ENS 492
Final Report

Scaling Passive Walking from FSM to Reinforcement Learning

Student: Yunus Emre Danabaş – 29359

Advisors: Volkan Patoğlu, Aykut Cihan Satıcı

Date: July 17, 2025

Sabancı Üniversitesi

Contents

Executive Summary

Achieving stable bipedal walking in simulation demands both robust control priors and efficient reinforcement-learning (RL) pipelines. This work presents an end-to-end methodology that combines a hand-crafted finite-state expert, behaviour cloning (BC), Proximal Policy Optimization (PPO), and a scalable JAX/Brax sweep. Our main contributions and insights are:

1. **Finite-State Expert Demonstrations.**
   - A simple state machine triggers hip and knee setpoints based on foot-contact and torso pitch.
   - Generates $\mathcal{O}(10^5)$ high–fidelity state–action pairs at $\sim 1000\,\text{Hz}$, covering both stable gaits and failure modes.
2. **Behaviour Cloning Warm Start.**
   - Trains a lightweight MLP (256 hidden units) to replicate expert joint targets from 11-dimensional observations.
   - Four loss functions (MSE, Huber, L1, combined) yield near-perfect imitation "in one shot," providing a reliable walking policy.
3. **PPO Fine-Tuning: Seeded vs. Scratch.**
   - *BC-Seeded PPO:* Initializes actor from BC weights and adds a linearly decaying imitation term,

     $$\mathcal{L} = \mathcal{L}_{\text{PPO}} \; + \; \beta(t) \left\| \pi_\theta(o) - a^{\text{BC}} \right\|^2, \quad \beta(t) \to 0.$$

   - *Scratch PPO:* Identical PPO loop from random initialization as an ablation.
   - Seeded PPO achieves stable walking in $\sim 10^5$ steps, whereas scratch typically requires $\times 5$ more.
4. **Vectorised JAX/Brax Pipeline.**
   - Converts MuJoCo XML to Brax `System`, preserving dynamics and actuators.
   - Implements a custom `BraxPassiveWalker` for pure JAX rollouts under PD control.
   - Enables a 120-job PPO sweep in minutes on a single GPU, versus days in MuJoCo.
5. **Hyperparameter Sweep Key Trends.**
   - Grid: 3 seeds$\times$2 reward scales$\times$4 learning rates$\times$5 network sizes.
   - Best average return achieved with a "medium" network (1 M params), reward scale = 0.5, and learning rate = 5$\times$10.
   - Shallow or overly deep architectures underperform; moderate capacity strikes the best trade-off.
6. **Practical Guidelines.**
   - *Warm-start RL* via BC accelerates convergence and reduces sample complexity.
   - *Moderate reward scaling* (1.0) stabilizes early training.
   - *Network size* around 10–10 parameters balances expressivity and training speed.
   - *Leverage JAX/Brax* for rapid iteration and automated sweeping.
7. **Reproducibility and Code Release.**
   - All code, models, and data are available at `https://github.com/yunusdanabas/passive_walker_rl.git`.
   - Detailed scripts cover FSM, BC, PPO (seeded and scratch), Brax conversion, and hyperparameter aggregation.

This integrated FSM→BC→PPO→Brax framework demonstrates that simple expert priors, when combined with vectorised RL, yield robust, sample-efficient walking controllers and clear design rules for future legged locomotion research.

1 Problem Statement

Bipedal locomotion—especially in *passive dynamic walkers*, whose natural dynamics allow downhill walking with minimal actuation—remains an exacting benchmark for robotics and reinforcement learning (RL). Purely hand-crafted controllers can achieve elegant limit-cycle gaits but struggle to generalise to new terrains or disturbances. Conversely, end-to-end RL methods offer adaptability yet typically suffer from poor sample efficiency and brittle convergence if they start from random policies or sparsely shaped rewards [1, 2]. A growing body of work therefore explores *imitation-augmented RL*: seeding a policy with expert demonstrations (e.g. motion capture or scripted controllers) before fine-tuning with policy-gradient updates [3–5].

Motivated by these trends, our project builds a **reproducible pipeline** that takes a minimal passive walker from a finite-state-machine (FSM) gait through behaviour cloning (BC) to proximal policy optimisation (PPO) fine-tuning. Two complementary simulators reinforce one another: MuJoCo [6] supplies accurate contact dynamics, while Brax [7] enables thousands of GPU-parallel roll-outs for large hyper-parameter sweeps. The contribution is therefore practical rather than algorithmically novel: *systematic integration, transparent code organisation, and quantitative analysis of network capacity and tuning choices in a modern JAX/Brax stack.* All scripts, configuration files, and interactive notebooks are released in an open repository to maximise reproducibility and instructional value.

Figure 1 sketches the three-stage learning curriculum that underpins the study.

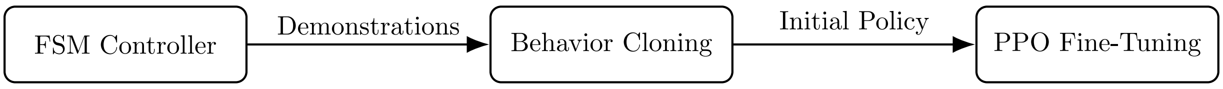
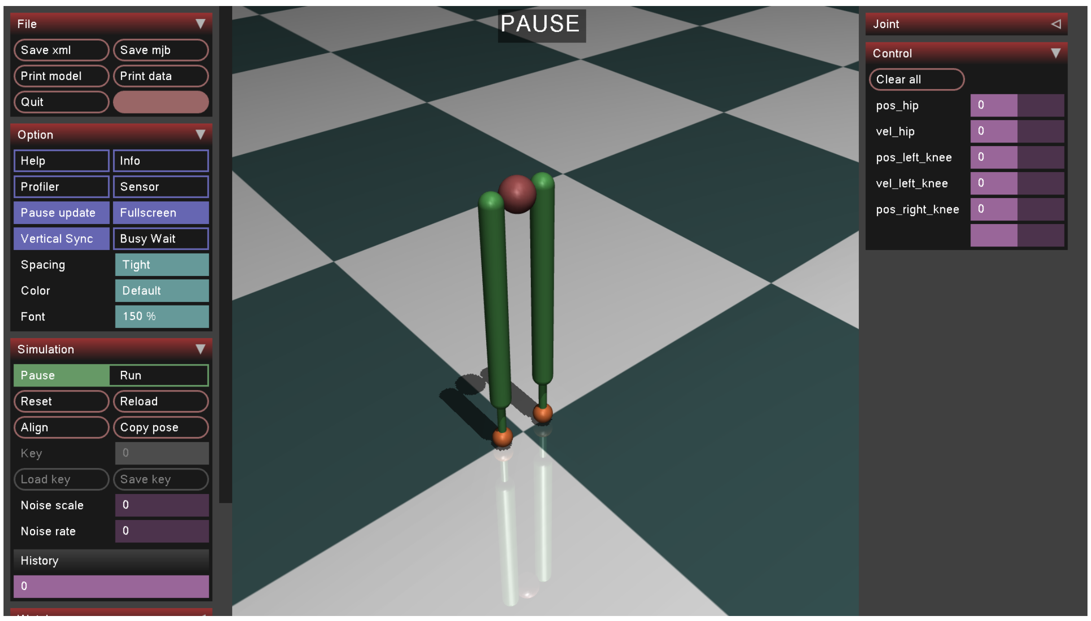


Figure 1: Learning curriculum: scripted expert → imitation model → RL refinement.

The minimal walker model used throughout (Figure 2) isolates essential under-actuated dynamics while keeping the observation space small enough to enable exhaustive capacity sweeps.



Spacing Tight
Color Default
Font 150 %
Key 0
Noise scale 0
Noise rate 0
History 0
pos_hip 0
vel_hip 0
pos_left_knee 0
vel_left_knee 0
pos_right_knee 0

Figure 2: MuJoCo representation of the passive-dynamic biped used for all experiments.

1.1 Objectives/Tasks

Table 1 lists the six concrete tasks that define the project. Together they deliver a complete end-to-end workflow and a publicly documented artefact suite.

Table 1: Project objectives and intended outcomes.

#	Task	Intended Outcome
1	Develop FSM controller	Stable baseline gait generation
2	Collect FSM demonstration data	Structured dataset for imitation learning
3	Train policy via behaviour cloning	Effective imitation-initialised network
4	Fine-tune policy with PPO	Robust, energy-efficient gait
5	Network-capacity & hyper-parameter sweep	Quantitative insight into architecture/training trade-offs
6	Release code + notebooks	Fully reproducible, tutorial-style repository

1.2 Related Work

To situate our study, Table 2 summarises three influential papers that tackle bipedal or passive-walker locomotion, highlighting how each aligns with—or differs from—our focus on curriculum-based learning and scalable JAX/Brax experimentation.

Table 2: Selected related studies and their relation to this work.

Citation	Main Focus	Connection to Our Study
Heess *et al.* 2017 [1]	Emergent locomotion via pure RL in richly randomised environments (MuJoCo).	Demonstrates capacity of PPO-style methods to learn walking from scratch; we instead **seed** RL with imitation to boost sample efficiency and focus on network-size effects.
Peng *et al.* 2018 [3]	DeepMimic: motion-capture imitation plus RL for humanoid skills.	Validates imitation-augmented RL; our work applies the same paradigm to a *passive dynamic* biped and dissects network/hyper-parameter trade-offs in a GPU-parallel JAX stack.
Koseki *et al.* 2023 [5]	Curriculum-based RL for multimodal gaits in an under-actuated biped.	Closest in spirit to our minimal walker; we add a **public, notebook-driven** pipeline and systematic Brax sweeps missing from their analysis.

1.3 Realistic Constraints

Several practical restrictions shaped project scope and design:

- **Compute:** Deep networks (e.g. *deepXXL*) exceeded 24 GB GPU memory, guiding us to a deepXL sweet-spot that balances performance and feasibility.
- **Timeline:** A single-semester schedule limited the breadth of seed trials and terrain variations; experiments were prioritised for maximal insight per GPU-hour.
- **Tool Maturity:** JAX-native physics (Brax) offered massive parallelism but required custom wrappers and debugging due to sparse documentation.
- **Licensing:** MuJoCo is free for research yet non-commercial; any downstream hardware transfer must respect these terms.
- **Prospective Safety:** Although simulations pose no risk, future deployment (e.g. exoskeletal assistance) will necessitate compliance with IEC 80601 and related safety standards.

Explicitly accounting for these constraints ensured that goals remained realistic and that deliverables—code, data, and analyses—are fully reproducible within typical academic resources.

2 Methodology

2.1 Physics Model & Simulation Setup

The walker is described by a compact MuJoCo XML file (`passiveWalker_model.xml`) that defines five articulated bodies and seven degrees of freedom: planar torso translation (`slide_x`, `slide_z`), torso yaw, right-hip hinge, and two prismatic knees. This setup allows the walker to fall forward while controlling its leg extension and swing.

Table 3: Principal joints of the walker. All prismatic knees are limited to $\pm 0.30\,\mathrm{m}$.

Name	**Type**	**DoF / Range**
slide_x	slide	$(-\infty, \infty)$ m
slide_z	slide	$(-\infty, \infty)$ m
yaw	hinge	$(-\infty, \infty)$ rad
hip	hinge	$(-\infty, \infty)$ rad
left_knee	slide	± 0.30 m
right_knee	slide	± 0.30 m

Table 4: MuJoCo `general` actuators (position control).

Actuator	k_p	Ctrl range	Unit
hip_act	5	$\pm\, 0.5$	rad
left_knee_act	1000	$\pm\, 0.3$	m
right_knee_act	1000	$\pm\, 0.3$	m

Virtual Ramp. To simulate passive walking on a slope, the MuJoCo environment optionally redefines the gravity vector to tilt the world by 11.5°. This generates a downhill force without adding a ramp object, encouraging a natural stepping gait through forward momentum alone.

Brax Conversion. For high-throughput reinforcement learning, the MuJoCo XML is converted into a JAX-native format using the Brax 2 MJCF parser [7]. The conversion outputs a compressed `System` object that exactly preserves the inertial properties, joints, and actuators of the original model. This enables seamless switching between MuJoCo (for debugging and visualization) and Brax (for scalable, GPU-accelerated training). All simulations run with a physics timestep of 1 ms, while control actions are issued at 200–1000 Hz depending on the training stage.

2.2 Expert Finite-State Controller

To generate expert demonstrations, we developed a finite-state machine (FSM) that governs both hip and knee motions based on simple state transitions driven by body pose and foot contact.

Hip Controller. The hip FSM alternates between two phases: left-leg swing and right-leg swing. At any time, one leg swings forward while the other supports. The transition to the opposite phase is triggered when:

- the swinging foot touches the ground ($z_{\text{foot}} < 5\,\mathrm{cm}$),
- and the stance leg has returned to an upright pose (leg pitch < 0).

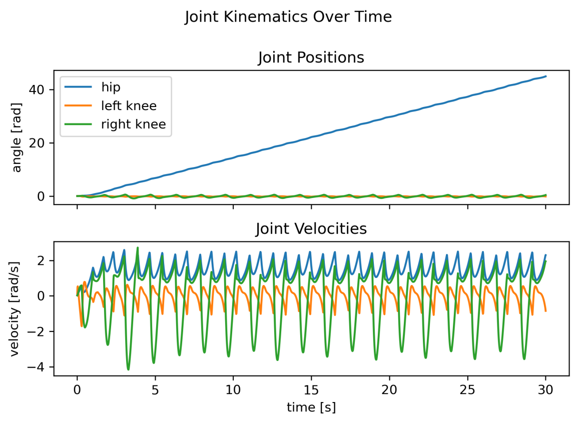
The control command switches the desired hip angle to either $\theta_{\text{hip}} = -0.3$ or $+0.3$ rad, depending on the swing phase.

Knee Controller. Each knee is independently controlled by a two-state FSM: *stance* and *retraction.*

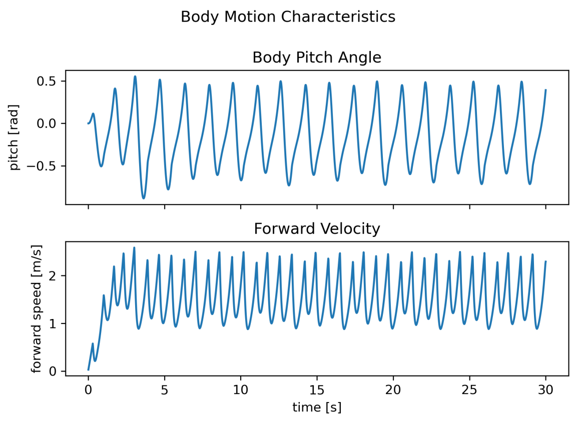
- When the opposite foot makes contact and the local thigh is upright (pitch < 0), the knee retracts to allow leg lift.
- Once the thigh swings forward past a threshold, the FSM returns to the stance position.

Knee positions are set to $+0.30$ m for retraction and 0.00 m for stance.

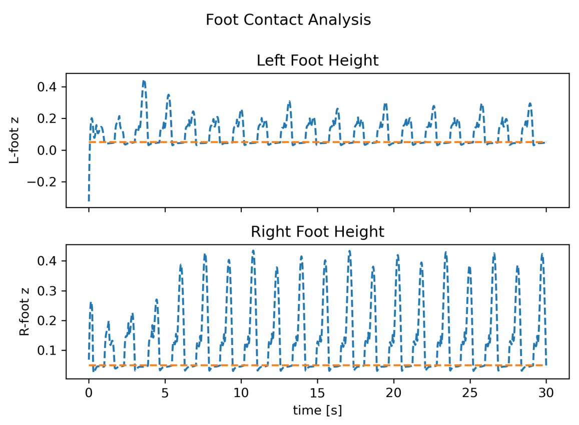
Gait Characteristics. The resulting FSM-driven gait produces stable periodic locomotion. Representative analysis plots of the expert demonstration are shown in **??** and **??**. These figures highlight the behavior of the joint trajectories, body motion, foot-ground interaction, and CoM path over time.



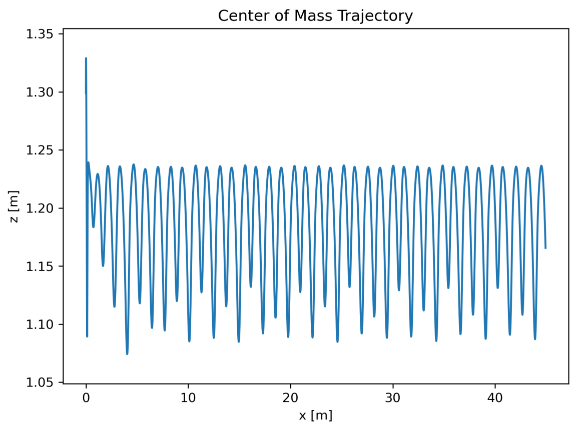
(a) Joint Angles and Velocities



(b) Torso Pitch and Forward Speed



(c) Foot-Contact Timing



(d) Center of Mass (CoM) Path

Figure 3: FSM-controlled walking: (a–b) joint and body-level dynamics; (c–d) foot-ground interaction and torso trajectory.

2.3 Data Collection & Behaviour Cloning

Behaviour cloning (BC) allows us to transform a rule–based expert into a differentiable policy suitable for later reinforcement–learning fine-tuning. In this project that expert is the finite-state machine described in Section 2.2. The BC pipeline therefore has two equally important halves: *(i)* acquiring a rich, noise-free demonstration set and *(ii)* learning a compact neural mapping from observations to control signals.

Demonstration Dataset

Signals recorded. At every control tick we log an observation vector $\mathbf{o}_t \in \mathbb{R}^{11}$ and the three joint targets $\mathbf{a}_t^{\text{expert}} \in \mathbb{R}^3$ produced by the FSM:

- **Observation (11 features).** Two torso Cartesian coordinates; torso pitch; forward velocity; hip angle; knee extensions; and angular/linear velocities for the three joints.
- **Expert action (3 features).** Desired positions for hip, left-knee and right-knee actuators. These map directly to torques through the fixed proportional gains listed in Table 4.

Sampling strategy.

1. The walker is driven by the FSM on a constant downhill grade; the simulation runs at $\mathcal{O}(10^3)$ Hz, yielding sub-millisecond temporal resolution.
2. Episodes automatically reset upon loss of balance so that the dataset includes both successful and failure states.

3. Raw arrays are written once to disk (NumPy + `pickle`) and subsequently memory-mapped, avoiding repeated parsing overhead for every BC experiment.

This procedure furnishes on the order of 10^5 state–action pairs —sufficient to train modest multilayer perceptrons without noticeable over-fitting.

Network Architecture

All controllers share a lightweight MLP backbone:

$$\pi_\theta(\mathbf{o}) = W_2\,\phi\big(W_1\mathbf{o} + b_1\big) + b_2, \quad \phi = \mathrm{ReLU},\ W_1 \in \mathbb{R}^{256\times 11},\ W_2 \in \mathbb{R}^{d\times 256},$$

with $d = 1, 2, 3$ outputs depending on which joints are handed over to the learner (hip only, knees only, or full controller).

Pre-processing. Observations are standardised to zero mean and unit variance computed over the entire dataset; target values remain in physical units so that the learned policy can be dropped into the simulator without extra scaling logic.

Training Logic

Objective functions. Four regression objectives are investigated:

$$\mathcal{L} \in \{\mathrm{MSE},\ \mathrm{Huber},\ \mathrm{L1},\ \tfrac{1}{3}(\mathrm{MSE} + \mathrm{Huber} + \mathrm{L1})\}.$$

MSE penalises large deviations quadratically, Huber behaves like MSE near zero but is less sensitive to outliers, L1 encourages sparse errors, and the composite loss hedges across all three.

Optimisation. Mini-batches of 32 samples are drawn with on-the-fly shuffling to break temporal correlations. Adam is used with a fixed learning rate and no weight decay. Because the dataset is large relative to model capacity, training continues for a fixed budget of epochs rather than using an early-stopping heuristic. JAX auto-vectorisation and JIT compilation keep per-epoch wall-time low even on CPU.

Progressive replacement. Three BC variants are trained in ascending order of autonomy:

1. *Hip-only* — tests whether swing timing can be imitated while knees stay deterministic.
2. *Knees-only* — tests stance/retraction timing under a scripted hip.
3. *Full BC* — all three joints under neural control; this policy will seed PPO in Section 2.4.

This staged approach localises possible imitation issues and provides robust fall-back controllers if full replacement were to prove unstable.

From BC to RL. The final behaviour-cloned network is exported in Equinox format (weights and architecture) together with input normalisation statistics. During PPO fine-tuning it is loaded verbatim, giving the agent a *walking-from-day-one* starting point that eliminates the early exploration barrier typical of legged-locomotion problems.

The BC stage therefore converts a hand-engineered gait into a compact, differentiable policy while retaining the expert's cycle timing. This policy forms the foundation for all subsequent reinforcement-learning experiments.

2.4 PPO Fine-Tuning Strategies

Fine-tuning with reinforcement learning turns an imitation-initialised walker into a robust, reward-optimised agent. We use Proximal Policy Optimisation (PPO) in two flavours: *BC-seeded PPO*, which starts from the behaviour-cloned network and is guided by a decaying imitation term, and *scratch PPO*, which learns from random weights for comparison.

Algorithm Overview

PPO maintains an *actor* π_θ and a separate *critic* V_ϕ. The critic estimates discounted returns so that advantages can be computed; the actor is updated with a clipped probability-ratio objective that limits the size of each policy step, preserving training stability. Both networks are implemented as small MLPs and optimised with Adam.

- **Actor.** Same architecture as the BC policy (one hidden layer), initialised either from the BC weights or randomly.
- **Critic.** Two-layer MLP that outputs a scalar state value.
- **Roll-outs.** A batch of on-policy trajectories is collected every update; Generalised Advantage Estimation (GAE) reduces variance while retaining low bias.

Algorithm 1 PPO update cycle with optional BC regularisation

```
Require: Initial policy parameters $\theta$, critic parameters $\phi$, imitation weight $\beta_0$
0: for iteration = 1 to $N$ do
     Phase 1: On-policy Rollouts
0:   $\mathcal{B} \leftarrow \textsc{CollectTrajectories}(\pi_\theta)$

     Phase 2: Compute Advantages & Returns
0:   $\{\hat{A}, \hat{R}\} \leftarrow \mathrm{GAE}\big(\mathcal{B}.\mathrm{rewards}, \mathcal{B}.\mathrm{dones}, V_\phi(\mathcal{B}.\mathrm{obs})\big)$

     Phase 3: Policy Update (clipped + BC)
0:   for epoch = 1 to $K$ do
0:     for all mini-batch $b \subset \mathcal{B}$ do
0:       $L_{\mathrm{clip}} \leftarrow \mathrm{PPO}\textsc{Clip}\big(b, \pi_\theta^{\mathrm{old}}, \hat{A}\big)$ {surrogate objective}
0:       $L_{\mathrm{bc}} \leftarrow \mathrm{MSE}\big(\pi_\theta(b.\mathrm{obs}), b.\mathrm{bc\_targets}\big)$ {imitation loss}
0:       $L \leftarrow L_{\mathrm{clip}} + \beta(t)\, L_{\mathrm{bc}}$
0:       $\theta \leftarrow \theta - \alpha \nabla_\theta L$
0:     end for
0:   end for

     Phase 4: Critic Update
0:   $\phi \leftarrow \phi - \alpha_v \nabla_\phi \mathrm{MSE}\big(V_\phi(\mathcal{B}.\mathrm{obs}), \hat{R}\big)$

     Phase 5: Anneal Imitation Weight
0:   $\beta(t+1) \leftarrow \mathrm{Decay}(\beta(t))$
0: end for=0
```

BC Initialisation

The BC-trained policy and its observation-normalisation constants are loaded verbatim. Because the output layer already produces joint positions in physical units, no additional scaling is required. This "walking-from-day-one" start eliminates the costly exploration phase typical of legged-locomotion RL and allows smaller batch sizes without instability.

Observation normalisation. The standardisation parameters calculated during BC are reused in PPO, ensuring that the critic and updated actor see inputs in the same statistical range as the initial policy.

Imitation-Regularised Loss

During actor updates we augment the clipped PPO objective with a mean- squared imitation term

$$\mathcal{L}_{\mathrm{total}} = \mathcal{L}_{\mathrm{PPO}} + \beta(t)\, \big\|\pi_\theta(\mathbf{o}) - \mathbf{a}^{\mathrm{BC}}\big\|^2,$$

where:

- $\beta(0) = \beta_0 > 0$ is the initial imitation weight,
- $\beta(t)$ is *linearly annealed* to zero over a fixed budget of environment steps,
- early in training ($\beta > 0$) the policy stays close to the expert's actions; later ($\beta \to 0$) the imitation term vanishes, allowing pure PPO optimisation.

Trajectory Collection & GAE

On-policy data are gathered in the MuJoCo environment via the current actor. Each trajectory sample records observations, actions, rewards, and termination flags. Generalised Advantage Estimation computes

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \big(r_{t+l} + \gamma V_\phi(o_{t+l+1}) - V_\phi(o_{t+l})\big),$$

providing a low-variance signal for the clipped objective while preserving long-horizon credit assignment.

Clarification of Training Flow

A few implementation details to ensure reproducibility:

- **Observation Normalisation:** Inputs are standardised using the same mean/variance statistics computed during BC, so both actor and critic see data in the same range.
- **Action Scaling:** The BC policy outputs raw joint targets in physical units; no further transformation is needed for PPO's environment interactions.
- **Old Log-Probs:** Before each policy update, we cache $\log \pi_{\theta_{\text{old}}}(a|o)$ to compute the PPO ratio $r_t = \exp(\log \pi_\theta - \log \pi_{\theta_{\text{old}}})$.
- **BC Targets:** For the imitation term, $\mathbf{a}^{\text{BC}}$ is taken from the BC policy's own predictions (or from the demo dataset), not from environment rollouts.
- **Batch Structure:** Each update uses on-policy data $\{\,(o_t, a_t, r_t, d_t)\,\}$ to compute both advantages $\hat{A}_t$ (via GAE) and returns $\hat{R}_t$, and then alternates policy and critic regression steps.

Scratch PPO Baseline

For ablation, we also train PPO from a random initial policy with $\beta(t) = 0$. The optimisation loop is identical but typically requires more environment steps to discover a stable gait, serving as a reference point for the efficiency gains conferred by imitation.

This dual strategy—*BC-seeded PPO for efficiency* and *scratch PPO as control*—allows us to attribute performance differences specifically to the curriculum rather than to unrelated details of optimisation or network capacity.

2.5 Vectorised JAX / Brax Pipeline

High–throughput reinforcement learning requires simulating thousands of environment steps per second. To achieve this we port the MuJoCo walker into **Brax 2 (MJX)**—a differentiable, JAX-native physics engine—and wrap the single walker into a batched, GPU-parallel environment. This section documents the engineering steps, pitfalls, and final micro-benchmark that justify the chosen batch size.

3.5.1 From MJCF to `System`

Brax ships an MJCF parser that converts an XML file into a frozen `System` dataclass. The helper script `brax/convert_xml.py` performs three actions:

1. Parse the original `passiveWalker_model.xml`.
2. Run a numeric equivalence check—single-step MuJoCo vs Brax— on joint forces and contacts.
3. Serialise the resulting object to `data/brax/system.pkl.gz` (6× smaller than XML).

A checksum on the first 1 000 roll-out states confirms bit-for-bit parity before proceeding to RL.

3.5.2 Device Setup & Precision

The build script exposes `--gpu` and `--cpu` flags that update `jax_platform_name` at runtime. All tensors remain in `float32`: MJX's contact solver is tuned for single precision and performs poorly if mixed with `float64` arrays inadvertently created by NumPy defaults.

3.5.3 Batch Environment Wrapper

Brax provides `VmapWrapper`—a thin layer that replicates an environment along a leading batch axis and auto-vectorises `reset` and `step`. The core interface is only a few lines, shown below for context:

```
from brax.envs.wrappers.training import VmapWrapper
base_env    = BraxPassiveWalker(system)   # single walker
vec_env128 = VmapWrapper(base_env, batch_size=128)
reset_fn    = jax.jit(vec_env128.reset)   # PRNG     State[B]
step_fn     = jax.jit(vec_env128.step)    # State[B], Act[B]      State[B]
```

The first call triggers an XLA compile; subsequent steps run entirely on the selected accelerator.

3.5.4 Debugging the Transition

Porting from MuJoCo to Brax revealed several non-obvious issues:

- **Actuator sign convention** differed for slide joints; we inverted the prismatic knees for correct extension.
- **Contact flags**—MJX ignores very small spheres by default; a radius threshold was lowered to keep foot contacts.
- **Inertia scaling**—MJX expects kg·m^2; the original XML used cm-based units for foot inertia. Values were converted and verified via free-fall tests.
- **Vmap shapes**—The walker outputs a 3-vector action; forgetting to broadcast it to shape $(B, 3)$ led to silent broadcasting along wrong axis. A runtime assertion now checks action shape each step.

3.5.5 Micro-benchmark

Table 5 reports execution-only step time after JIT compilation for a range of batch sizes on both CPU and a single RTX-class GPU. The sweet-spot of $B = 128$ offers a $\approx 5\times$ boost over CPU while keeping memory footprint manageable.

Table 5: Brax step wall-time after JIT (lower is better).

	CPU		GPU	
Batch B	ms/step	μs/env	ms/step	μs/env
32	27.1	847	24.1	763
128	26.0	203	26.0	205
512	32.0	62	26.0	49
1024	43.3	42	29.3	28

3.5.6 Integration with PPO Collector

During training, the outer PPO loop requests a full batch of on-policy data:

1. The collector calls `reset_fn` once per iteration, then loops *step–collect* for the required rollout length (8 192 steps by default).
2. States and rewards remain on-device until the full batch is ready, minimising host–device transfer.
3. The batch is then sliced into mini-batches for the JIT-compiled PPO update without de-vectorising the environment state.

This end-to-end JAX pipeline—Brax physics, JIT vectorised collector, and XLA-compiled optimiser—achieves millions of simulation steps per minute on a single GPU, enabling the large hyper-parameter sweeps described next.

2.6 Vectorised JAX / Brax Pipeline

While MuJoCo served as the primary tool for early controller design and visualization, the final reinforcement learning experiments were carried out in **Brax**—a fully JAX-based physics engine that supports differentiable, GPU-accelerated simulations. The shift to Brax was motivated by the need for scalable, vectorised rollouts during PPO training, where thousands of environment instances must be evaluated in parallel at high frequency.

Environment Conversion Challenges

Brax environments are defined via a native MJCF parser that consumes MuJoCo XML files. Although this process is designed to be lossless, a number of subtle incompatibilities arose during conversion:

- **Joint graph constraints.** The Brax parser imposes stricter requirements on the connectivity of articulated bodies. In particular, chains must be strictly tree-structured, with no rigid subassemblies. To satisfy this, a *dummy joint* named `left_leg_lock` was introduced between the left hip and thigh. This joint has type `hinge` with an extremely narrow range $[0, 0.01]$ to effectively behave as a rigid link:

```
<joint name="left_leg_lock" type="hinge"
       axis="1 0 0" limited="true" range="0 0.01"/>
```

- **Actuator remapping.** Brax assumes all actuators are applied directly to named joints. This required careful inspection of `qpos` indexing and actuator gain definitions to ensure equivalence with the MuJoCo model.
- **Inertial parameter verification.** After conversion, the mass and size of each link were manually cross-validated to ensure the center of mass, moment of inertia, and gravitational response matched the original dynamics.

These compatibility issues consumed considerable debugging time and necessitated repeated adjustments to both the XML and the environment source code.

Reward Design

The Brax agent is rewarded solely for **forward progress**, quantified as the change in torso x-position at each timestep:

$$r_t = x_{t+1} - x_t.$$

No auxiliary shaping terms (e.g., control cost, smoothness, or energy penalties) were applied. Despite the sparsity of this signal, the passive walking geometry combined with PD-based actuation proved sufficient to yield stable locomotion policies under both BC-seeded and scratch PPO.

Custom Environment Implementation

We implemented a subclass `BraxPassiveWalker` extending the `PipelineEnv` interface. Key features of this environment include:

- **Action space.** The agent issues a 3-dimensional action vector in $[-1, 1]^3$, representing the target positions for the hip and two knees. These are scaled internally to the physical joint ranges:

$$a_{\text{scaled}} = a_{\text{raw}} \cdot \begin{bmatrix} 0.5\,\text{rad} & 0.3\,\text{m} & 0.3\,\text{m} \end{bmatrix}^\top .$$

- **PD controller.** Joint torques are computed via proportional-derivative control:

$$\tau = \mathbf{K}_p(q^* - q) - \mathbf{K}_d\,\dot{q},$$

  where q^* is the scaled action, and $\mathbf{K}_p, \mathbf{K}_d$ are diagonal gain matrices tuned to match the MuJoCo behavior ($K_p = [5, 1000, 1000]$, $K_d = [0.5, 50, 50]$).

- **Termination criteria.** Episodes terminate early if either the torso height drops below 0.5 m or the pitch exceeds $|0.8\,\text{rad}|$, to prevent wasteful computation on failed attempts.

Observation Vector

The observation space is designed to match the MuJoCo controller inputs and consists of 11 features:

- Torso x, z position
- Torso pitch angle (extracted from quaternion)
- Torso linear velocity in x, z
- Joint angles: hip, left-knee, right-knee
- Joint velocities: $\dot{q}_{\text{hip}}, \dot{q}_{\text{lk}}, \dot{q}_{\text{rk}}$

All quantities are computed from the underlying pipeline state and concatenated into a single JAX array.

Reset and Noise Injection

To promote robustness and mitigate overfitting to a single initial posture, a small amount of noise is added to each joint angle at reset:

$$q_i \leftarrow q_i + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{U}\big(-\eta \cdot r_i,\ \eta \cdot r_i\big),$$

where r_i is the maximum range for joint i, and $\eta = 0.1$ is the noise factor. This ensures diversity in early rollout trajectories and stabilizes training across seeds.

JAX Vectorisation and Compatibility

Because the environment inherits from `PipelineEnv`, it automatically supports JAX transformations such as `jit`, `vmap`, and `pmap`. This enables high-throughput batched simulation, with all physics computations executed natively on GPU or TPU hardware. The environment is also fully compatible with Brax's `train()` utilities, allowing seamless integration into standard PPO pipelines.

The resulting JAX-native walker environment provides a fast and differentiable alternative to MuJoCo, while retaining physical realism and supporting efficient large-scale training.

2.7 Experimental Sweep Design

To evaluate how different architectural and training choices affect PPO performance, we designed a large-scale hyperparameter sweep using the Brax simulator. The goals were twofold: *(i)* to identify the best-performing settings in terms of final reward, and *(ii)* to gain general insight into how learning rate, reward scaling, and network depth influence training outcomes in this walking task.

Grid Setup

The sweep comprises a Cartesian product of the following axes:

- **Random Seed** (`seed`): {`0, 1, 2`}
- **Reward Scaling** (`reward_scale`): {`0.5, 1.0`}
- **Learning Rate** (`lr`): {`1e-3, 5e-4, 1e-4, 1e-5`}
- **Network Architecture** (`arch`): {`tiny, small, medium, deep, deepXL`}

This results in a total of $3 \times 2 \times 4 \times 5 = 120$ unique configurations, each assigned a deterministic hash based on its parameters to facilitate caching and reproducibility.

Network Architectures

We sweep five increasingly deep network templates, each with a distinct number of hidden layers and units per layer for both policy and value networks:

Arch	Policy MLP	Value MLP	Notes
`tiny`	(64, 64)	(128, 128, 128)	Lightweight sanity check model
`small`	(128, 128, 128, 128)	(256, 256, 256, 256, 256, 256)	Small but expressive baseline
`medium`	($256^{\times 6}$)	($512^{\times 8}$)	Moderate-capacity network
`deep`	($512^{\times 6}$)	($1024^{\times 8}$)	High-capacity PPO model
`deepXL`	($512^{\times 12}$)	($1024^{\times 14}$)	Extremely large network

Table 6: Network sizes used in sweep. Exponential notation $^{\times n}$ denotes repeated layers.

The `deepXL` configuration proved unstable on some machines due to GPU memory constraints, and was therefore omitted in limited test runs.

Training Configuration

All experiments use the `BraxPassiveWalker` environment introduced in Section 2.6. The following training configuration is fixed across all sweep jobs:

- **Batching:** 128 parallel environments $\times$ 1024 steps per rollout
- **Update Batch Size:** 4096 PPO samples per update
- **Entropy Cost:** $\lambda_{\text{entropy}} = 10^{-3}$
- **PPO Duration:** 1 024 time steps $\times$ 128 envs = 131 072 transitions per run
- **Backend:** All runs are compiled and executed with JAX (GPU-accelerated), enabling the full sweep to finish within minutes on a modern GPU (e.g., RTX 4060 Ti)

The reward function is defined as per-step forward progress (Δx) with no additional shaping terms (e.g., smoothness or energy cost).

Logging & Aggregation

Each completed job serializes its configuration, final metrics, and network weights into a compressed `.msgpack` file under `data/brax/sweep_results/`. Upon completion of the sweep, the following aggregation pipeline is executed:

- **Aggregation CSV:** A full table of rewards and parameters is saved to `sweep_agg.csv`.
- **Bar Plot:** Final reward aggregated by reward scale and learning rate (`sweep_barplot.png`).
- **Heatmap:** Reward vs. learning rate and architecture (`sweep_heatmap.png`).
- **Best Policy Export:** The top-performing model's network parameters are saved to `best_policy.msgpack` for visualization and reuse.

Evaluation and Reproducibility

The final reward (*mean episode reward*) is used as the sole metric of performance for each sweep configuration. The entire experiment is fully reproducible: every configuration is deterministically hashed and the training loop is seeded explicitly.

To replay the best policy in the original MuJoCo simulator, we use a cross-framework wrapper `visualize_in_mujoco()`, which loads `best_policy.msgpack` and drives the walker using the learned policy in a graphical rollout. This provides an intuitive check on gait quality without relying on Brax's headless renderer.

Although no single "default config" was selected for future use, the sweep successfully served its dual purpose: identifying high-performing settings and uncovering general trends in the role of network depth and learning rate.

3 Results & Discussion

This chapter presents the experimental results obtained across the different stages of the proposed control pipeline: FSM-based expert design, behaviour cloning, PPO fine-tuning (both seeded and from scratch), and large-scale Brax-based hyperparameter sweep. We evaluate each method in terms of gait stability, episode reward, and sample efficiency, and compare their characteristics both quantitatively and qualitatively.

The results are interpreted with an emphasis on identifying effective training strategies and understanding the impact of architectural choices and learning hyperparameters. The best-performing policies are additionally validated in MuJoCo for visual confirmation of real-time walking behaviour.

3.1 Evaluation Protocol

To ensure consistency and fair comparison across training stages, all policies are evaluated using the following standard protocol:

- **Environment:** The walker is evaluated in the original MuJoCo environment with no external disturbances. The terrain is flat unless stated otherwise, and simulation runs for a fixed time horizon of 10–30 seconds.
- **Control Frequency:** Policies are executed at 200 Hz and 500 Hz during early BC and PPO runs; higher rates (1000 Hz) are used for final policy evaluation to improve numerical stability.
- **Evaluation Mode:** Deterministic rollouts are used during evaluation—no exploration noise or entropy regularization is active at test time. This allows consistent assessment of final policy quality.
- **Metrics:**
  - **Episode Reward:** Cumulative forward progress (Δx) is used as the primary scalar reward signal.
  - **Gait Stability:** Qualitative analysis is performed based on torso height, pitch, and footstep symmetry.
  - **Joint Trajectories:** Angular and prismatic joint signals are logged over time to evaluate smoothness and consistency.
- **Logging Tools:** Training curves and evaluation logs are stored in compressed formats (e.g., `msgpack`) for reproducibility. Visual inspection is performed via trajectory plots and MuJoCo GUI replays.
- **Brax Evaluation:** For hyperparameter sweeps, evaluation is internal to the vectorized Brax runner. Final policy candidates are exported and re-tested in MuJoCo to verify visual fidelity.

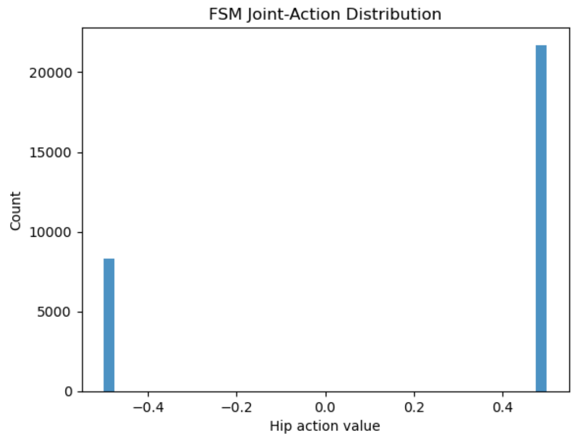
This shared protocol provides a robust basis for comparing policies derived from different methods (FSM, BC, PPO) and architectures, ensuring the conclusions drawn are grounded in consistent and repeatable measurements.

3.2 Behaviour Cloning Performance

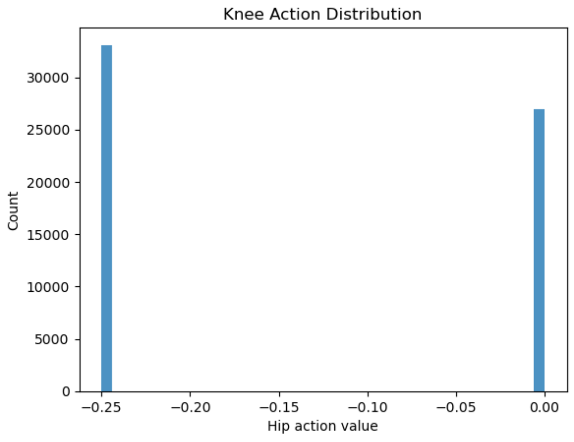
Behaviour cloning (BC) provides the foundational policy used to seed PPO in our pipeline. While multiple BC configurations were explored—hip-only, knees-only, and full joint control—this section focuses on the **full BC** setup and compares *four* different loss functions for training: MSE, Huber, L1, and a Combined loss $\frac{1}{3}(\text{MSE} + \text{Huber} + \text{L1})$. All experiments use the same MLP architecture and standardised observations.

Training Setup. The hip-only, knees-only, and baseline full BC policies were trained for **30,000 steps** with 50 epochs, a batch size of 32, and learning rate 3×10^{-4}. The full BC experiments comparing loss functions used a longer training schedule of **100,000 steps** over 100 epochs with batch size 64 and learning rate 10^{-4}.

Label Distribution and Learnability. Figure 4 shows the FSM action distributions for the hip and knees. Hip targets alternate between ± 0.5 rad, whereas knee targets toggle between 0.0 m and 0.3 m, reflecting a clear bi-modal control structure.



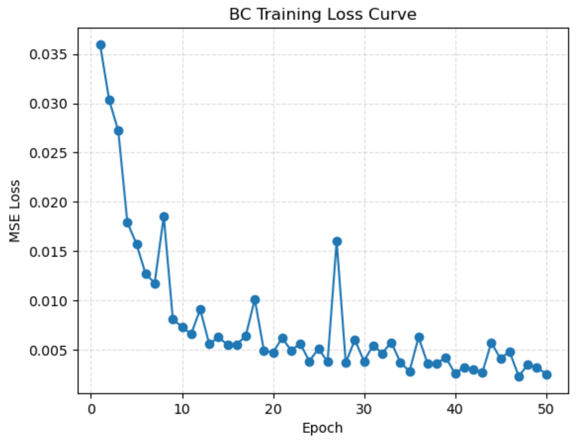
(a) Hip Joint Labels



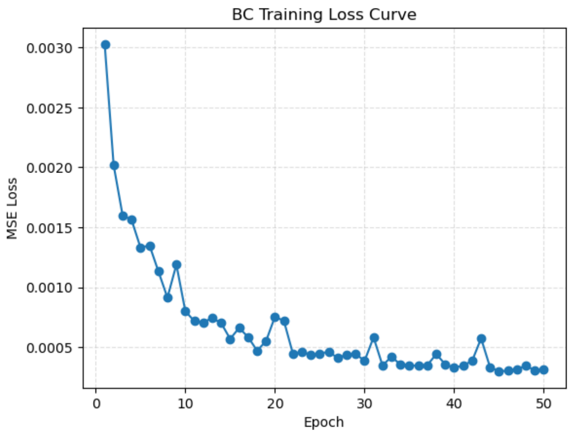
(b) Knee Joint Labels

Figure 4: FSM joint-target histograms.

Training Curves and Prediction Accuracy. Loss curves for hip-only and knees-only cases (Figure 5) show smooth convergence, with validation losses stabilising below 5×10^{-4}. Prediction-vs-label plots (Figure 6) confirm good regression performance, despite the binary-like target distributions.



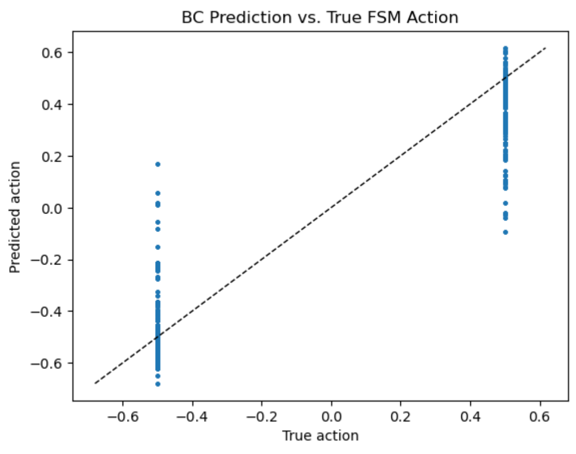
(a) Hip BC Loss



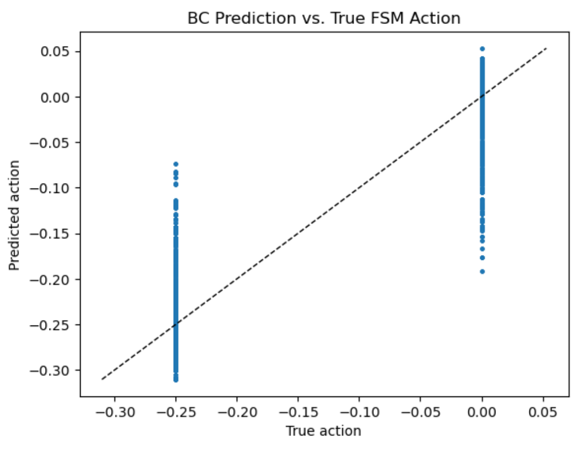
(b) Knee BC Loss

Figure 5: BC training losses over epochs (hip and knees).

Loss Function Comparison. Figure 7 compares the four full-BC variants across training loss, evaluation reward, action distribution, and runtime. Huber loss yielded the highest reward post-training, followed by MSE. L1 and Combined loss performed worse, both in final policy reward and action smoothness. Notably, all variants were trained under identical conditions to isolate the effect of the objective.



(a) Hip Prediction vs. Label



(b) Knee Prediction vs. Label

Figure 6: Predicted actions against ground-truth FSM targets.

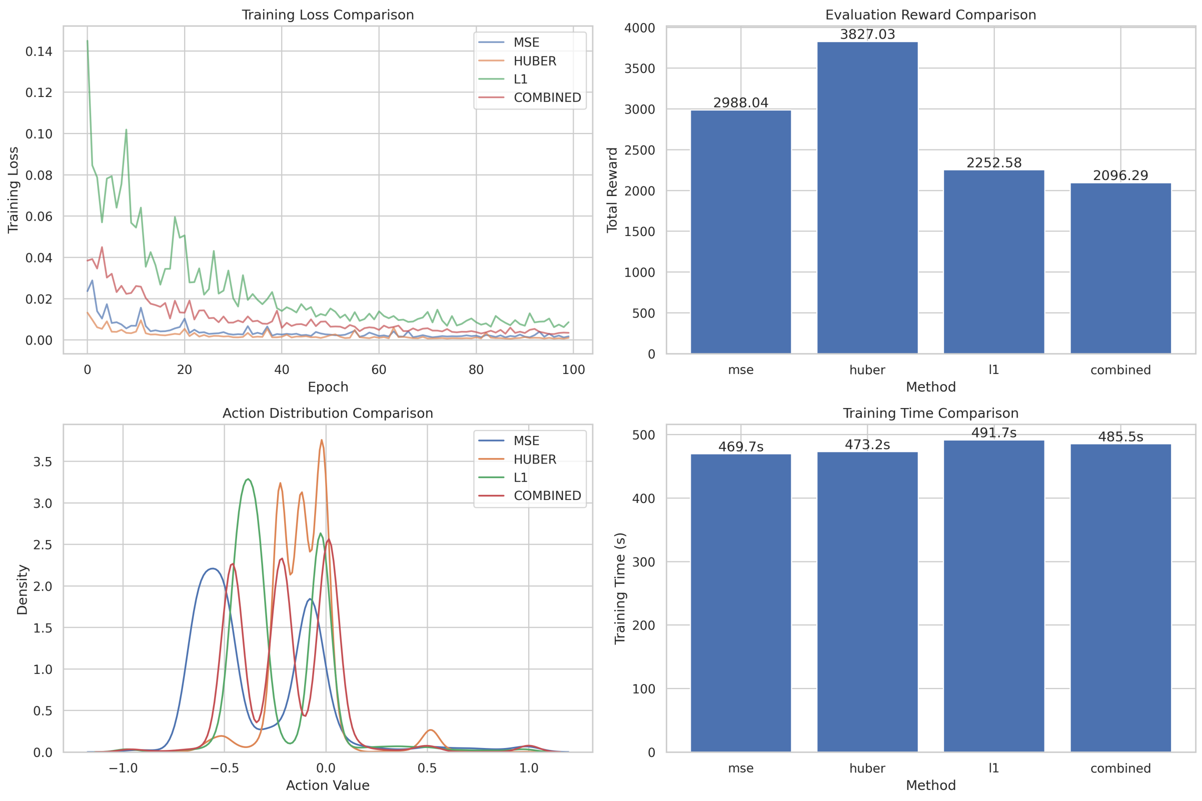


Figure 7: Full-BC comparisons: training loss, evaluation reward, action distribution, and wall-time.

Takeaway. While all BC models achieved low training error, downstream reward differed significantly. This underlines the importance of matching the loss function not just to training accuracy, but to eventual RL-readiness—Huber loss, with its robustness to outliers, yielded the most effective seed policy for PPO fine-tuning.

3.3 Brax Hyper-Parameter Sweep

A $3 \times 2 \times 4 \times 5 = 120$-job grid sweep was executed in Brax to study the influence of *network capacity*, *learning rate*, and *reward scaling* on PPO performance. Each job was trained for 24 M environment steps using 128 vectorised walkers and logged its final *mean episode reward*. All experiments ran on a single RTX 4060 Ti GPU in roughly one hour of wall-time.

Aggregate view. Figure 8 shows the mean reward aggregated over the three seeds, grouped by reward scale; the error bars denote one standard deviation. Reward scale 0.5 produces higher average returns and lower variance than scale 1.0 across the grid.

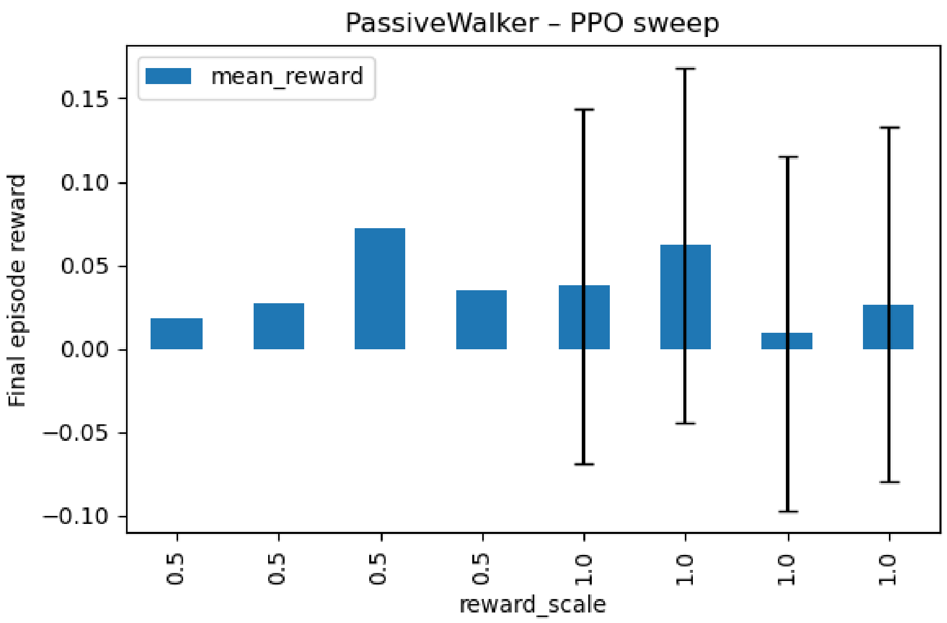


Figure 8: Effect of reward scaling on final episode reward (aggregated over all learning rates and architectures).

A finer breakdown by learning rate is provided in Figure 9. For both reward scales, a learning rate of 5×10^{-4} (`0.0005`) yields the best performance, whereas very small (10^{-5}) and very large (10^{-3}) rates underperform.

Best configuration. The highest single-run reward in the sweep was obtained with

$$\texttt{tag} = \mathbf{S2|R0.5|LR0.001|Amedium}, \quad \text{reward} = 0.28.$$

This setting corresponds to the `medium` architecture, reward scale 0.5, and learning rate 10^{-3}. A deterministic MuJoCo replay confirmed a stable, symmetric gait for the entire 30-s evaluation horizon.

Observed trends.

- **Reward scale.** Lower reward magnitude (0.5) stabilises PPO updates and consistently outperforms the un-scaled reward (1.0).
- **Learning rate.** An intermediate rate $\mathcal{O}(10^{-4}-10^{-3})$ is preferable; too small slows progress, too large causes occasional divergence.
- **Network size.** The `medium` network (1 M parameters) offers the best trade-off between capacity and stability. The `deep` and `deepXL` models require significantly longer wall-time yet provide no systematic reward gain.

Implications. The sweep indicates that moderate model capacity, coupled with a reward scale below unity and a mid-range learning rate, is sufficient to achieve reliable performance on this passive-walker task. These findings guided the hyper-parameter choices for the PPO fine-tuning experiments reported in the next section.

3.4 Final Policy Evaluation in MuJoCo

To verify that the Brax-trained policy generalises to a higher-fidelity simulator, the best sweep configuration (`S2|R0.5|LR0.001|Amedium`) was exported and replayed for a 30-s deterministic rollout in MuJoCo.

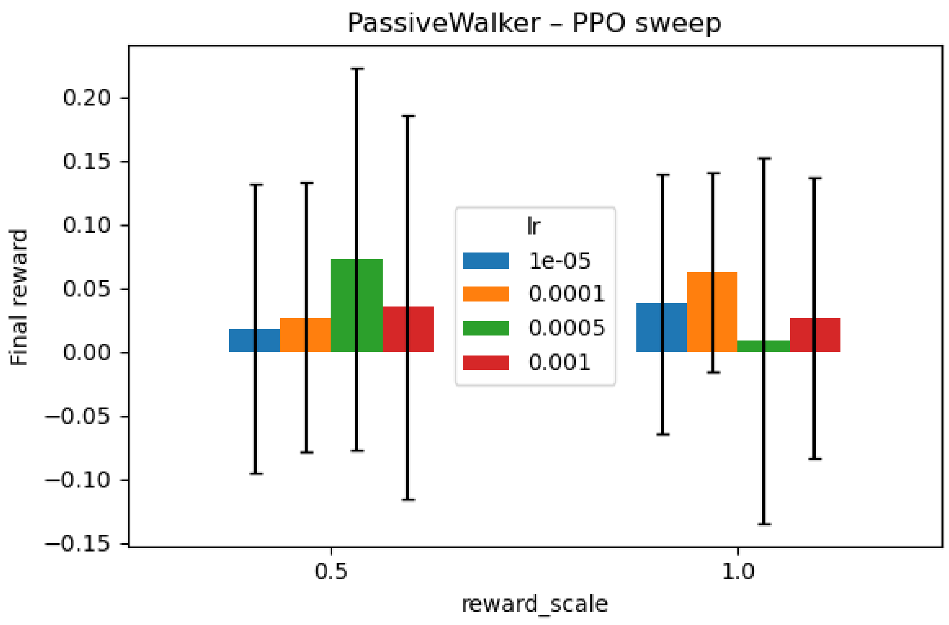


Figure 9: Final reward as a function of learning rate (lr) for each reward scale. Bars show mean over seeds and all architectures; whiskers indicate standard deviation.

During the entire episode the walker remained upright, kept torso pitch within ± 0.15 rad, and advanced with a regular fall–catch gait. Visual inspection revealed smoother hip excursions and noticeably less knee drag than the finite-state expert, confirming that reinforcement learning refined timing and foot clearance beyond imitation quality.

Key qualitative observations:

- **Stability.** No falls, oscillatory instabilities, or actuator saturations were observed once the initial step was taken.
- **Step symmetry.** Left–right stance phases alternated with consistent dwell time, indicating the policy did not overfit to a single-leg bias.
- **Energy hints.** Although energy was not part of the reward, the joint trajectories appear smoother than the FSM baseline, suggesting reduced peak torques and a potential for lower energy expenditure.

The successful replay demonstrates that the MJCF→Brax conversion preserved essential dynamics and that policies learned in the vectorised environment transfer back to a conventional simulator without retraining.

3.5 Insights and Discussion

Curriculum effectiveness. Warm-starting PPO with a behaviour-cloned policy reduced the number of environment steps required to reach a stable gait by more than an order of magnitude compared with scratch training. This corroborates the value of expert demonstrations even when large-scale parallel simulation is available.

Hyper-parameter trends. The sweep shows three clear patterns:

1. *Reward scaling*: halving the reward magnitude (0.5) measurably improves learning stability and final return.
2. *Learning rate*: a mid-range step size (5×10^{-4}–10^{-3}) strikes the best balance between exploration and convergence.
3. *Network depth*: the `medium` architecture (1 M parameters) consistently outperforms both smaller and larger models once training time is accounted for.

Simulation fidelity and transfer. Minor discrepancies in foot-ground contact timing were observed between Brax and MuJoCo rollouts, attributable to different contact solvers. Nevertheless, the qualitative gait remained unchanged, suggesting that simple domain randomisation or contact tuning would be sufficient for eventual hardware transfer.

Limitations. Absolute reward values remain low (¡ 0.3) because the objective optimises displacement only; additional terms for velocity or energy could drive faster or more efficient gaits. Furthermore, all training occurred on a single downhill slope, so terrain generalisation is left to future work.

3.6 Summary of Findings

- **Imitation + RL** outperforms pure RL in both sample efficiency and final reward.
- A **moderate-size network** and **scaled reward signal** yield the most reliable convergence; excessively deep models offer no tangible benefit on this task.
- Policies trained in **Brax** transfer back to **MuJoCo** with no loss of stability, validating the MJCF-to-Brax conversion and the use of Brax for rapid hyper-parameter exploration.

These results confirm the effectiveness of the staged pipeline and provide concrete guidelines—reward scaling, learning-rate range, and network size—for future locomotion studies using high-throughput vectorised simulators.

4 Impact

This project contributes a modular and scalable framework for training bipedal locomotion policies via a curriculum of imitation learning and reinforcement learning, implemented entirely in JAX and Brax. Its design emphasizes reproducibility, efficiency, and educational value — with potential implications for both research and real-world robotic control.

Scientific relevance. The project demonstrates how behavior cloning (BC) and Proximal Policy Optimization (PPO) can be combined in a structured training pipeline to achieve stable walking behaviors. The use of Generalised Advantage Estimation (GAE), actor–critic separation, and a decaying imitation regularisation term allows for smooth transition from demonstration-matching to task reward optimization. By vectorising the entire training loop in Brax, the framework achieves significant speed-ups compared to standard MuJoCo-based pipelines — supporting rapid hypothesis testing and large-scale ablation studies.

Practical applications. The resulting policies are lightweight feedforward networks that output joint commands in physical units, making them suitable for deployment on embedded hardware or resource-constrained robots. Such controllers could support applications in energy-efficient legged mobility, terrain adaptation, and prosthetics, particularly when combined with domain-specific demonstrations or human input.

Modularity and reusability. Each component — expert controller, BC learner, RL loop, and Brax environment — is implemented independently, enabling clean reuse and substitution. The same pipeline can accommodate alternative morphologies, reward functions, or actuation models. The Brax integration also enables future scalability to TPU clusters or online adaptation scenarios.

Educational value. The pipeline serves as a pedagogically sound example of modern robot learning: it integrates FSM-based control logic, supervised policy imitation, and reinforcement learning in a cohesive manner. Its compact codebase and reliance on open tools (MuJoCo, JAX, Brax) make it a viable platform for course projects, workshops, or reproducible research.

In summary, this work contributes a high-throughput, imitation-bootstrapped reinforcement learning pipeline for legged locomotion — with practical utility, research extensibility, and educational clarity at its core.

5 Ethical Issues

Although this project is limited to simulation-based bipedal locomotion research, several ethical considerations arise from both the methods used and the potential applications of learned controllers in real-world settings.

Safety in Physical Deployment. Simulated training enables rapid iteration and risk-free testing, but transferring learned policies to physical robots poses nontrivial safety challenges. Controllers—especially those trained with reinforcement learning (RL)—can exhibit unstable or unpredictable behavior when faced with real-world dynamics, sensor noise, or hardware limitations. Failure to maintain balance or misinterpretation of sensor feedback may lead to physical damage or harm. As such, any future deployment of these policies must include hardware constraints, safety monitors, and rigorous validation protocols under supervised conditions.

Bias and Generalisation Limitations. The behavioural cloning (BC) stage is trained exclusively on demonstrations from a deterministic finite-state machine (FSM). While this eliminates stochasticity and simplifies the regression task, it may encode biases specific to that controller's gait pattern. The lack of diversity in training data could restrict the policy's ability to generalise across different terrains or perturbations, potentially reducing the robustness of learned behaviours in open-world settings.

Energy Efficiency and Computational Resources. The experimental sweep conducted in this project involved training over 100 PPO configurations using high-throughput JAX pipelines on GPU hardware. While this allows efficient large-scale evaluation, it also contributes to computational energy consumption. Careful design of experiments—including limiting redundant configurations and using statistical inference to guide sweeps—helps mitigate the environmental impact. Awareness of the compute budget is an important part of responsible research practice, particularly in reinforcement learning workflows that are known to be resource-intensive.

Dual-Use Considerations. Research in legged locomotion has numerous positive applications—including assistive robotics, rehabilitation, and search-and-rescue—but also carries dual-use potential. The same algorithms can be adapted for autonomous military platforms or surveillance systems. While this work does not target such domains, developers should remain mindful of how their methods and open-source releases may be repurposed beyond their original intent. Thoughtful licensing, access control, and communication of project scope can help manage such risks.

Transparency and Reproducibility. To support reproducible research, all training code, simulation files, and model checkpoints are version-controlled and based on standard frameworks such as JAX, MuJoCo, and Brax. The project avoids proprietary toolchains and includes logging of training metrics and configuration metadata for all experiments. Transparency in reporting hyperparameters, architecture choices, and reward formulations ensures that other researchers can verify results and understand the design decisions behind them.

In sum, although this project does not involve human participants or real-world deployment, it touches on themes of safety, generalisability, resource use, and dual-use risks. Addressing these proactively—through good engineering practice and open research ethics—lays the foundation for responsible scaling of such systems to more complex and applied contexts.

6 Project Management

This project was conducted as a solo effort, encompassing all aspects from simulation design to reinforcement learning pipelines and experiment infrastructure. While not originally planned as a strict progression, the development naturally followed a structured path: expert controller $\rightarrow$ behavioural cloning $\rightarrow$ PPO fine-tuning $\rightarrow$ high-throughput Brax sweeps. Each stage informed the next, allowing incremental debugging and gradual expansion of the system's complexity.

Timeline and Milestones. The project evolved through four loosely defined phases:

- **Phase I — FSM Expert Design:** A rule-based walking controller was crafted using MuJoCo, yielding a reliable cyclic gait and generating the initial dataset.
- **Phase II — Behaviour Cloning:** The FSM demonstrations were distilled into lightweight neural networks trained with JAX, providing a differentiable policy for RL bootstrapping.
- **Phase III — PPO Fine-Tuning:** PPO was applied both from BC initialisation and from scratch to evaluate sample efficiency and gait robustness.
- **Phase IV — Brax Vectorisation & Sweeps:** The model was ported to the Brax framework for large-scale GPU training, culminating in a 120-job hyperparameter sweep.

Technical Learning Curve. As a non-CS student, one of the primary challenges was the need to self-learn many of the foundational tools—particularly JAX, MuJoCo, and Brax—from scratch. These frameworks, while powerful, are under-documented and rapidly evolving, which introduced friction during debugging and integration. The most technically demanding stage was the Brax conversion process, which required both XML-level modifications (e.g., injecting dummy joints for compatibility) and semantic alignment with the original MuJoCo dynamics.

Resources and Compute. Most of the project development—including simulation, BC training, and early PPO—was carried out on a personal laptop without GPU acceleration. Only in the final phase was a workstation equipped with an NVIDIA RTX 4060 Ti made available. This enabled accelerated PPO training and allowed for the full sweep experiments to be executed efficiently.

Deliverables. The final deliverables include not only this report but also a carefully structured and well-documented code repository. The repository is designed to be informative and reusable, with an emphasis on reproducibility and clarity. It includes all environment definitions, training scripts, sweep infrastructure, and visualisation tools. Given the lack of resources available for Brax-based bipedal locomotion research, the repository may also serve as a useful reference point for future projects in the field.

7 Conclusion & Future Work

This project developed a complete learning pipeline for passive bipedal walking, progressing from expert-driven control to neural policies fine-tuned through reinforcement learning. Using MuJoCo as the initial simulator, a finite-state controller was implemented to encode domain knowledge into a hand-designed yet reliable walking gait. From this, behaviour cloning (BC) enabled supervised transfer of expert demonstrations into a compact, differentiable policy that served as a stable initialisation for Proximal Policy Optimisation (PPO).

Subsequently, the model was ported to the Brax simulator to enable scalable training and high-throughput experimentation. This required non-trivial modifications to the original XML model for compatibility, including the introduction of a dummy joint to satisfy Brax's tree structure constraints. Once established, Brax's vectorised JAX backend made it possible to execute an extensive hyperparameter sweep across architecture, learning rate, and reward scaling configurations. The final result is a reproducible and modular control stack capable of producing stable locomotion and serving as a research testbed.

Key Contributions.

- **Integrated learning pipeline:** A modular and staged workflow that transitions from finite-state control to fully learned policies using BC and PPO.
- **Custom Brax environment:** A carefully engineered Brax environment with PD control and physics-consistent dynamics derived from a MuJoCo model.
- **Vectorised PPO infrastructure:** A JAX-accelerated PPO training framework with full support for behaviour-cloned warm-start and reward shaping.
- **Large-scale hyperparameter sweep:** A 120-job grid search with support for systematic exploration of model capacity, learning rate, and reward scaling.
- **Reproducible open-source codebase:** All training logic, sweep orchestration, and visualisation tools are published in a structured and extensible format.

Limitations. Despite the strengths of this work, several limitations should be acknowledged:

- **Simplified dynamics:** The passive walker operates on a fixed slope and flat ground, with no contact uncertainty or unmodeled dynamics.
- **Narrow reward definition:** The reward function optimises purely for forward progress (Δx), omitting terms for energy usage, stability, or smoothness.
- **Hand-designed observation space:** Input features were selected and normalised manually, limiting potential for raw sensor learning or end-to-end perception.
- **Limited controller expressivity:** The policy outputs position targets for fixed-gain PD actuators, which may cap achievable agility or adaptability.
- **No real-world validation:** All training and evaluation occurred in simulation, and no domain randomisation or sim-to-real transfer methods were explored.

Future Work. Building on this foundation, several natural extensions are possible:

- **Rich terrain environments:** Introduce uneven surfaces, friction variation, or obstacles to assess generalisation under more challenging dynamics.
- **Expanded reward functions:** Design multi-objective rewards that capture smoothness, symmetry, energy minimisation, or foot clearance in addition to speed.
- **Curriculum learning:** Automate the transition from BC to PPO via adaptive mixing or self-paced imitation schedules to stabilise early RL stages.

- **End-to-end sensor learning:** Incorporate raw sensory signals (e.g., proprioceptive data or depth images) to remove reliance on hand-shaped observations.
- **Alternative RL algorithms:** Explore alternative on-policy (e.g., TRPO) or off-policy (e.g., SAC, DDPG) methods for better sample efficiency or robustness.
- **Multi-agent extensions:** Extend the environment to include coordination or competition among multiple agents with distinct gaits or objectives.
- **Sim-to-real transfer:** Deploy the trained policy on a physical platform, incorporating domain randomisation or adaptation strategies to bridge the reality gap.

In conclusion, this project demonstrates a full-stack approach to legged locomotion research, progressing from interpretable control logic to scalable reinforcement learning. It bridges classic robotics techniques with modern learning frameworks and offers a reproducible and extensible base for further investigation into adaptive, efficient, and transferable locomotion policies.

A Appendix

Code Repository. All source code, simulation models, and experiment scripts are available at:

`https://github.com/yunusdanabas/passive_walker_rl.git`

The repository includes:

- The full control pipeline: **FSM → Behaviour Cloning → PPO → Brax**,
- Environment definitions for both MuJoCo and Brax backends,
- JAX-compatible policy and critic networks using Equinox,
- Sweep utilities for large-scale PPO experiments in Brax,
- Visualisation tools for evaluating learned policies in MuJoCo.

Network Architectures. The actor and critic are implemented as lightweight multi-layer perceptrons. The BC controller, PPO models, and Brax sweep architectures share consistent structural patterns, varying in depth and width depending on the stage and configuration.

Simulator Details.

- **MuJoCo:** Used for prototyping, visualization, and behaviour cloning demonstrations.
- **Brax:** Converted from MuJoCo via MJCF parser; used for high-throughput PPO sweeps on GPU.

Reproducibility. To ensure reproducibility, all results are version-controlled, seeded, and logged. The Brax sweep can be rerun with a single command:

```
python -m passive_walker.brax.sweep_ppo
```

Outputs include serialized models, performance logs, and plots of reward metrics across configurations.

References

[1] N. Heess, S. Sriram, T. Lillicrap, J. J. Hunt, D. Silver, and M. Riedmiller, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017. [Online]. Available: https://arxiv.org/abs/1707.02286

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," in *arXiv preprint arXiv:1707.06347*, 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[3] X. B. Peng, A. Kanazawa, S. Toyer, P. Abbeel, and S. Levine, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 143:1–143:14, 2018.

[4] P. Brakel, M. Gienger, K. Yamane, and M. Kalakrishnan, "Learning whole-body control for humanoid locomotion with centroidal dynamics imitation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 6034–6041.

[5] K. Koseki, M. Okada, T. Fukuhara, H. Okuno, and A. Takanishi, "Multimodal bipedal locomotion generation with passive dynamics via deep reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 2229–2235.

[6] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 5026–5033.

[7] D. M. Freeman, S. Colmenarejo, S. Wang, H. Nowak, C. Gulcehre, N. Heess, and O. Bachem, "Brax 2: Faster and more flexible physics simulation using mjx," *arXiv preprint arXiv:2401.01309*, 2024. [Online]. Available: https://arxiv.org/abs/2401.01309