        ## I have 3 cameras and it is not perfectly fit the type of cameras Pi0 uses
        ## So I pass the front image instead of the right wrist image - it still works
        base_image = _parse_image(data["observation/images/top"])
        wrist_image = _parse_image(data["observation/images/wrist"])
        front_image = _parse_image(data["observation/images/front"])

        # Create inputs dict. Do not change the keys in the dict below.
        inputs = {
            "state": state,
            "image": {
                "base_0_rgb": base_image,
                "left_wrist_0_rgb": wrist_image,
                "right_wrist_0_rgb": front_image,
            },
            "image_mask": {
                "base_0_rgb": np.True_,
                "left_wrist_0_rgb": np.True_,
                "right_wrist_0_rgb": np.True_,
            },
        }

        # Pad actions to the model action dimension. Keep this for your own dataset.
        # Actions are only available during training.
        if "action" in data:
            # We are padding to the model action dim.
            actions = transforms.pad_to_dim(data["action"], self.action_dim)
            inputs["actions"] = actions
```

All Symbols

front_image

2 References Search

In this file

52 front_image = _parse_image(data["observation

60 "right_wrist_0_rgb": front_image,

Search for this symbol

The trickiest part is aligning the names of the 1 or 3 cameras.

Physical-Intelligence / openpi

Type / to search

Code | Issues 79 | Pull requests 19 | Discussions | Actions | Projects | Security | Insights

main | openpi / src / openpi / training / config.py | Go to file t

kvablack update paths — 2d70d96 · 3 days ago | History

Code | Blame | 981 lines (890 loc) · 43.2 KB · Raw

```python
"""See _CONFIGS for the list of available configs."""

import abc
from collections.abc import Sequence
import dataclasses
import difflib
import logging
import pathlib
from typing import Any, Literal, Protocol, TypeAlias

import etils.epath as epath
import flax.nnx as nnx
from typing_extensions import override
import tyro

import openpi.models.model as _model
import openpi.models.pi0_config as pi0_config
import openpi.models.pi0_fast as pi0_fast
import openpi.models.tokenizer as _tokenizer
import openpi.policies.aloha_policy as aloha_policy
import openpi.policies.droid_policy as droid_policy
import openpi.policies.libero_policy as libero_policy
import openpi.shared.download as _download
import openpi.shared.normalize as _normalize
import openpi.training.droid_rlds_dataset as droid_rlds_dataset
import openpi.training.misc.roboarena_config as roboarena_config
import openpi.training.optimizer as _optimizer
import openpi.training.weight_loaders as weight_loaders
```

You need to make some changes in this file. Read the comments to understand better

main | openpi / src / openpi / training / config.py | Top

Code | Blame | 981 lines (890 loc) · 43.2 KB · Raw

```python
class SimpleDataConfig(DataConfigFactory):
    def create(self, assets_dirs: pathlib.Path, model_config: _model.BaseModelConfig) -> DataConfig:
        )


@dataclasses.dataclass(frozen=True)
class LeRobotAlohaDataConfig(DataConfigFactory):
    # If true, will convert joint dimensions to deltas with respect to the current state before passing to the model.
    # Gripper dimensions will remain in absolute values.
    use_delta_joint_actions: bool = True
    # If provided, will be injected into the input data if the "prompt" key is not present.
    default_prompt: str | None = None
    # If true, this will convert the joint and gripper values from the standard Aloha space to
    # the space used by the pi internal runtime which was used to train the base model. People who
    # use standard Aloha data should set this to true.
    adapt_to_pi: bool = True

    # Repack transforms.
    repack_transforms: tyro.conf.Suppress[_transforms.Group] = dataclasses.field(
        default=_transforms.Group(
            inputs=[
                _transforms.RepackTransform(
                    {
                        "images": {"cam_high": "observation.images.top"},
                        "state": "observation.state",
                        "actions": "action",
                    }
                )
            ]
        )
    )
    # Action keys that will be used to read the action sequence from the dataset.
    action_sequence_keys: Sequence[str] = ("action",)

    @override
    def create(self, assets_dirs: pathlib.Path, model_config: _model.BaseModelConfig) -> DataConfig:
        data_transforms = _transforms.Group(
            inputs=[aloha_policy.AlohaInputs(adapt_to_pi=self.adapt_to_pi)],
```

main openpi / src / openpi / training / config.py ↑ Top

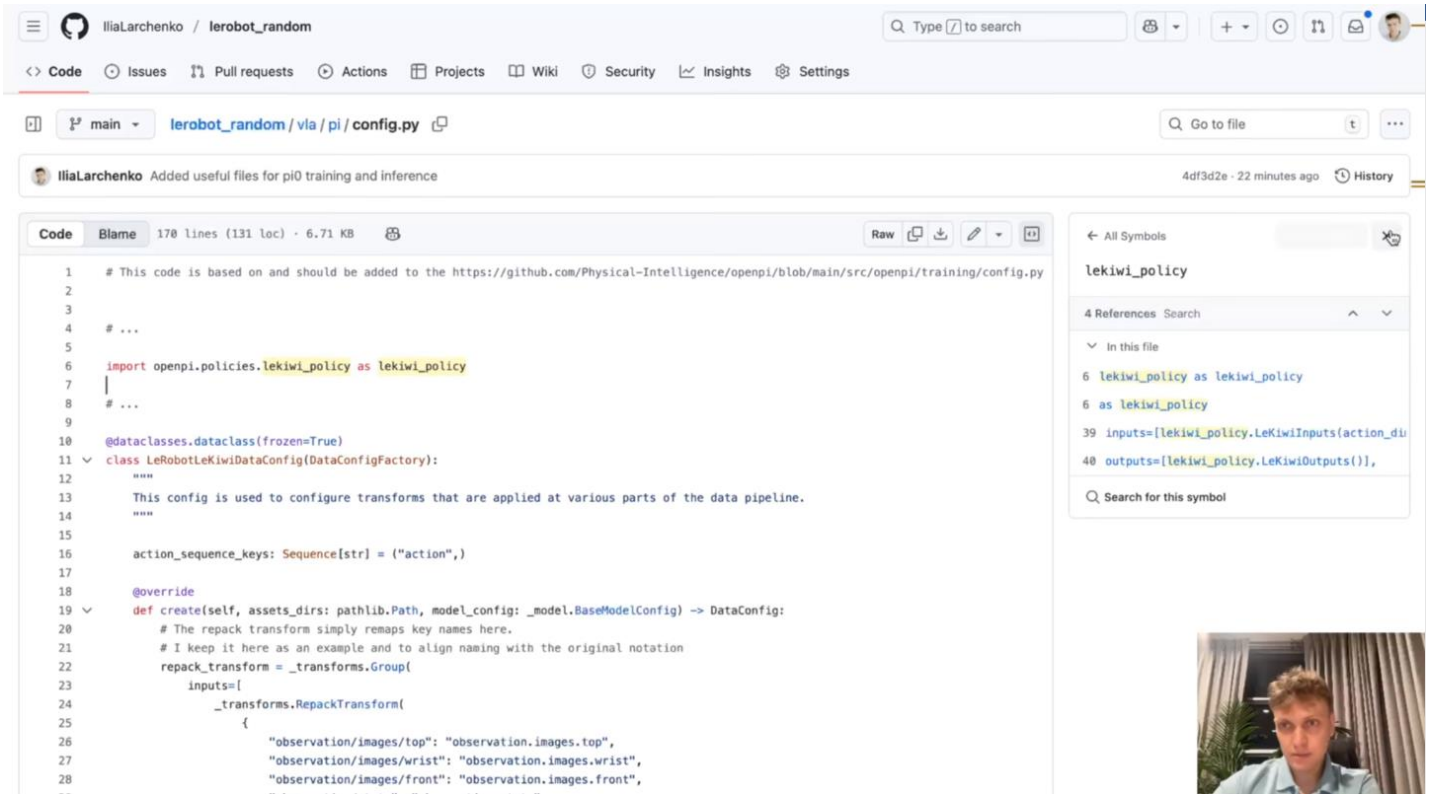
Code Blame 981 lines (890 loc) · 43.2 KB · Raw

```
358  class RLDSDroidDataConfig(DataConfigFactory):
373      def create(self, assets_dirs: pathlib.Path, model_config: _model.BaseModelConfig) -> DataConfig:

417  @dataclasses.dataclass(frozen=True)
418  class LeRobotDROIDDataConfig(DataConfigFactory):
419      """
420      Example data config for custom DROID dataset in LeRobot format.
421      To convert your custom DROID dataset (<10s of hours) to LeRobot format, see examples/droid/convert_droid_data_to_lerobot.py
422      """
423
424      @override
425      def create(self, assets_dirs: pathlib.Path, model_config: _model.BaseModelConfig) -> DataConfig:
426          repack_transform = _transforms.Group(
427              inputs=[
428                  _transforms.RepackTransform(
429                      {
430                          "observation/exterior_image_1_left": "exterior_image_1_left",
431                          "observation/exterior_image_2_left": "exterior_image_2_left",
432                          "observation/wrist_image_left": "wrist_image_left",
433                          "observation/joint_position": "joint_position",
434                          "observation/gripper_position": "gripper_position",
435                          "actions": "actions",
436                          "prompt": "prompt",
437                      }
438                  )
439              ]
440          )
441          # We assume joint *velocity* actions, so we should *not* apply an additional delta transform.
442          data_transforms = _transforms.Group(
443              inputs=[droid_policy.DroidInputs(model_type=model_config.model_type)],
444              outputs=[droid_policy.DroidOutputs()],
445          )
446          model_transforms = ModelTransformFactory()(model_config)
447
448          return dataclasses.replace(
449              self.create_base_config(assets_dirs, model_config),
450              repack_transforms=repack_transform,
451              data_transforms=data_transforms,
452              model_transforms=model_transforms
```

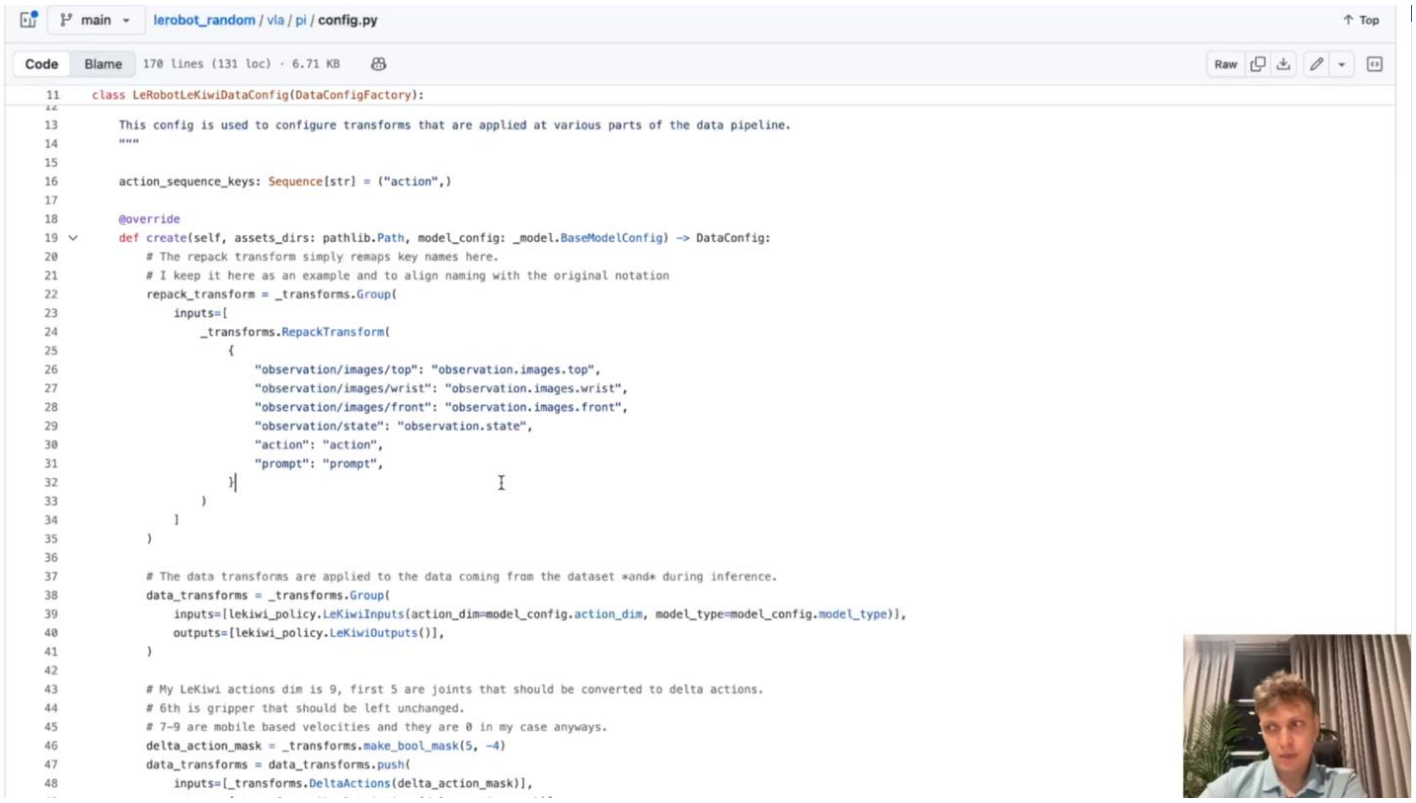
main openpi / src / openpi / training / config.py ↑ Top

Code Blame 981 lines (890 loc) · 43.2 KB · Raw

```
551  _CONFIGS = [
923      ),
924      #
925      # Debugging configs.
926      #
927      TrainConfig(
928          name="debug",
929          data=FakeDataConfig(),
930          batch_size=2,
931          model=pi0_config.Pi0Config(paligemma_variant="dummy", action_expert_variant="dummy"),
932          save_interval=100,
933          overwrite=True,
934          exp_name="debug",
935          num_train_steps=10,
936          wandb_enabled=False,
937      ),
938      TrainConfig(
939          name="debug_restore",
940          data=FakeDataConfig(),
941          batch_size=2,
942          model=pi0_config.Pi0Config(paligemma_variant="dummy", action_expert_variant="dummy"),
943          weight_loader=weight_loaders.CheckpointWeightLoader("./checkpoints/debug/debug/9/params"),
944          overwrite=True,
945          exp_name="debug",
946          num_train_steps=10,
947          wandb_enabled=False,
948      ),
949      TrainConfig(
950          name="debug_pi05",
951          model=pi0_config.Pi0Config(pi05=True, paligemma_variant="dummy", action_expert_variant="dummy"),
952          data=FakeDataConfig(),
953          batch_size=2,
954          num_train_steps=10,
955          overwrite=True,
956          exp_name="debug_pi05",
957          wandb_enabled=False,
958      ),
```




This is my example config file used for the training




Align the naming of the inputs and outputs in your dataset.

main lerobot_random / vla / pi / config.py ↑ Top

Code Blame 170 lines (131 loc) · 6.71 KB Raw

```python
class LeRobotLeKiwiDataConfig(DataConfigFactory):
    def create(self, assets_dirs: pathlib.Path, model_config: _model.BaseModelConfig) -> DataConfig:
                        "observation/state": "observation.state",
                        "action": "action",
                        "prompt": "prompt",
                    }
                )
            ]
        )

        # The data transforms are applied to the data coming from the dataset *and* during inference.
        data_transforms = _transforms.Group(
            inputs=[lekiwi_policy.LeKiwiInputs(action_dim=model_config.action_dim, model_type=model_config.model_type)],
            outputs=[lekiwi_policy.LeKiwiOutputs()],
        )

        # My LeKiwi actions dim is 9, first 5 are joints that should be converted to delta actions.
        # 6th is gripper that should be left unchanged.
        # 7-9 are mobile based velocities and they are 0 in my case anyways.
        delta_action_mask = _transforms.make_bool_mask(5, -4)
        data_transforms = data_transforms.push(
            inputs=[_transforms.DeltaActions(delta_action_mask)],
            outputs=[_transforms.AbsoluteActions(delta_action_mask)],
        )

        # Model transforms include things like tokenizing the prompt and action targets
        # You do not need to change anything here for your own dataset.
        model_transforms = ModelTransformFactory()(model_config)

        # We return all data transforms for training and inference. No need to change anything here.
        return dataclasses.replace(
            self.create_base_config(assets_dirs, model_config),
            repack_transforms=repack_transform,
            data_transforms=data_transforms,
            model_transforms=model_transforms,
            action_sequence_keys=self.action_sequence_keys,
        )
```

Transform the joint actions into delta actions.

main lerobot_random / vla / pi / config.py ↑ Top

Code Blame 170 lines (131 loc) · 6.71 KB Raw

```python
class LeRobotLeKiwiDataConfig(DataConfigFactory):
    def create(self, assets_dirs: pathlib.Path, model_config: _model.BaseModelConfig) -> DataConfig:
        model_transforms = ModelTransformFactory()(model_config)

        # We return all data transforms for training and inference. No need to change anything here.
        return dataclasses.replace(
            self.create_base_config(assets_dirs, model_config),
            repack_transforms=repack_transform,
            data_transforms=data_transforms,
            model_transforms=model_transforms,
            action_sequence_keys=self.action_sequence_keys,
        )


_CONFIGS = [
    # ...
    TrainConfig(
        # Change the name to reflect your model and dataset.
        name="pi0_lekiwi_lora",
        model=pi0_config.Pi0Config(paligemma_variant="gemma_2b_lora", action_expert_variant="gemma_300m_lora", action_horizon=30),

        data=LeRobotLeKiwiDataConfig(
            repo_id="IliaLarchenko/vla_demo",
            base_config=DataConfig(prompt_from_task=True),
        ),

        weight_loader=weight_loaders.CheckpointWeightLoader("gs://openpi-assets/checkpoints/pi0_base/params"),

        # In my case, 5k steps was enough
        num_train_steps=5_000,

        freeze_filter=pi0_config.Pi0Config(
            paligemma_variant="gemma_2b_lora", action_expert_variant="gemma_300m_lora", action_horizon=30
        ).get_freeze_filter(),

        # Turn off EMA for LoRA finetuning.
        ema_decay=None,
```

All Symbols

constant

_CONFIGS

Definition Search

In this file

66 _CONFIGS = [

Search for this symbol

Finally, you need to create your training config. I created multiple configs in my file above.

main ▾ lerobot_random / vla / pi / config.py ↑ Top

Code Blame 170 lines (131 loc) · 6.71 KB Raw

```
63          )
64
65
66  _CONFIGS = [
67      # ...
68      TrainConfig(
69          # Change the name to reflect your model and dataset.
70          name="pi0_lekiwi_lora",
71          model=pi0_config.Pi0Config(paligemma_variant="gemma_2b_lora", action_expert_variant="gemma_300m_lora", action_horizon=30),
72
73          data=LeRobotLeKiwiDataConfig(
74              repo_id="IliaLarchenko/vla_demo",
75              base_config=DataConfig(prompt_from_task=True),
76          ),
77
78          weight_loader=weight_loaders.CheckpointWeightLoader("gs://openpi-assets/checkpoints/pi0_base/params"),
79
80          # In my case, 5k steps was enough
81          num_train_steps=5_000,
82
83          freeze_filter=pi0_config.Pi0Config(
84              paligemma_variant="gemma_2b_lora", action_expert_variant="gemma_300m_lora", action_horizon=30
85          ).get_freeze_filter(),
86
87          # Turn off EMA for LoRA finetuning.
88          ema_decay=None,
89      ),
90
91      TrainConfig(
92          # Change the name to reflect your model and dataset.
93          name="pi0_lekiwi",
94          model=pi0_config.Pi0Config(paligemma_variant="gemma_2b", action_expert_variant="gemma_300m", action_horizon=30),
95
96          data=LeRobotLeKiwiDataConfig(
97              repo_id="IliaLarchenko/vla_demo",
98              base_config=DataConfig(prompt_from_task=True),
99          ),
100
```

README Contributing Apache-2.0 license

## 2. Defining training configs and running training

To fine-tune a base model on your own data, you need to define configs for data processing and training. We provide example configs with detailed comments for LIBERO below, which you can modify for your own dataset:

- `LiberoInputs` and `LiberoOutputs` : Defines the data mapping from the LIBERO environment to the model and vice versa. Will be used for both, training and inference.
- `LeRobotLiberoDataConfig` : Defines how to process raw LIBERO data from LeRobot dataset for training.
- `TrainConfig` : Defines fine-tuning hyperparameters, data config, and weight loader.

We provide example fine-tuning configs for $\pi_0$, $\pi_0$-FAST, and $\pi_{0.5}$ on LIBERO data.

Before we can run training, we need to compute the normalization statistics for the training data. Run the script below with the name of your training config:

```
uv run scripts/compute_norm_stats.py --config-name pi05_libero
```

Now we can kick off training with the following command (the `--overwrite` flag is used to overwrite existing checkpoints if you rerun fine-tuning with the same config):

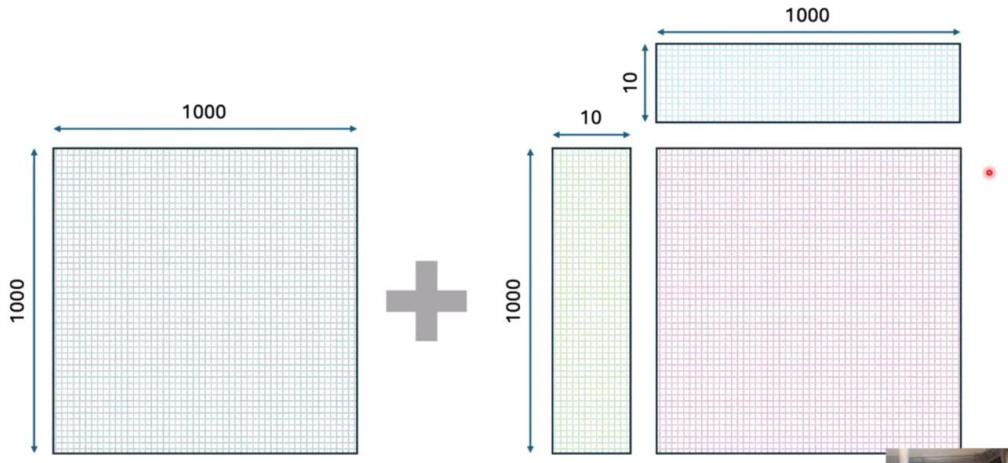
```
XLA_PYTHON_CLIENT_MEM_FRACTION=0.9 uv run scripts/train.py pi05_libero --exp-name=my_experimen
```

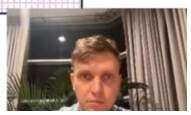
The command will log training progress to the console and save checkpoints to the `checkpoints` directory. You can also monitor training progress on the Weights & Biases dashboard. For maximally using the GPU memory, set `XLA_PYTHON_CLIENT_MEM_FRACTION=0.9` before running training -- this enables JAX to use up to 90% of the GPU memory (vs. the default of 75%).

**Note:** We provide functionality for *reloading* normalization statistics for state / action normalization from pre-training. This can be beneficial if you are fine-tuning to a new task on a robot that was part of our pre-training mixture. For more details on how to reload normalization statistics, see the norm_stats.md file.

## 3. Spinning up a policy server and running inference

Next, let us see how LoRA finetuning works below



README Contributing Apache-2.0 license

## Running Inference for a Pre-Trained Model

Our pre-trained model checkpoints can be run with a few lines of code (here our $\pi_0$-FAST-DROID model):

```
from openpi.training import config as _config
from openpi.policies import policy_config
from openpi.shared import download

config = _config.get_config("pi05_droid")
checkpoint_dir = download.maybe_download("gs://openpi-assets/checkpoints/pi05_droid")

# Create a trained policy.
policy = policy_config.create_trained_policy(config, checkpoint_dir)

# Run inference on a dummy example.
example = {
    "observation/exterior_image_1_left": ...,
    "observation/wrist_image_left": ...,
    ...
    "prompt": "pick up the fork"
}
action_chunk = policy.infer(example)["actions"]
```

You can also test this out in the example notebook.

We provide detailed step-by-step examples for running inference of our pre-trained checkpoints on DROID and ALOHA robots.

**Remote Inference**: We provide examples and code for running inference of our models **remotely**: the model can run on a different server and stream actions to the robot via a websocket connection. This makes it easy to use more powerful GPUs off-robot and keep robot and policy environments separate.

**Test inference without a robot**: We provide a script for testing inference without a robot. This script will generate a random observation and run inference with the model. See here for more details.

After the model is finetuned, you can now start running inference on it

IliaLarchenko / lerobot_random

Type / to search

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main lerobot_random / vla / pi / evaluate_pi0.py

Go to file t

IliaLarchenko Added useful files for pi0 training and inference 4df3d2e · 34 minutes ago History

Code Blame 108 lines (83 loc) · 3.7 KB Raw

```python
#!/usr/bin/env python3

# Adapted from https://github.com/huggingface/lerobot/blob/main/examples/lekiwi/evaluate.py
# And other examples from LeRobot repo
# LeRobot and OpenPi should be installed
# Run `python evaluate_pi0.py` to run the model

import os
# Configure JAX memory allocation before importing JAX-related modules
# Is needed to resolve possible issues with memory allocation
os.environ["XLA_PYTHON_CLIENT_MEM_FRACTION"] = "0.9"

import time
import numpy as np

from lerobot.robots.lekiwi import LeKiwiClient, LeKiwiClientConfig
from lerobot.utils.robot_utils import busy_wait

# Import Pi0 model from openpi
from openpi.training import config as pi0_config
from openpi.policies import policy_config
from huggingface_hub import snapshot_download

# Configuration
FPS = 30
TASK_DESCRIPTION = "Pick the orange ball and put it in the green cup"
ACTIONS_TO_EXECUTE = 20  # Execute this many actions from each predicted chunk
```

main lerobot_random / vla / pi / evaluate_pi0.py ↑ Top

Code Blame 108 lines (83 loc) · 3.7 KB Raw

```python
from huggingface_hub import snapshot_download

# Configuration
FPS = 30
TASK_DESCRIPTION = "Pick the orange ball and put it in the green cup"
ACTIONS_TO_EXECUTE = 20  # Execute this many actions from each predicted chunk

# Load Pi0 model
print("Loading Pi0 model...")

# Use your robot Pi0 config - in my case "pi0_lekiwi_fast", "pi0_lekiwi" or "pi05_lekiwi"
config = pi0_config.get_config("pi0_lekiwi")

# Use you trained policy HF directory
# You should upload your model (assets and params directories from the checkpoint) to Hugging Face to use it here
checkpoint_dir = snapshot_download(repo_id="YOUR_HF_REPO_PATH")
pi0_policy = policy_config.create_trained_policy(config, checkpoint_dir)
print("Pi0 model loaded successfully")

# Connect to robot
# Use your robot IP and ID
robot_config = LeKiwiClientConfig(remote_ip="YOUR_ROBOT_IP", id="YOUR_ROBOT_ID")
robot = LeKiwiClient(robot_config)
robot.connect()

if not robot.is_connected:
    raise ValueError("Robot is not connected!")

print(f"Robot connected. Starting control loop with task: '{TASK_DESCRIPTION}'")
print(f"Will execute {ACTIONS_TO_EXECUTE} actions from each predicted chunk")

# Control loop variables
step = 0
last_actions = None
action_index = 0
pred_times = []

while True:
```

main lerobot_random / vla / pi / evaluate_pi0.py ↑ Top

Code Blame 108 lines (83 loc) · 3.7 KB Raw

```python
print(f"Will execute {ACTIONS_TO_EXECUTE} actions from each predicted chunk")

# Control loop variables
step = 0
last_actions = None
action_index = 0
pred_times = []

while True:
    t0 = time.perf_counter()

    # Run prediction when we need new actions (either first time or when we've executed enough actions)
    if last_actions is None or action_index >= min(ACTIONS_TO_EXECUTE, len(last_actions)):
        # Get robot observation
        observation = robot.get_observation()

        # Prepare input for Pi0
        # Adjust for your robot
        pi0_input = {
            "prompt": TASK_DESCRIPTION,
            "observation/state": observation["observation.state"],
        }

        for k in ["top", "wrist", "front"]:
            pi0_input[f"observation/images/{k}"] = observation[k]

        # Run inference
        t_pred_start = time.perf_counter()
        output = pi0_policy.infer(pi0_input)
        t_pred = time.perf_counter() - t_pred_start

        # Keep track of last 10 prediction times
        pred_times.append(t_pred)
        pred_times = pred_times[-10:]  # Keep last 10

        last_actions = output["actions"]
        action_index = 0
```

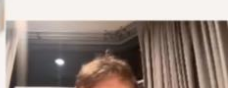
main lerobot_random / vla / pi / evaluate_pi0.py ↑ Top

Code Blame 108 lines (83 loc) · 3.7 KB Raw

```python
        for k in ["top", "wrist", "front"]:
            pi0_input[f"observation/images/{k}"] = observation[k]

        # Run inference
        t_pred_start = time.perf_counter()
        output = pi0_policy.infer(pi0_input)
        t_pred = time.perf_counter() - t_pred_start

        # Keep track of last 10 prediction times
        pred_times.append(t_pred)
        pred_times = pred_times[-10:]  # Keep last 10

        last_actions = output["actions"]
        action_index = 0

        print(f"Step {step}: Predicted {len(last_actions)} actions, will execute {min(ACTIONS_TO_EXECUTE, len(last_actions))}")
        print(f"Prediction took {t_pred:.3f}s (avg over last {len(pred_times)}: {np.mean(pred_times):.3f}s)")

    # Execute action
    else:
        # Get current action from the sequence
        action = last_actions[action_index]

        # Convert action array to robot's expected format
        action_dict = {}
        for i, action_name in enumerate(list(robot.action_features.keys())):
            if i < len(action):
                action_dict[action_name] = action[i]

        robot.send_action(action_dict)
        action_index += 1
        step += 1

    # Maintain FPS
    busy_wait(max(1.0 / FPS - (time.perf_counter() - t0), 0.0))
```

That’s it.

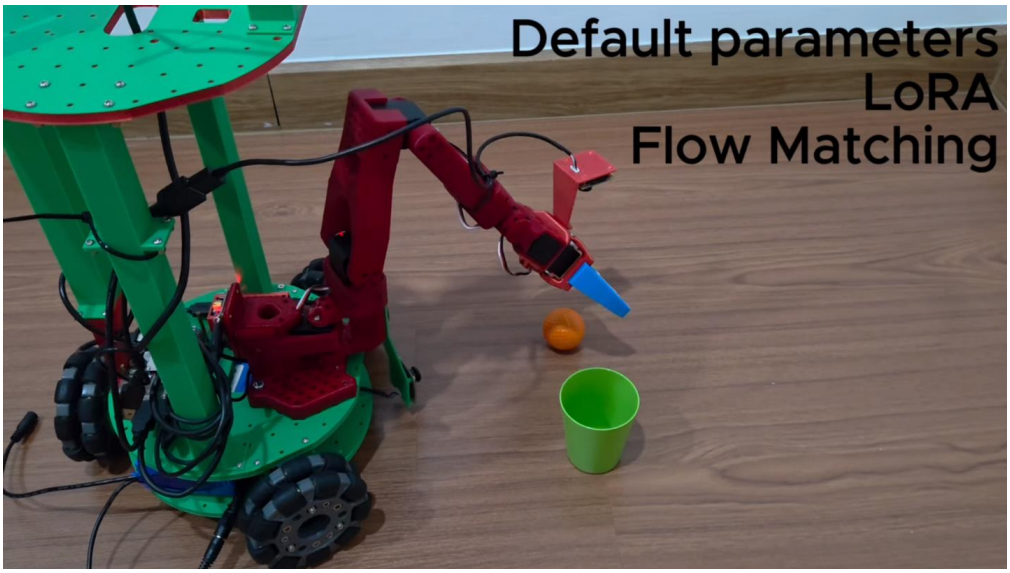

*After post-training, the robot can unload the dryer, bring the clothes over to the table, and fold the clothes into a stack. The video is uncut, from a single policy operating fully autonomously.*

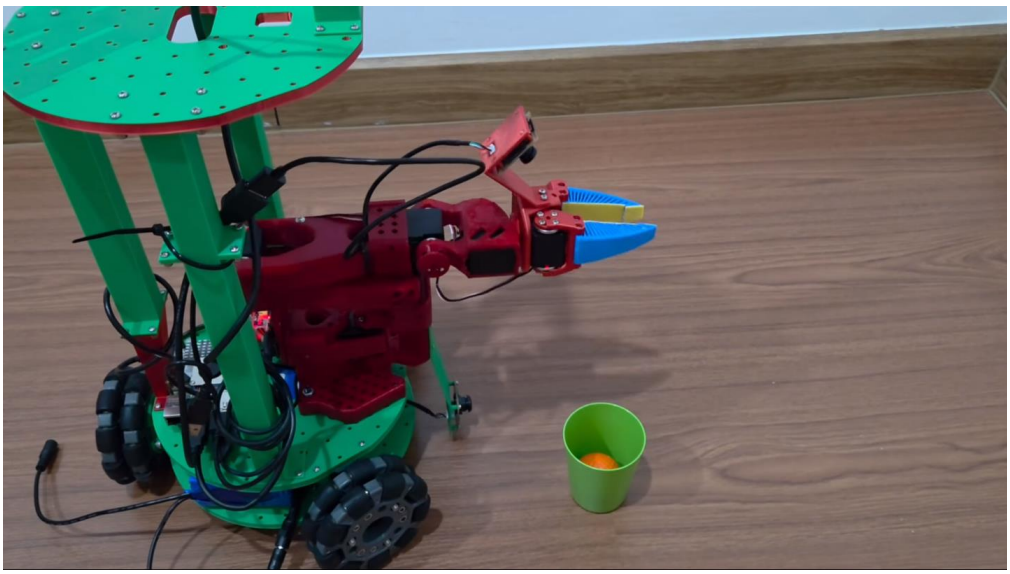


**Laundry**. We fine-tuned $\pi_0$ to fold laundry, using either a mobile robot or a fixed pair of arms. The goal is to get the clothing into a neat stack. This task is exceptionally difficult for robots (...and some humans): while a single t-shirt laid flat on the table can sometimes be folded just by repeating a pre-scripted set of motions, a pile of tangled laundry can be crumpled in many different ways, so it is not enough to simply move the arms through the same motion. To our knowledge, no prior robot system has been demonstrated to perform this task at this level of complexity.

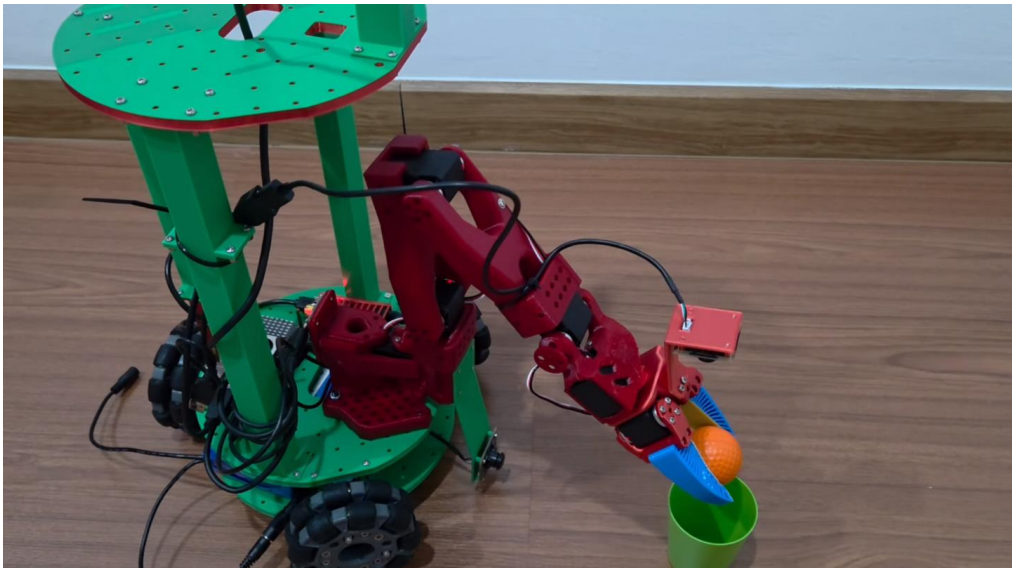

Notably, by training on diverse data, we find that the robot is able to recover when someone tries to intervene in a variety of different ways.


**Table bussing.** We also fine-tuned the model to bus a table. This requires the robot to pick up dishes and trash on the table, putting any dishes, cutlery, or cups into a bussing bin, and putting trash into the trash bin. This task requires the robot to handle a dizzying variety of items. One of the exciting consequences of training $\pi_0$ on large and diverse datasets was the range of emergent strategies that the robot employed: instead of simply grasping each item in turn, the model could stack multiple dishes to put them into the bin together, or shake off trash from a plate into the garbage before placing the plate into the bussing bin.

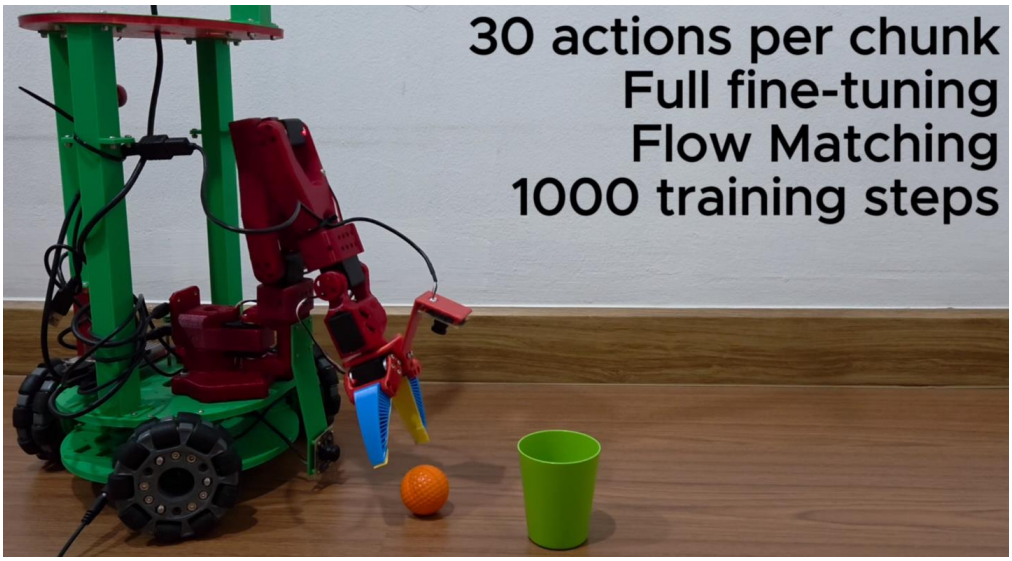


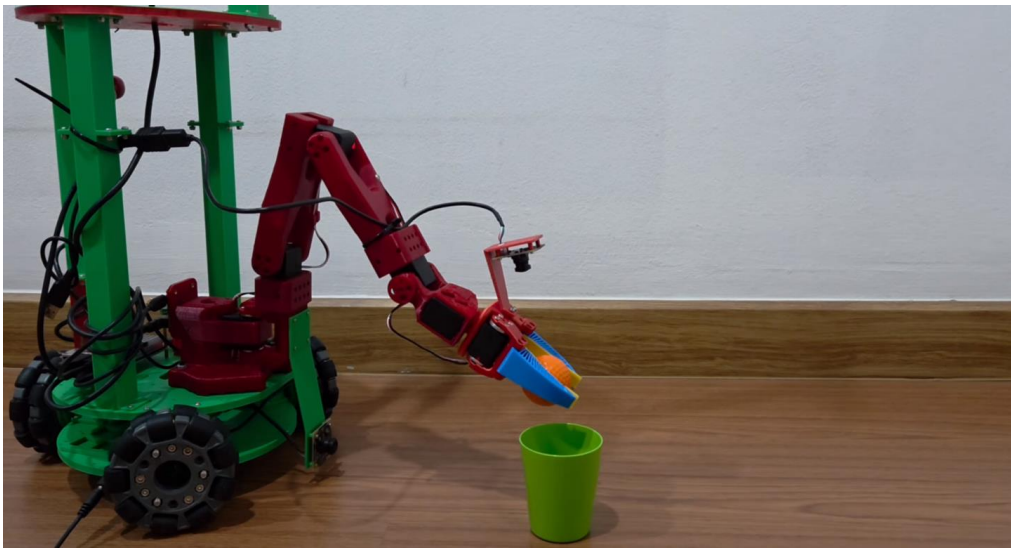



The issue of accuracy is likely the default predicted actions of 50 steps pre chunk

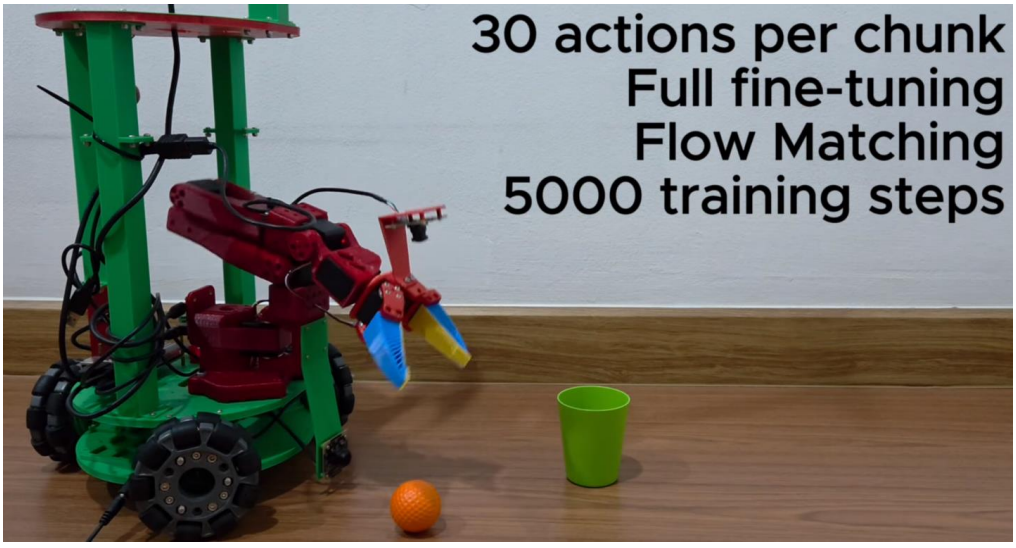



This is better now using 20 predicted chunks for the same model; it completes the tasks correctly now. The wheels are even now moving correctly by transforming the velocity actions into wheel movements.

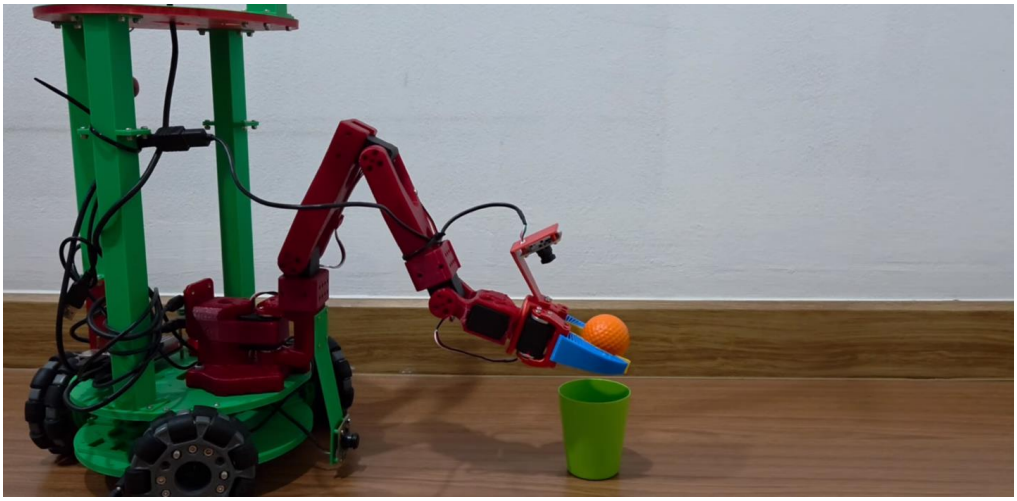


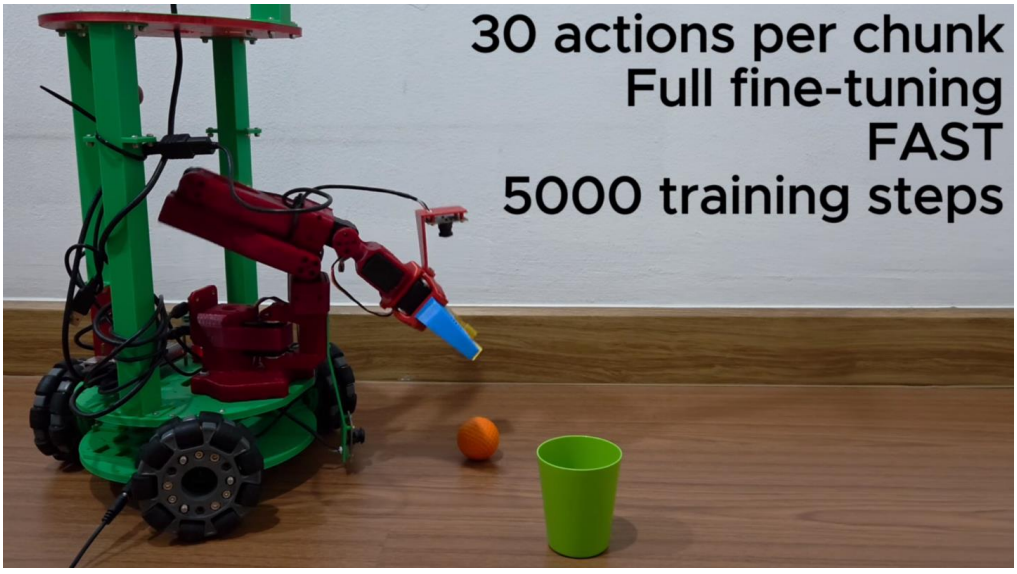
This model achieved the best result and the model converges very fast in training. Based on my inference computer GPU, it currently takes 800ms to make a prediction that contains the 20 next actions, it then runs 20 actions that were just predicted in about 700ms. So, it takes about 1.5s for the robot to execute the predicted actions. A faster GPU inference hardware might speed things up.



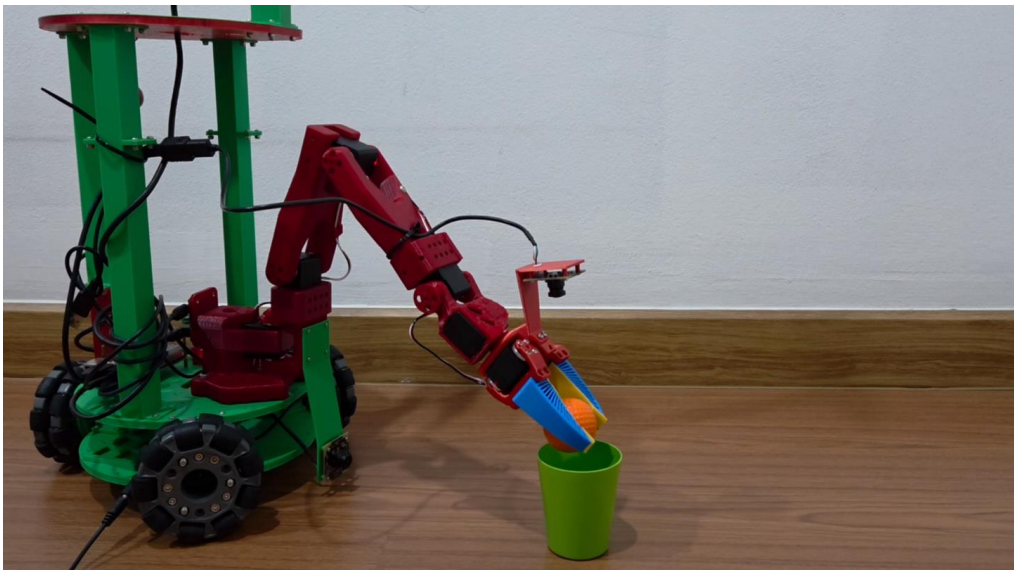

Training longer results in better performance up toa threshold. It recovers from failure and acts faster too.



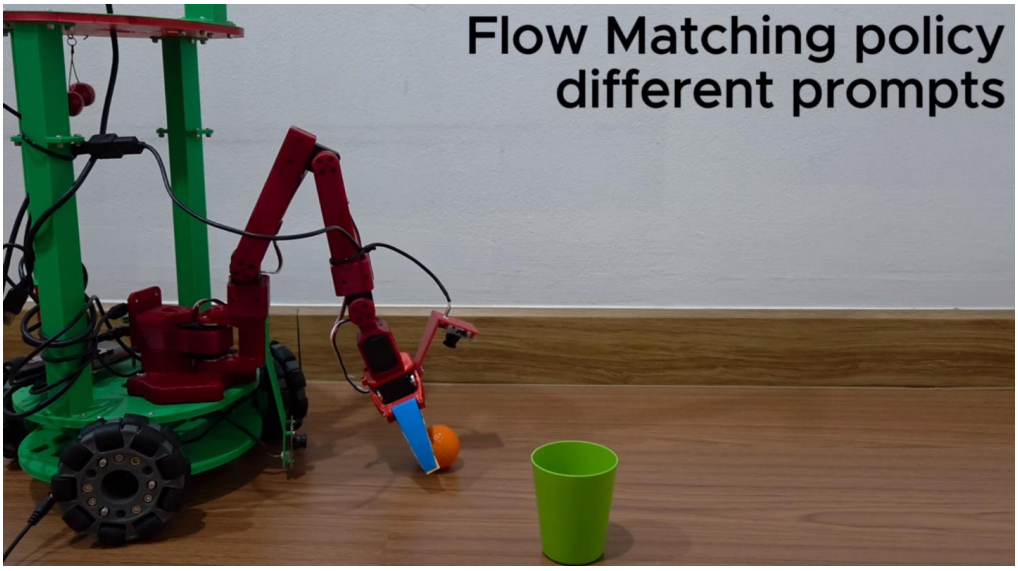
It predicts 30 actions at a time but the robot executes 20 of the actions per cycle of 1.5s.

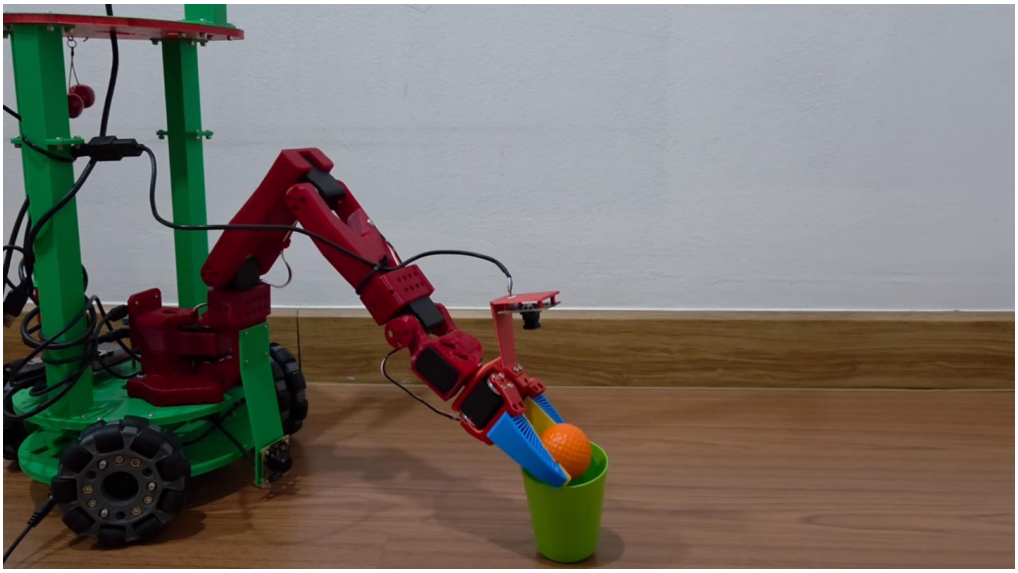



Instead of using Flow Matching, you can train Pi0 with a FAST tokenizer that will predict action tokens autoregressively.

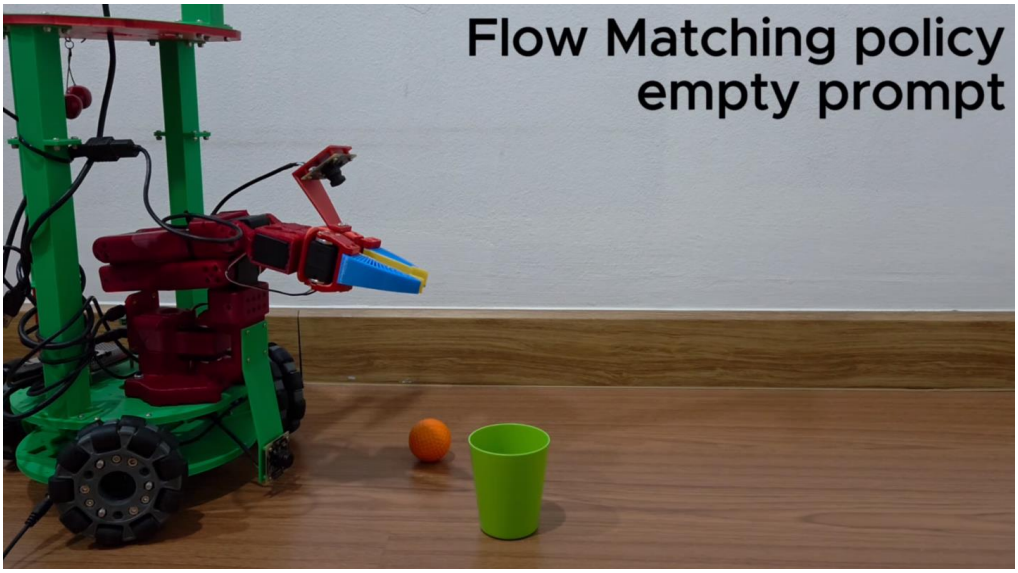


FAST has slower inference time of about 1.5s, which is slower than previous Flow Matching, FAST has faster training convergence in the checkpoints and training steps needed, it also has a fewer success rate. LoRA + FAST didn’t work.

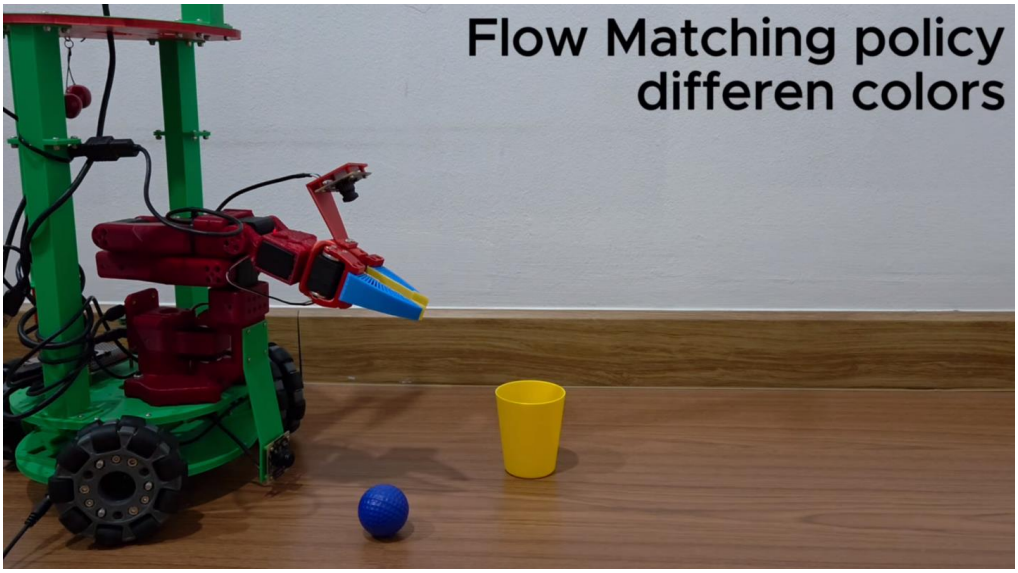


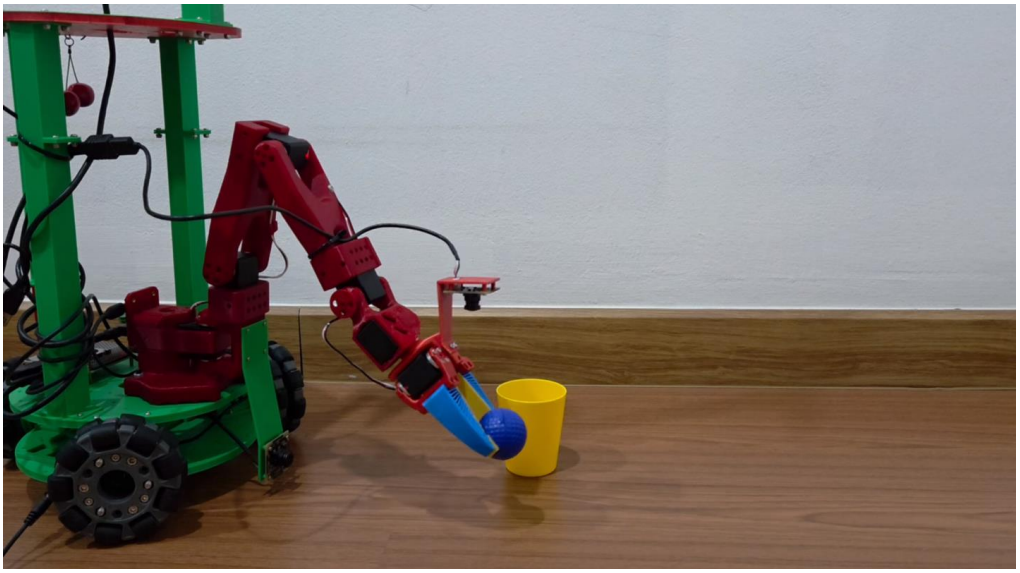
We can give text command to the robot, here we give it a wrong prompt to grab the cup instead. The robot just keeps picking up the ball and dropping in the cup.

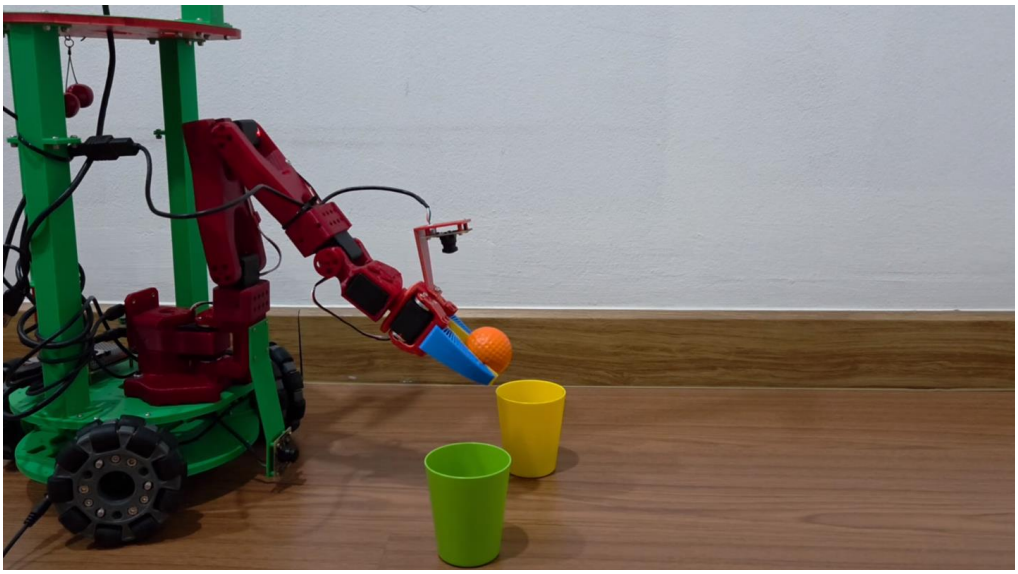



Now I left the prompt empty but the robot keeps grabbing the ball and putting it in the cup, displaying no command following. The prompt is just ignored.

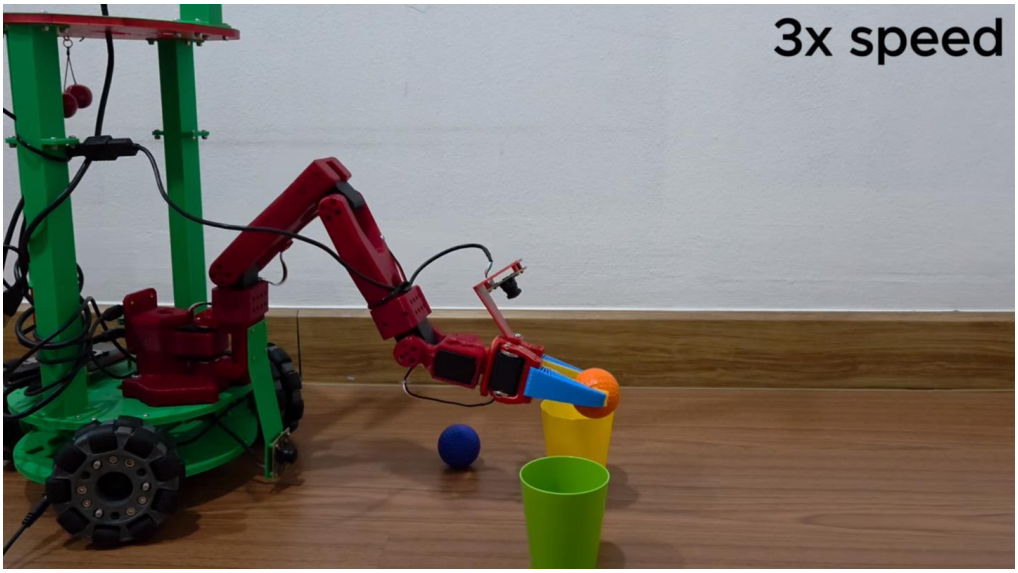


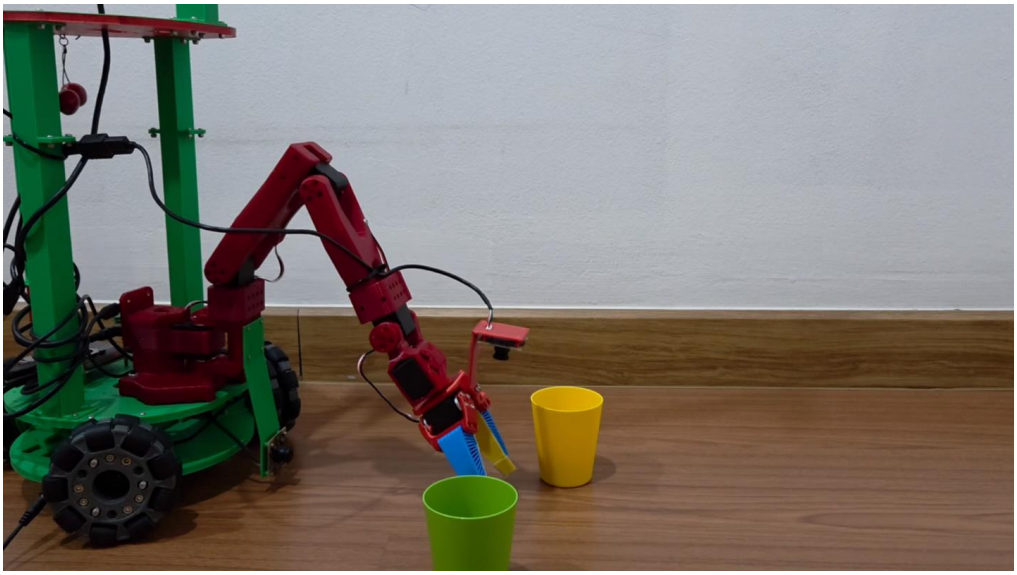

The robot is capable of working with different color balls.





Robot becomes indecisive when giving 2 cups sometimes because it was trained with a few small, single cup samples.



The ball sometimes tries to pick up fathom balls


It works fine in a different environment; the policy is robust enough.