A Appendix - Brax System Specification

In this section, we demonstrate how to construct a Brax scene using the ProtoBuf specification, as well as a short snippet constructing the same scene pythonically.

```
substeps: 1
dt: .01
gravity { z: -9.8 }
bodies {
  name: "Parent"
  frozen { position { x: 1 y: 1 z: 1 } rotation { x: 1 y: 1 z: 1 } }
  mass: 1
  inertia { x: 1 y: 1 z: 1 }
}
bodies {
  name: "Child"
  mass: 1
  inertia { x: 1 y: 1 z: 1 }
}
joints {
  name: "Joint"
  parent: "Parent"
  child: "Child"
  stiffness: 10000
  child_offset { z: 1 }
  angle_limit { min: -180 max: 180 }
}
```

```
import brax.physics.config_pb2 as config_pb2

simple_system = config_pb2.Config()
simple_system.dt = .01
simple_system.gravity.z = -9.8

parent_body = simple_system.bodies.add()
parent_body.name = "Parent"
parent_body.frozen.position.x, parent_body.frozen.position.y,
    parent_body.frozen.position.z = 1, 1, 1
parent_body.frozen.rotation.x, parent_body.frozen.rotation.y,
    parent_body.frozen.rotation.z = 1, 1, 1
parent_body.mass = 1
parent_body.inertia.x, parent_body.inertia.y, parent_body.inertia.z = 1, 1, 1

child_body = simple_system.bodies.add()
child_body.name="Child"
child_body.mass = 1
child_body.inertia.x, child_body.inertia.y, child_body.inertia.z = 1, 1, 1

joint = simple_system.joints.add()
joint.name="Joint"
joint.parent="Parent"
joint.child="Child"
joint.stiffness = 10000
joint.child_offset.z = 1

joint_limit = joint.angle_limit.add()
joint_limit.min = 180
joint_limit.max = 180
```

B Appendix - Grasp Trajectory

In this appendix we provide a figure depicting a performant policy for grasp.

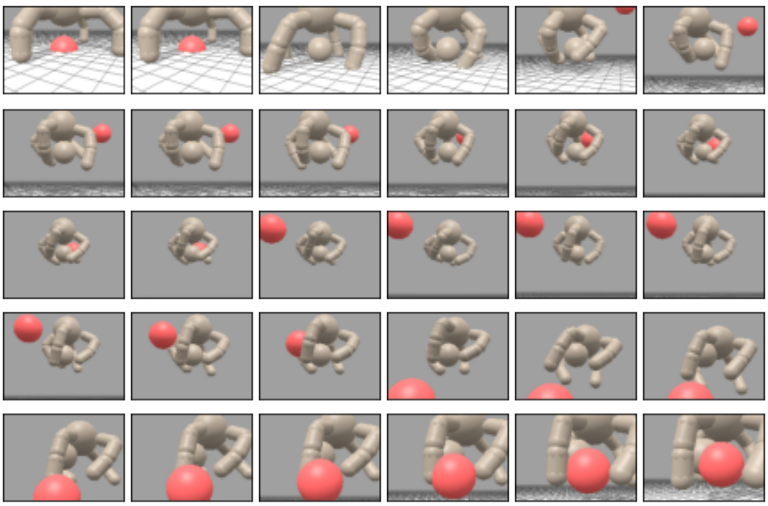

Figure 7: Snapshots from the first 300 steps of a performant grasping policy, simulated and trained within Brax. The hand is able to pick up the ball and carry it to a series of red targets. Once the ball gets close to the red target, the red target is respawned at a different random location.

C Appendix - Hyperparameters for Figures

In this appendix, we detail hardware and hyperparameters used for all training curves in all figures.

Fig. 3: Hardware: 128 Core Intel Xeon Processor at 2.2 Ghz for running MuJoCo environment. 32 Core Intel Xeon Processor at 2.0 GhZ for running 32x-CPU curve. Standard PPO uses[28] and braxppo uses our repo.

Hyperparameters for "standard ppo":
numsteps: 10000000
eval_every_steps: 10000
ppo_learning_rate: 3e-4
ppo_unroll_length: 16
batch_size: 2048 // 16
ppo_num_minibatches: 32
ppo_entropy_cost: 0.0
ppo_value_loss_coef: 0.25
ppo_num_epochs: 10
obs_normalization: True
ppo_clip_value: False

Hyperparameters for braxppo:
total_env_steps: 10000000
eval_frequency: 20
reward_scaling: 10
episode_length: 1000
normalize_observations: True
action_repeat: 1
entropy_cost: 1e-3
learning_rate: 3e-4
discounting: 0.95
num_envs: 2048
unroll_length: 5
batch_size: 1024
num_minibatches: 16
num_update_epochs: 4

Fig. 4: Hardware: 32 Core Intel Xeon Processor at 2.2 Ghz for running environments. 2x2 TPUv2 for learning algorithm, using[28]
SAC Hyperparameters for humanoid and ant:
sac_learning_rate: 3e-4
sac_reward_scale: 0.1
min_replay_size: 10000
num_steps: 5000000
eval_every_steps: 10000
grad_updates_per_batch: 64

SAC Hyperparameters for halfcheetah:
sac_learning_rate: 6e-4
sac_reward_scale: 10.0
min_replay_size: 10000
num_steps: 5000000
eval_every_steps: 10000
grad_updates_per_batch: 32
discount: .97

Fig. 6: Hardware: 32 Core Intel Xeon Processor at 2.2 Ghz for running environments. 1x1 TPUv2 for learning algorithm, using brax-APG and brax-PPO.
brax-ppo hparams: total_env_steps: 40000000
reward_scaling: 5
episode_length: 100
normalize_observations: True
action_repeat: 4
entropy_cost: 1e-3
discounting: 0.95
num_envs: 512
learning_rate: 0.0009626744626757241
unroll_length: 50
batch_size: 256
num_minibatches: 32
num_update_epochs: 8

brax-apg (using ADAM as the underlying optimizer) hparams:
number of training steps: 400
episode_length: 100
action_repeat: 4
learning_rate: 0.002
num_envs: 38
max_gradient_norm: 1e8

D Appendix - Hyperparameter Sweeps

In this appendix, we plot the top 5 performing training curves found in our exhaustive hyperparameter sweeps. The precise values of hyperparameters can be found in zipped, sorted json files **here**[41]. Because these files are sorted first by environment, and then by reward, the indices in the figure legends match the indices of the hyperparameters in the corresponding lists, starting at each indicated environment. For example, "ppo_10_million halfcheetah index 1" means this training curve came from the ppo_10_million dataset here[41], and that it was the first hyperparameter listed under the halfcheetah.

All plots were generated on a 1x1 TPUv2—i.e., the hardware available on Colab's free TPU tier.

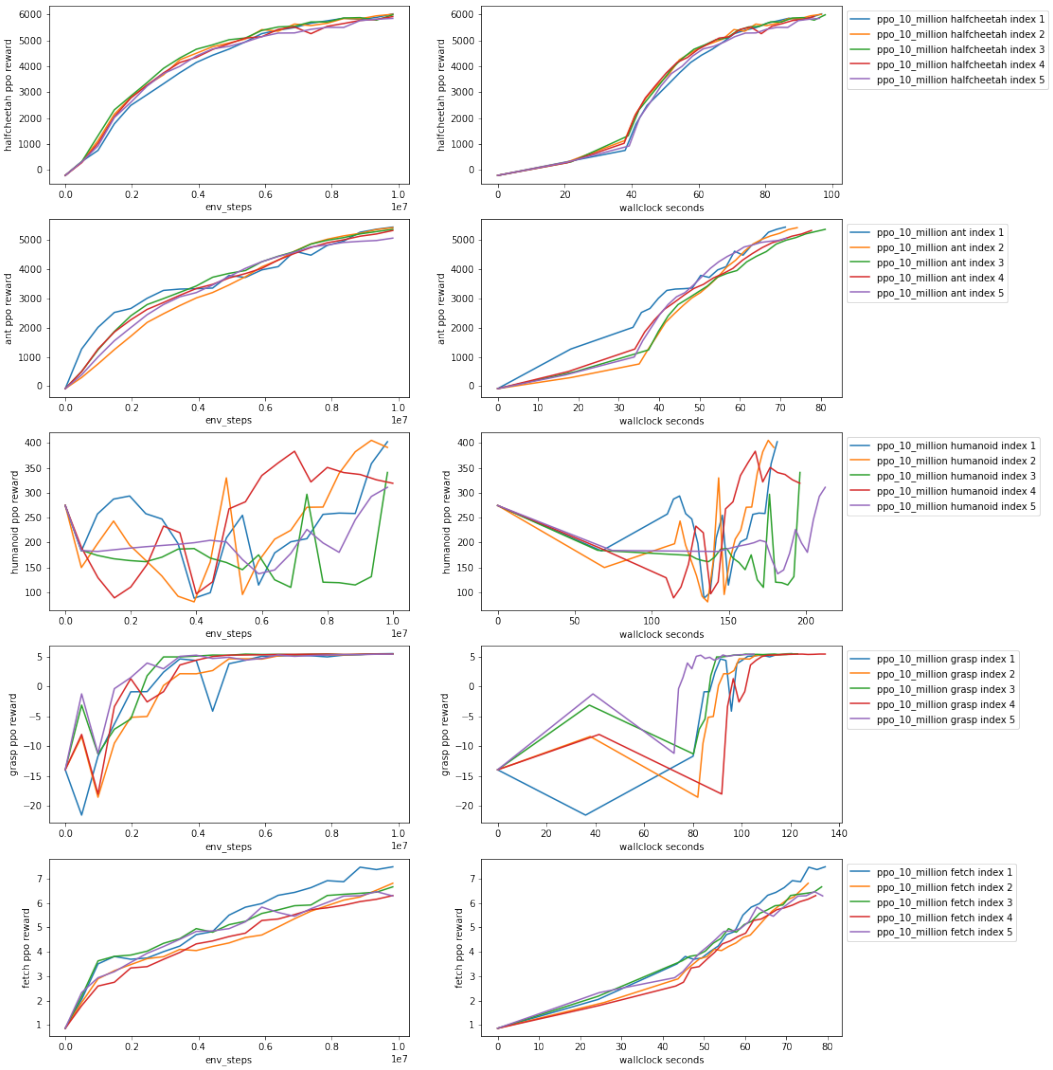

Figure 8: Reward curves for the 5 environments in this release over 10 million steps of braxppo training. Left column indicate reward versus number of steps, right column is the same data duplicated with wallclock time in seconds instead of steps. Note that grasp and humanoid do not find successful policies within 10 million steps.

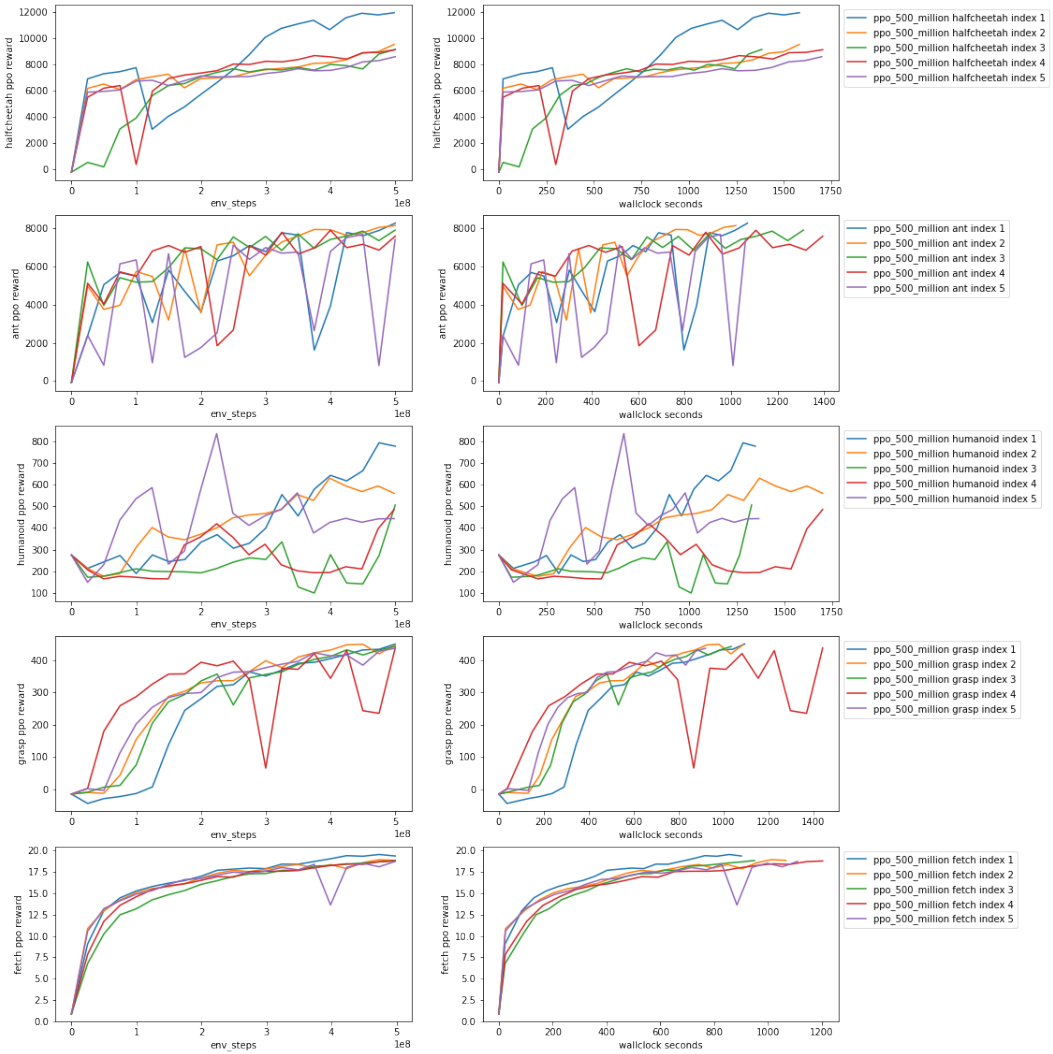

Figure 9: Reward curves for the 5 environments in this release over 500 million steps of braxppo training. Left column indicate reward versus number of steps, right column is the same data duplicated with wallclock time in seconds instead of steps. All policies but humanoid are solvable over this timescale with ppo.

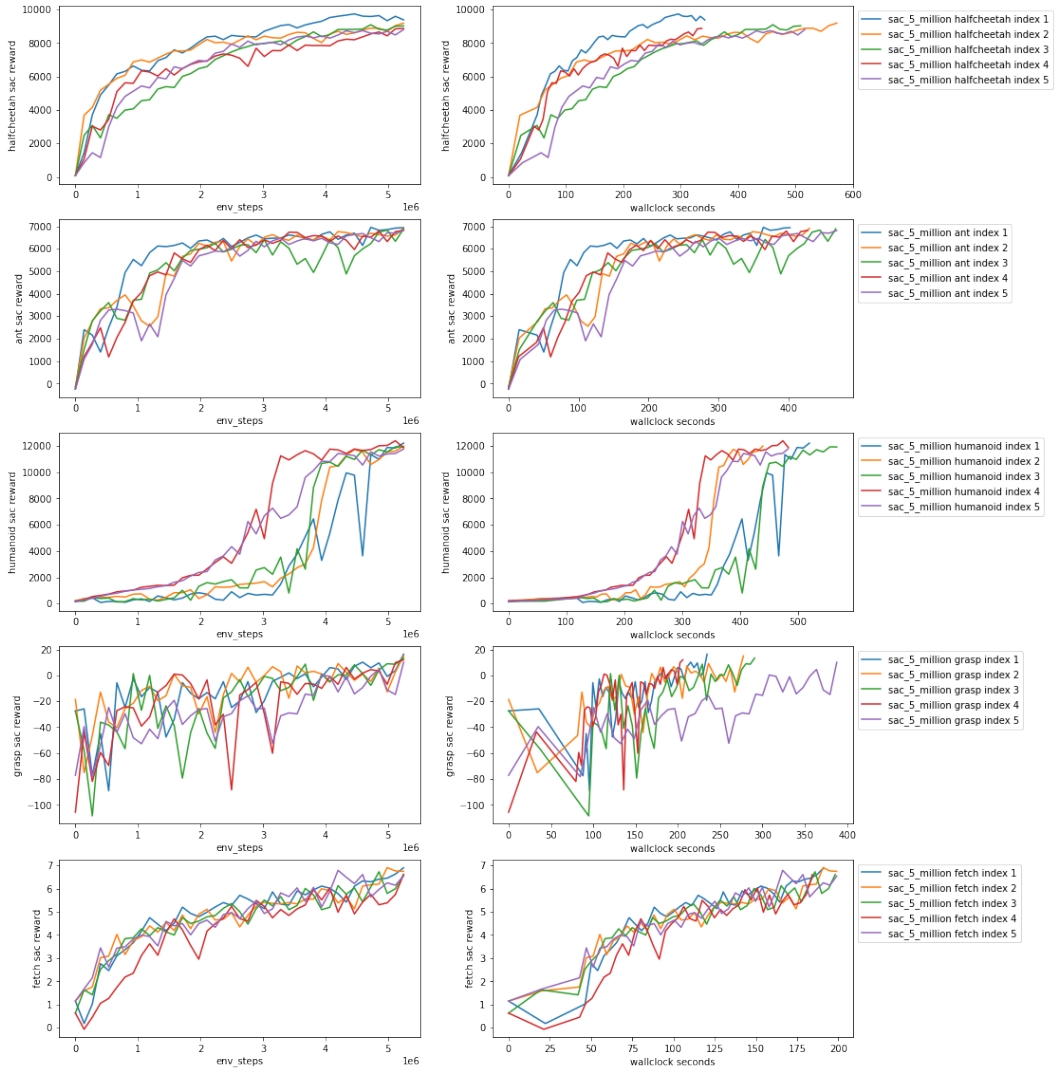

Figure 10: Reward curves for the 5 environments in this release over 5 million steps of brax-sac training. Left column indicate reward versus number of steps, right column is the same data duplicated with wallclock time in seconds instead of steps. Note that humanoid *is* solved via SAC, but here grasp is *not*.

E Appendix - Major Differences from Mujoco

In this appendix, we call out major differences between our implementations of halfcheetah, ant, and humanoid compared to the original MuJoCo-*-v2 envs. Over time, we will work to bring closer parity between our implementation and MuJoCo's, but we defer exhaustive analysis (e.g., trying to transfer policies between Brax and MuJoCo, sim2real style) to future work.

E.1 Halfcheetah

MuJoCo uses joints with the world to achieves 2d-planar motion. In contrast, Brax directly masks integration updates to rotational and translational motion so that the halfcheetah can only move in a plane, and only rotate around one axis. Mass, inertia, and actuator scales were chosen to be as close to MuJoCo's halfcheetah as possible. We speculate that the remaining performance difference in Fig. 4 come down to engine-specific differences of how we implement contact physics, as well as our specific actuation model. This environment is also known to be somewhat pathological[49], where the highest performing policies found in mujoco halfcheetah are physics-breaking, thus comparing extremely high performing policies (e.g., >10,000 score) between the two implementations is theoretically fraught, because it amounts to comparing ways in which the two engines break down.

Because the standard SAC hyperparameters we used in our ACME implementation have been aggressively optimized for the existing MuJoCo environments, it's also possible that we're simply in a poor hypervolume in hyperparameter space for our Brax search, though we did search fairly aggressively for performant hyperparameters. Regardless of the absolute magnitudes of the reward difference between the two implementations, both braxppo and braxsac find quite performant locomotive gaits. We will continue to reduce the performance disparity in future releases.

E.2 Ant

Brax's Ant has tuned mass, inertia, and actuator strengths. Brax's reward function ignores the contact cost (see[50]). Otherwise, this environment achieves the most qualitatively similar gaits between the two engines.

E.3 Humanoid

Besides the usual tuning of mass, inertia, and actuator strengths, humanoid's reward function is slightly modified. The regularization penalty for torquing joints is lower in our environment (.01 compared to MuJoCo's .1), and the reset condition for where the torso triggers a done is from .6 to 2.1 in Brax, compared with MuJoCos 1.0 and 2.0.

Additionally, we implement three-degree-of-freedom actuators slightly differently than MuJoCo. For more details, see our joints implementation[51].

F Appendix - Large Scale Failure Analysis

In this appendix, we report results of scaling up the number of parallel environments on an accelerator until accelerator failure for a suite of large clusters of TPUs. Fig. 11 depicts scaling curves similar to 2, but over 2 more orders of magnitude. At the highest end, the TPUv3 8x16 architecture can simualte tens of millions of Ants in parallel, at over 100 million steps per second, representing hundreds of millions of parallel simulated rigid bodies.

Note that because the raw steps per second is asymptoting to a constant in these curves, this means that serial execution time in slowing down—e.g., doubling the number of parallel environments causes the total execution speed per environment to drop by half, because the accelerator's parallel capacity has already been saturated. Thus, datapoints far to the right, deep in the plateau of performance have slow serial execution speed. We achieve the best training performance before this plateau behavior has fully set in. For a single accelerator (e.g. a 1x1 slice of a TPUv2 or TPUv3), the best performance is typically around 2,048 environments in parallel (approximately 20,000 parallel rigid bodies).

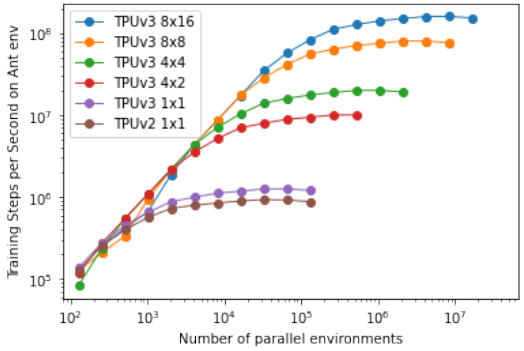

Figure 11: Performance of various TPU clusters on increasing numbers of Ant environments in parallel. Ant includes 10 rigid bodies, thus the number of simulated rigid bodies is 10x the indicated number of parallel environments. The endpoint of each curve indicates the environment size after which simulation became unstable.

G Appendix - Hyperparameter Sweeps for Differentiability Experiments

In this appendix we briefly discuss the hyperparameter sweeps done in order to select hyperparameters for the results presented in Sec. 7.

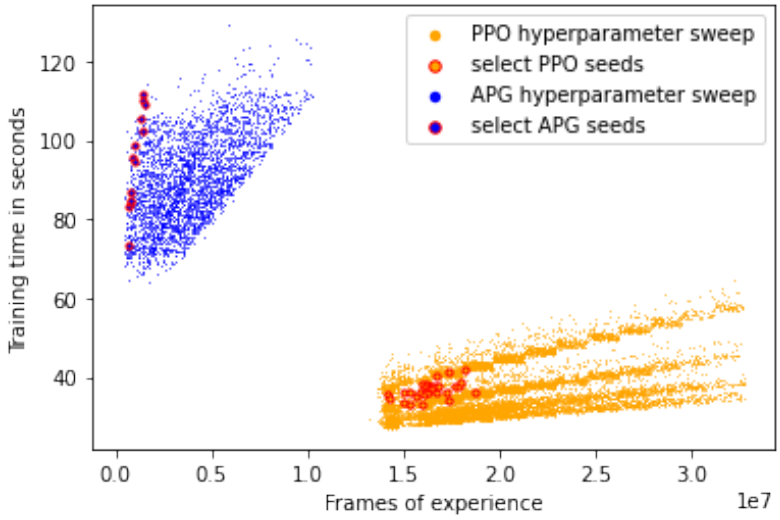

Figure 12: Each point on this plot represents a training run that reached a score of -1 on brax-reacherangle—a score that indicates the reacher arm can successfully navigate to all targets with high precision. The x-axis measures the number of environment frames required to reach a score of -1, and the y axis indicates the total wallclock time in seconds necessary to reach a score of -1. Points in red were used in the Figure data in Fig. 6, and represent different random seeds of the same set of hyperparameters. APG was run for 400 gradient steps total, and PPO was run for 40 million environment steps total.

Fig. 12 depicts the landscape of PPO and APG performance for brax-reacherangle. Points that are further to the left represent experiments that reached a score of -1 using fewer total environment frames, and points that are closer to the bottom represent experiments that reached a score of -1 faster, thus moving down and to the left is typically "better". Depicted this way, PPO and APG clearly trade off between training speed and sample efficiency on the pareto frontier of algorithms that solve this task.

At highest sample efficiency, PPO seems to require at least 15 million frames of experience, whereas APG can reliably train with less than a million total frames of experience.

PPO hyperparameter range:
num_envs: [256, 512, 1024, 2048]
learning_rate: loguniform from 3e-4 to 1
APG hyperparameters range:
num_envs: uniformly sampled from 2 to 256
learning_rate: loguniform from 3e-4 to 4e-3

While using larger learning rates typically achieves higher sample efficiency, exceeding the ranges used here resulted in instability.

H Appendix - License

Brax is released under an Apache License 2.0, and does not, to our knowledge, violate any copyrights. It will be hosted on github at `https://github.com/google/brax`.