Octo is a transformer-based robot policy trained on a diverse mix of 800k robot trajectories.

Octo

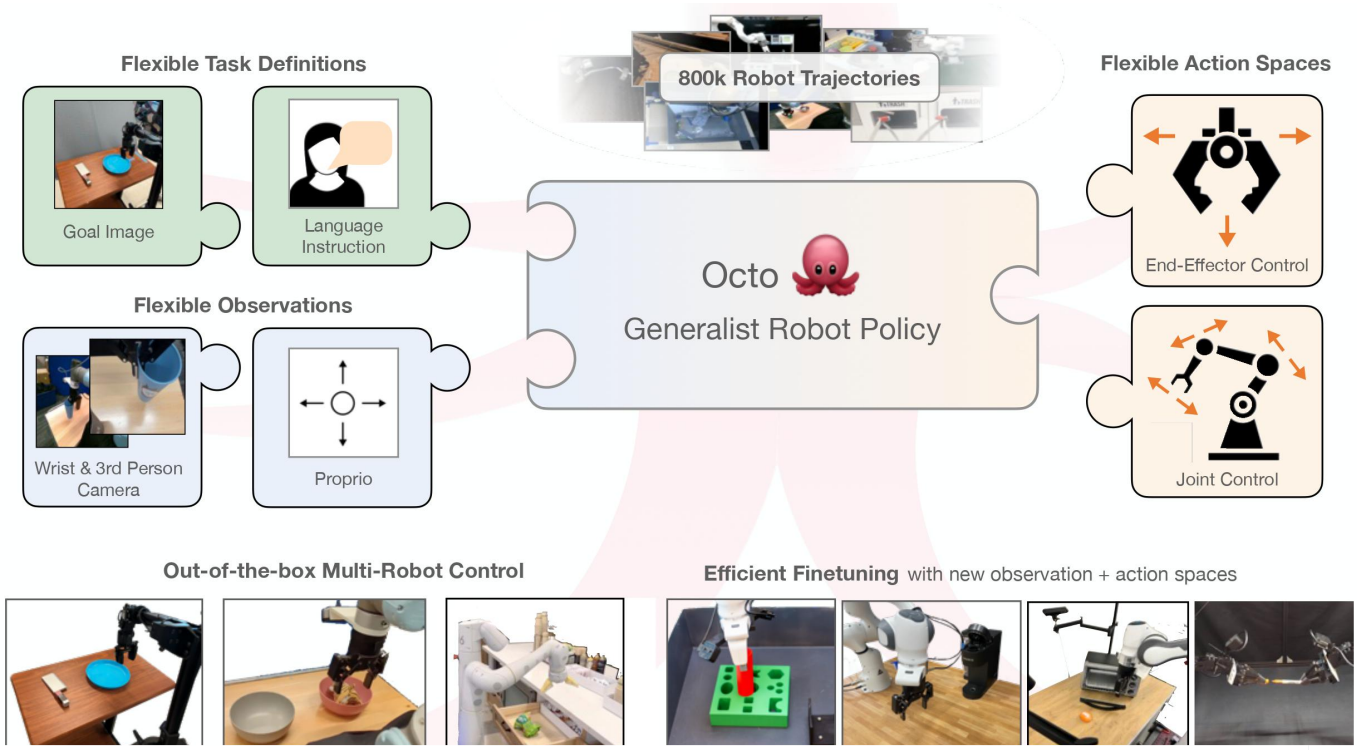
Open in Colab | License MIT | Project Page

This repo contains code for training and finetuning Octo generalist robotic policies (GRPs). Octo models are transformer-based diffusion policies, trained on a diverse mix of 800k robot trajectories.

Get Started

Follow the installation instructions, then load a pretrained Octo model! See examples for guides to zero-shot evaluation and finetuning and Open in Colab for an inference example.

```
from octo.model.octo_model import OctoModel
model = OctoModel.load_pretrained("hf://rail-berkeley/octo-base-1.5")
print(model.get_pretty_spec())
```



README MIT license

Out of the box, Octo supports multiple RGB camera inputs, can control various robot arms, and can be instructed via language commands or goal images. Octo uses a modular attention structure in its transformer backbone, allowing it to be effectively finetuned to robot setups with new sensory inputs, action spaces, and morphologies, using only a small target domain dataset and accessible compute budgets.

Installation

```
conda create -n octo python=3.10
conda activate octo
pip install -e .
pip install -r requirements.txt
```

For GPU:

```
pip install --upgrade "jax[cuda11_pip]==0.4.20" -f https://storage.googleapis.com/jax-releases/jax_cuda_releases.html
```

For TPU

```
pip install --upgrade "jax[tpu]==0.4.20" -f https://storage.googleapis.com/jax-releases/libtpu_releases.html
```

See the Jax Github page for more details on installing Jax.

Test the installation by finetuning on the debug dataset:

```
python scripts/finetune.py --config.pretrained_path=hf://rail-berkeley/octo-small-1.5 --debug
```

Checkpoints

You can find pretrained Octo checkpoints here. At the moment we provide the following model versions:

Model	Inference on 1x NVIDIA 4090	Size
Octo-Base	13 it/sec	93M Params
Octo-Small	17 it/sec	27M Params

Examples

We provide simple example scripts that demonstrate how to use and finetune Octo models, as well as how to use our data loader independently. We provide the following examples:

Octo Inference	Minimal example for loading and running a pretrained Octo model
Octo Finetuning	Minimal example for finetuning a pretrained Octo models on a small dataset with a new observation and action space
Octo Rollout	Run a rollout of a pretrained Octo policy in a Gym environment
Octo Robot Eval	Evaluate a pretrained Octo model on a real WidowX robot
OpenX Dataloader Intro	Walkthrough of the features of our Open X-Embodiment data loader
OpenX PyTorch Dataloader	Standalone Open X-Embodiment data loader in PyTorch

Octo Pretraining

To reproduce our Octo pretraining on 800k robot trajectories, run:

```
python scripts/train.py --config scripts/configs/octo_pretrain_config.py:<size> --name=octo --
config.dataset_kwargs.oxe_kwargs.data_dir=... --config.dataset_kwargs.oxe_kwargs.data_mix=oxe_magic_soup ...
```

To download the pretraining dataset from the Open X-Embodiment Dataset, install the rlds_dataset_mod package and run the prepare_open_x.sh script. The total size of the pre-processed dataset is ~1.2TB.

We run pretraining using a TPUv4-128 pod in 8 hours for the Octo-S model and in 14 hours for Octo-B.

Octo Finetuning

We provide a minimal example for finetuning with a new observation and action space.

We also provide a more advanced finetuning script that allows you to change hyperparameters via a config file and logs finetuning metrics. To run advanced finetuning, use:

```
python scripts/finetune.py --config.pretrained_path=hf://rail-berkeley/octo-small-1.5
```

We offer three finetuning modes depending on the parts of the model that are kept frozen: `head_only`, `head_mlp_only`, and `full` to finetune the full model. Additionally, one can specify the task type to finetune with: `image_conditioned`, `language_conditioned` or `multimodal` for both. For example, to finetune the full transformer with image inputs only use: `--config=finetune_config.py:full,image_conditioned`.

Octo Evaluation

Loading and running a trained Octo model is as easy as:

```
from octo.model import OctoModel

model = OctoModel.load_pretrained("hf://rail-berkeley/octo-small-1.5")
task = model.create_tasks(texts=["pick up the spoon"])
action = model.sample_actions(observation, task, rng=jax.random.PRNGKey(0))
```

We provide examples for evaluating Octo in a simulated Gym environment as well as on a real WidowX robot.

To evaluate on your own environment, simply wrap it in a Gym interface and follow the instructions in the Eval Env README.

Code Structure

	File	Description
Hyperparameters	config.py	Defines all hyperparameters for the training run.
Pretraining Loop	train.py	Main pretraining script.
Finetuning Loop	finetune.py	Main finetuning script.
Datasets	dataset.py	Functions for creating single / interleaved datasets + data augmentation.
Tokenizers	tokenizers.py	Tokenizers that encode image / text inputs into tokens.
Octo Model	octo_model.py	Main entry point for interacting with Octo models: loading, saving, and inference.
Model Architecture	octo_module.py	Combines token sequencing, transformer backbone and readout heads.
Visualization	visualization_lib.py	Utilities for offline qualitative & quantitative eval.

FAQ

What is the `timestep_pad_mask` in the observation dictionary?

The `timestep_pad_mask` indicates which observations should be attended to, which is important when using multiple timesteps of observation history. Octo was trained with a history window size of 2, meaning the model can predict an action using both the current observation and the previous observation. However, at the very beginning of the trajectory, there is no previous observation, so we need to set `timestep_pad_mask=False` at the corresponding index. If you use Octo with a window size of 1, `timestep_pad_mask` should always just be `[True]`, indicating that the one and only observation in the window should be attended to. Note that if you wrap your robot environment with the `HistoryWrapper` (see gym_wrappers.py), the `timestep_pad_mask` key will be added to the observation dictionary for you.

What is `pad_mask_dict` in the observation dictionary?

While `timestep_pad_mask` indicates which observations should be attended to on a timestep level, `pad_mask_dict` indicates which elements of the observation should be attended to within a single timestep. For example, for datasets without language labels, `pad_mask_dict["language_instruction"]` is set to `False`. For datasets without a wrist camera, `pad_mask_dict["image_wrist"]` is set to `False`. For convenience, if a key is missing from the observation dict, it is equivalent to setting `pad_mask_dict` to `False` for that key.

Does `model.sample_actions([...])` return the full trajectory to solve a task?

Octo was pretrained with an action chunking size of 4, meaning it predicts the next 4 actions at once. You can choose to execute all these actions before sampling new ones, or only execute the first action before sampling new ones (also known as receding horizon control). You can also do something more advanced like temporal ensembling.

Updates for Version 1.5

- Improved cross-attention between visual and language tokens by repeating language tokens at every timestep in the context window.
- Augmented the language instructions in the data with rephrasings from GPT-3.5.
- Bug fixes:
  - Turned off dropout in the diffusion head due to incompatibility with layer norm.
  - Fixed an off-by-one error with the attention mask.
  - Fixed an issue where different image augmentations did not get fresh random seeds.

Citation

```
@inproceedings{octo_2023,
    title={Octo: An Open-Source Generalist Robot Policy},
    author = {{Octo Model Team} and Dibya Ghosh and Homer Walke and Karl Pertsch and Kevin Black and Oier Mees and Sudeep
Dasari and Joey Hejna and Charles Xu and Jianlan Luo and Tobias Kreiman and {You Liang} Tan and Pannag Sanketi and Quan Vuong
and Ted Xiao and Dorsa Sadigh and Chelsea Finn and Sergey Levine},
    booktitle = {Proceedings of Robotics: Science and Systems},
```

```
        address  = {Delft, Netherlands},
        year = {2024},
    }
```

Releases 2

v1.5 Latest
on May 23, 2024

+ 1 release

Packages

No packages published

Contributors 13

Languages

- **Python** 100.0%