Passive Walker RL: From FSM to JAX/Brax

Expert Controllers & Scalable RL

Yunus Emre Danabaş

Mechatronic Engineering
Sabanci University

July 17, 2025

Sabancı Üniversitesi

Outline / Agenda

- Introduction & Motivation
- Methodology
- Results & Discussion
- Impact & Ethical Issues
- Project Management & Timeline
- Conclusion & Future Work
- Appendix

Introduction & Motivation

- Legged locomotion remains a challenging control problem:
  - High-dimensional dynamics, contact events, underactuation
  - Sample-inefficient exploration in standard RL
- Expert demonstrations can bootstrap learning and ensure safety
- JAX/Brax offers GPU-accelerated, vectorised simulation for fast RL
- Goal: combine rule-based experts with modern RL for efficient, robust passive walking

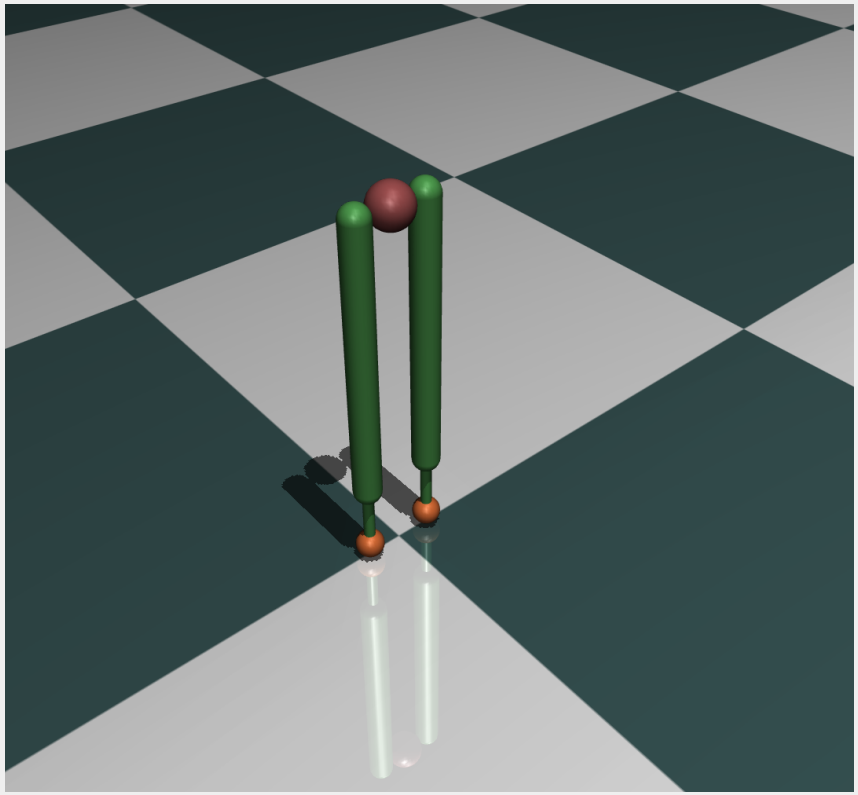
Problem Statement

- Design a control pipeline for a planar passive walker:
  - Leverage a finite-state expert for stable baseline gait
  - Learn a differentiable policy via behaviour cloning (BC)
  - Fine-tune with Proximal Policy Optimization (PPO)
- Achieve both sample efficiency and high final performance
- Scale up training using a vectorised JAX/Brax implementation
- Systematically explore hyperparameters to identify robust configurations

Key Contributions

1. **Expert–to–RL Pipeline:** FSM → Behaviour Cloning → BC-seeded PPO for “walk-from-day-one” learning
2. **Imitation-Regularised PPO:** Clipped surrogate loss augmented with decaying BC term for stability
3. **Vectorised JAX/Brax Pipeline:** High-throughput, GPU-accelerated simulation enabling large-scale experiments
4. **Large-Scale Hyperparameter Sweep:** 120-job grid over reward scaling, learning rate, network capacity
5. **Open-Source Release:** github.com/yunusdanabas/passive_walker_rl

Physics Model & Simulation Setup



- **MuJoCo model:** 5 bodies, 7 DoFs (planar slide x, z, yaw hinge, hip hinge, two prismatic knees)
- **Virtual ramp:** gravity tilted 11.5° downhill
- **Simulation:** physics timestep = 1 ms; control at 200–1 000 Hz

Joints & Actuators

Joint	Type
slide_x	prismatic
slide_z	prismatic
yaw	hinge
hip	hinge
left_knee	prismatic
right_knee	prismatic

Ranges: slide x, z, yaw, hip unbounded; knees ± 0.30 m.

Actuator	k_p
hip_act	5
left_knee_act	1000
right_knee_act	1000

Control range: hip ± 0.5 rad; knees ± 0.3 m.

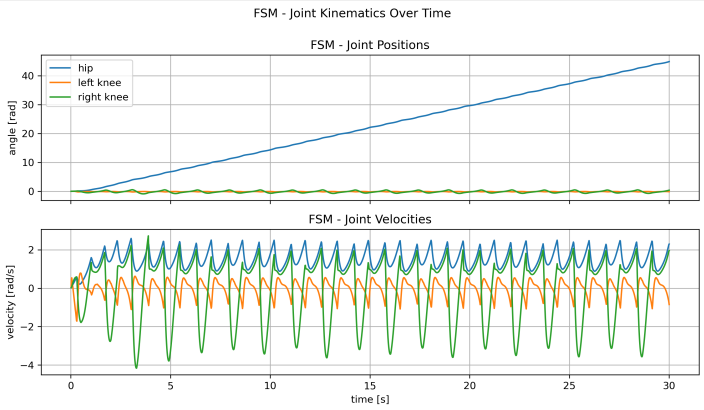
Expert Finite-State Controller

FSM Logic (Detailed Conditions)

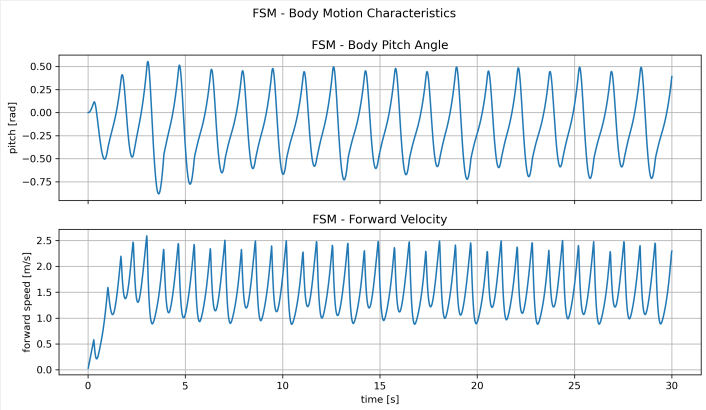
- **Hip Controller:**
  - *Phase:* one leg swings, the other supports.
  - *Switch when:*
    - Swing foot contacts ground ($z_{\text{foot}} < 0.05\,\mathrm{m}$),
    - And trunk returns upright (pitch < 0).
  - *Action:* desired hip angle toggles between $\pm 0.3\,\mathrm{rad}$.

- **Knee Controller:**
  - *States:* **Stance** (extended) vs. **Retraction** (bent).
  - *To Retract:*
    - Opposite foot has just landed ($z_{\text{other foot}} < 0.05\,\mathrm{m}$),
    - And local thigh upright (pitch < 0).
  - *To Extend:*
    - Swing leg has moved forward past neutral (hip pitch $>$ threshold).
  - *Positions:* retracted = $+0.30\,\mathrm{m}$, stance = $0\,\mathrm{m}$.

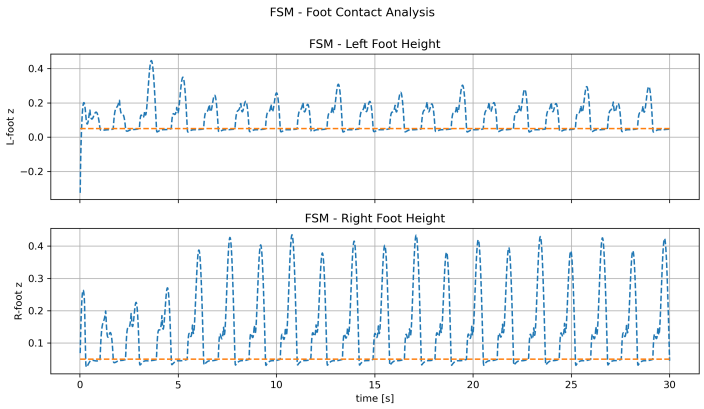
FSM Kinematics Summary



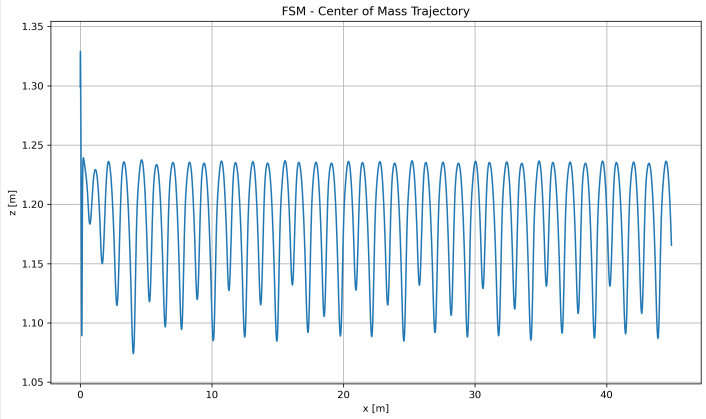
(a) Joint Angles & Velocities



(b) Torso Pitch & Speed



(c) Foot–Ground Contact



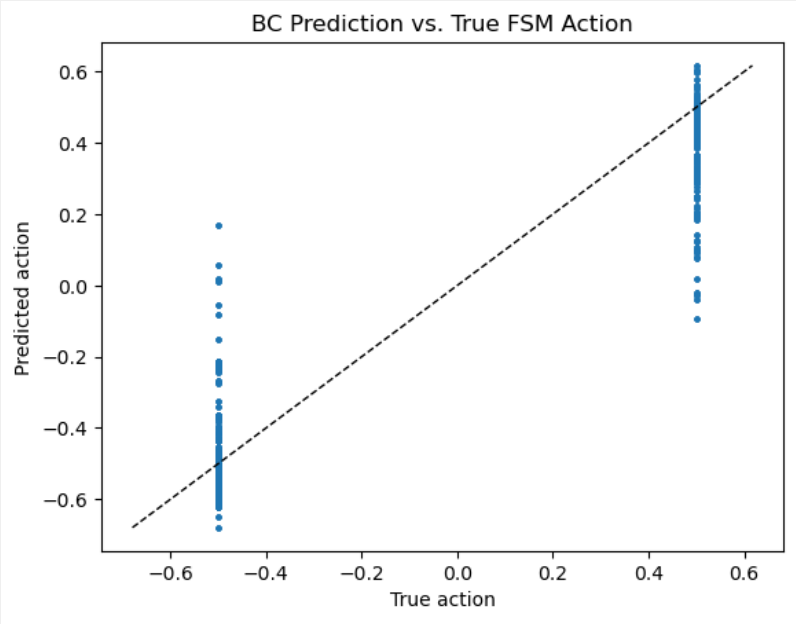
(d) Center of Mass Path

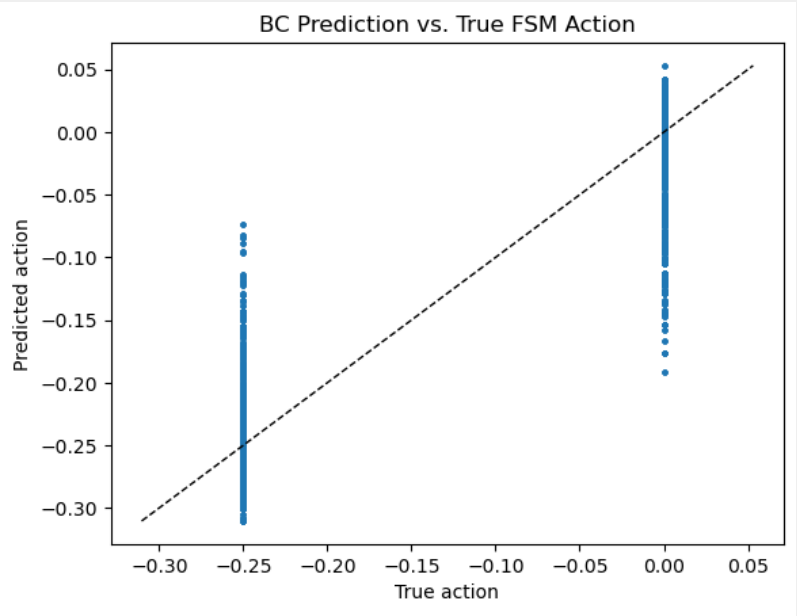
Figure: Expert FSM reference trajectories: (top) joint kinematics and (bottom) foot contact timing and CoM trajectory.

Behaviour Cloning Pipeline

- **Variants explored:**
  - Hip–only BC
  - Knees–only BC
  - Full BC (hip + both knees) with four loss functions: MSE, Huber, L1, Combined
- **Data:** $\sim 10^5$ expert steps at $\sim 10^3$ Hz from FSM $\rightarrow$ observations (11 dims) $\rightarrow$ expert actions (3 dims)
- **Model:** 2-layer ReLU MLP (256 hidden units)
- **Training:** $\mathcal{O}(10^5)$ samples, $\sim 10^2$ epochs, batch ~ 64, $\alpha \sim 10^{-4}$

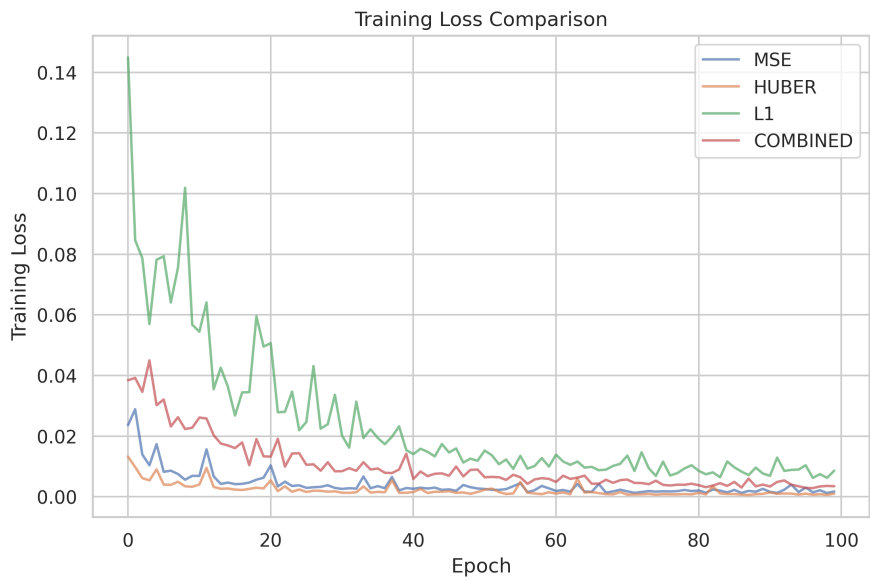
BC Prediction vs. True FSM Actions

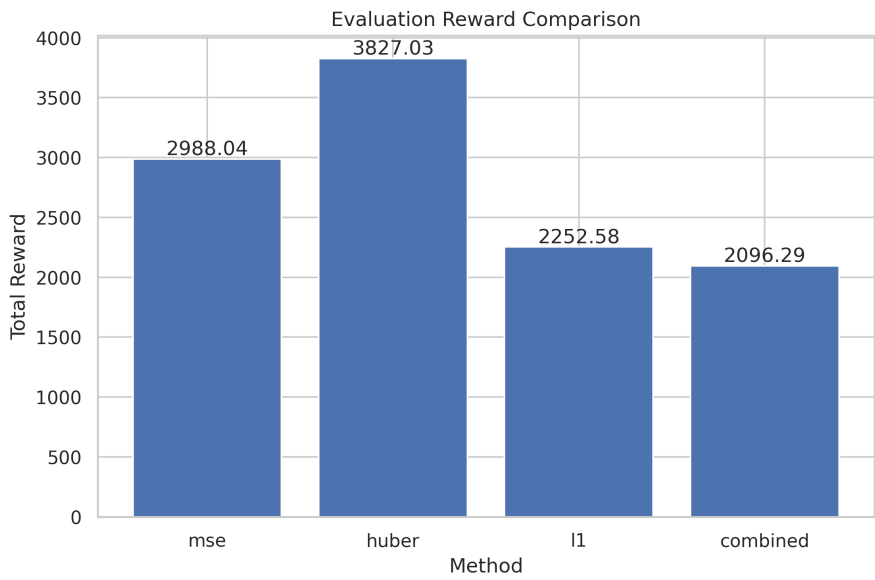


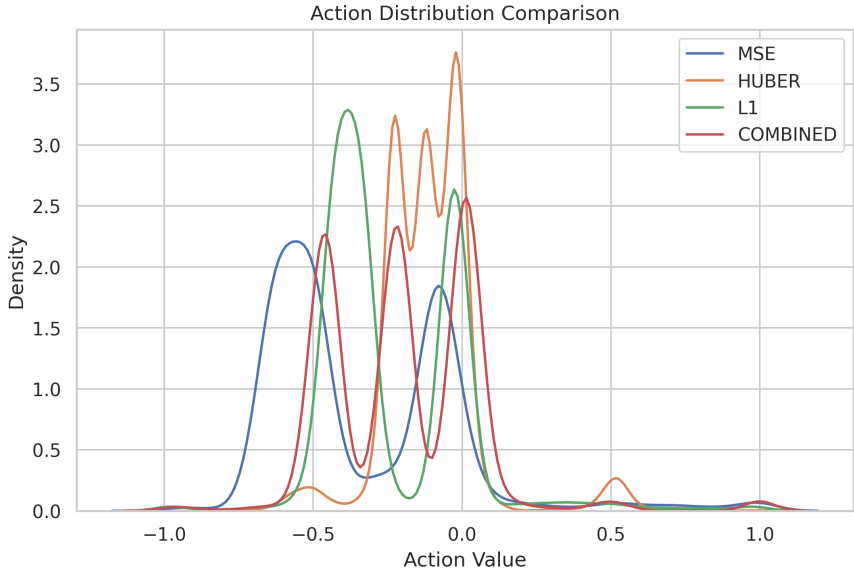


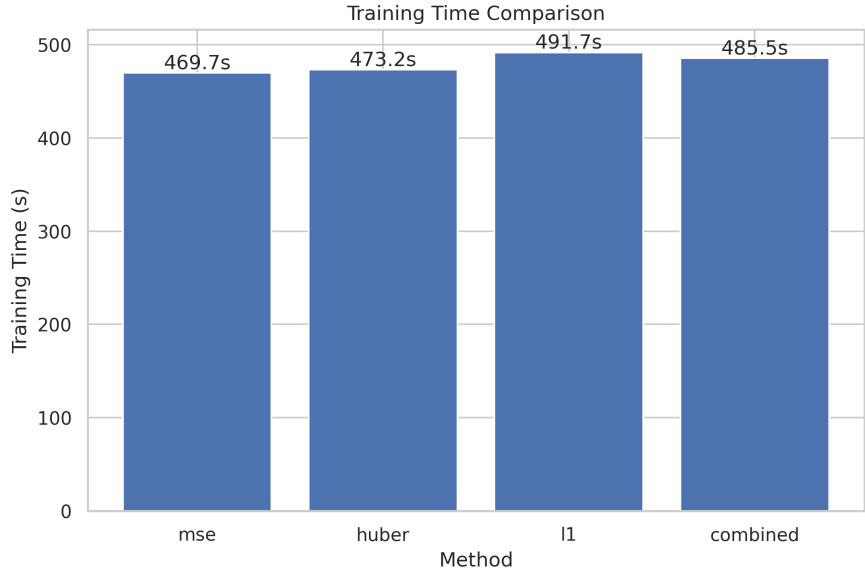
- Scatter of BC predictions versus FSM labels shows tight clustering around $y = x$ for both joints.
- Indicates low bias and accurate reproduction of expert commands across the gait cycle.
- Used 100,000 samples and three loss variants (MSE, Huber, L1, combined).

BC Loss Variants: Visual Comparison







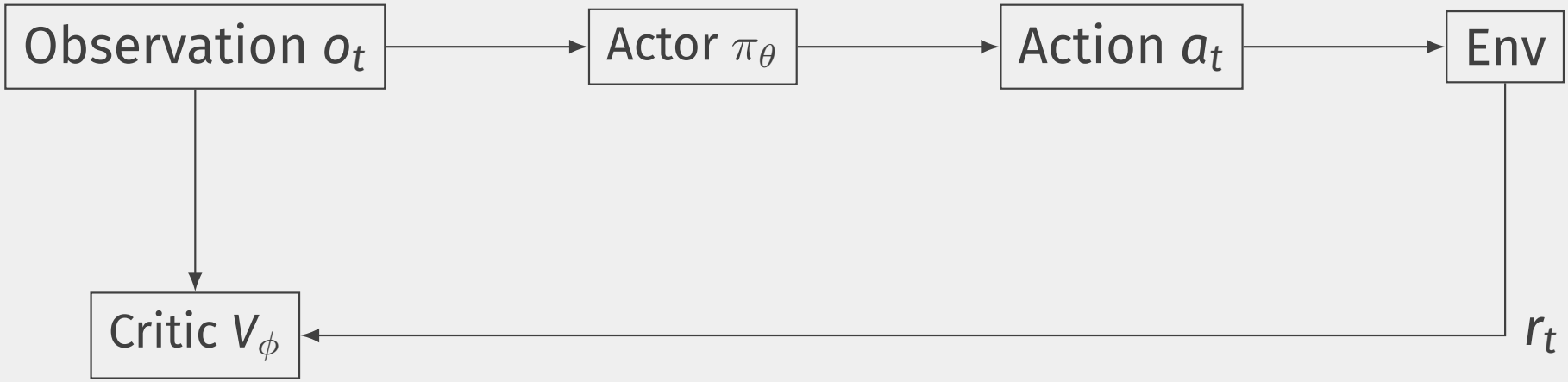


BC Loss Variants: Key Insights

- **Training Loss Convergence:**
  - All four loss functions (MSE, Huber, L1, Combined) converge over 100 epochs.
  - Huber and L1 exhibit slightly smoother decay and robustness to outliers.
- **Evaluation Reward:**
  - Huber-trained policy achieves the highest total reward when deployed.
  - Combined loss performs comparably but with greater variance.
- **Action Distribution:**
  - MSE yields tightly concentrated actions around the expert mean.
  - Combined loss produces a broader distribution, indicating exploratory behavior.
- **Training Efficiency:**
  - MSE and Huber are fastest per epoch.
  - L1 and Combined incur minor extra cost due to additional absolute/huber computations.

PPO Fine-Tuning Strategies

PPO Fine-Tuning Overview



- **Actor** (π_θ): small MLP, one hidden layer
- **Critic** (V_ϕ): two-layer MLP
- **Rollouts**: on-policy data + GAE
- **Updates**: PPO clip + $\beta(t)$-regularisation

PPO Update Algorithm

Require: $\theta,\ \phi,\ \beta_0$

for iteration $= 1 \ldots N$ **do**
  $\mathcal{B} \leftarrow$ CollectTrajectories(π_θ)
  $\{\hat{A}, \hat{R}\} \leftarrow$ GAE$(\mathcal{B}.\text{rewards},\ \mathcal{B}.\text{dones},\ V_\phi(\mathcal{B}.\text{obs}))$
  for epoch $= 1 \ldots K$ **do**
    for all mini-batch $b \subset \mathcal{B}$ **do**
      $L_{\text{clip}} \leftarrow$ PPOClip$(b, \pi_\theta^{\text{old}}, \hat{A})$
      $L_{\text{bc}} \leftarrow \|\pi_\theta(b.\text{obs}) - b.\text{bc\_targets}\|^2$
      $L \leftarrow L_{\text{clip}} + \beta(t)\, L_{\text{bc}}$
      $\theta \leftarrow \theta - \alpha \nabla_\theta L$
    end for
  end for
  $\phi \leftarrow \phi - \alpha_v \nabla_\phi \|V_\phi(\mathcal{B}.\text{obs}) - \hat{R}\|^2$
  $\beta(t+1) \leftarrow \text{Decay}(\beta(t))$
end for

PPO + BC Loss Functions

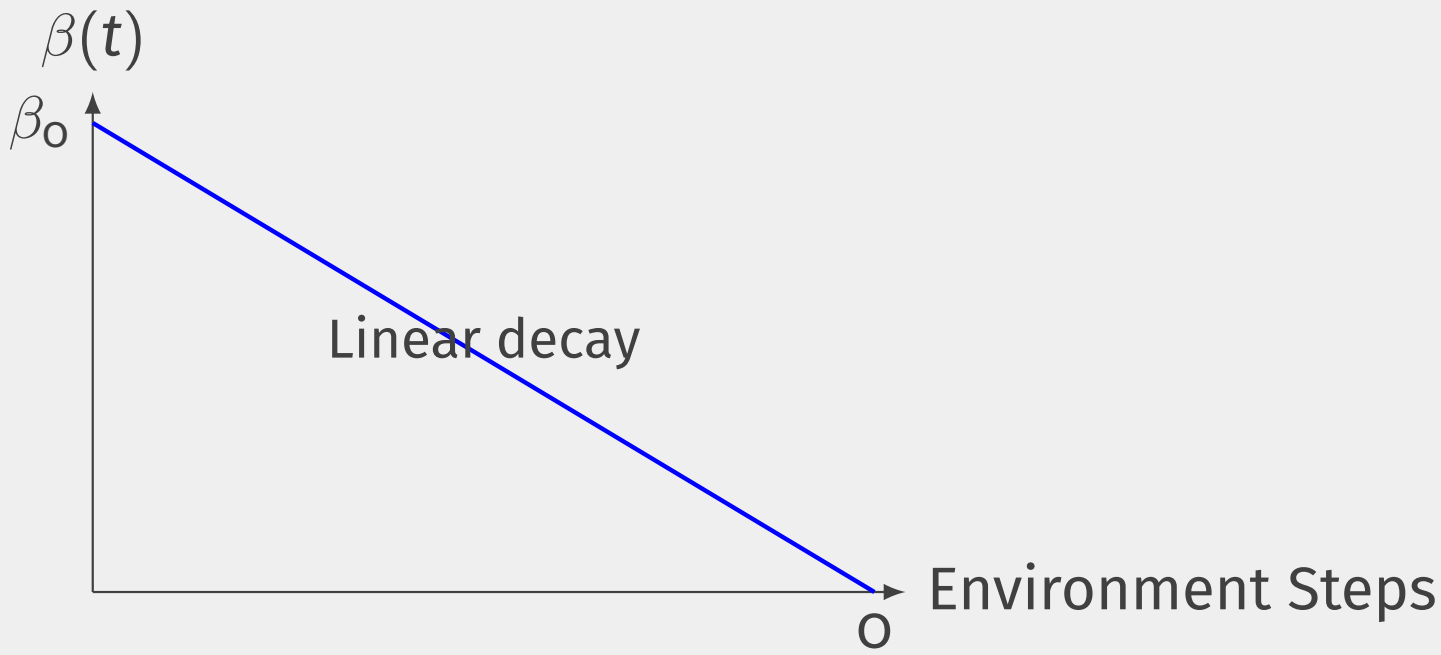
Clipped Surrogate Objective

$$L_{\mathrm{PPO}} = -\mathbb{E}_t\Big[\min\big(r_t\,\hat{A}_t,\ \mathrm{clip}(r_t, 1-\epsilon, 1+\epsilon)\,\hat{A}_t\big)\Big], \quad r_t = \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|o_t)}$$

Total Loss with BC Regularisation

$$L_{\mathrm{total}} = L_{\mathrm{PPO}} \ + \ \beta(t)\,\mathbb{E}_t\big[\|\pi_\theta(o_t) - a_t^{\mathrm{BC}}\|^2\big], \quad \beta(0) = \beta_0,\ \beta(N) \to 0.$$

Imitation Weight Annealing



- $\beta(0) = \beta_0 > 0$: strong imitation early
- $\beta(t) \downarrow 0$: pure PPO later
- Balances stability vs. exploration

BC-seeded vs Scratch PPO

BC-seeded PPO

- Policy initialised from BC weights
- Imitation term $\beta(t) > 0$ early
- Faster convergence to walking gait
- Smaller batch sizes, fewer env steps

Scratch PPO

- Random policy initialization
- No imitation regularisation ($\beta(t) \equiv 0$)
- Requires more exploration
- Longer training to reach stability

Training Flow Recap

1. **Rollouts:** collect on-policy trajectories under π_θ
2. **Advantage Estimation:** compute $\hat{A}, \hat{R}$ via GAE
3. **Policy Update:** clipped PPO + $\beta(t)\,\mathrm{BC}$ loss
4. **Critic Update:** regress V_ϕ to returns $\hat{R}$
5. **Annealing:** linearly decay imitation weight $\beta(t)$

Repeat for N iterations

Vectorised JAX/Brax Pipeline

Pipeline Overview

- **MuJoCo → Brax:** Parse your passive walker XML via `brax.io.mjcf.load_model` into a JAX-native `System`.
- **Environment Wrapper:** `BraxPassiveWalker` inherits `PipelineEnv`, implements `reset` / `step` in pure JAX.
- **PD Control & Reward:** $\tau = K_p(q_{\text{targ}} - q) - K_d\,\dot{q}$, reward $= \Delta x$, terminates on low torso height or large pitch.
- **Vectorisation & JIT:** Compile once with `jax.jit`, run `vmap` over N envs in parallel.
- **Train with PPO:** Call `brax.training.agents.ppo.train` on batched data for maximum throughput.

BraxPassiveWalker: reset

```python
def reset(self, rng):
    # 1) sample initial noise on joints
    rng, sub = jax.random.split(rng)
    noise = (2*jax.random.uniform(sub,(3,))-1)*self.reset_noise
    q0    = self.sys.init_q.at[self.act_idx].add(noise*self.action_scale)
    qd0   = jnp.zeros_like(self.sys.init_qd)

    # 2) initialize pipeline state
    ps = self.pipeline_init(q0, qd0)

    # 3) return Brax State
    return State(
      pipeline_state=ps,
      obs=self._get_obs(ps),
      reward=0.0,
      done=0.0,
      metrics={"prev_x": ps.x.pos[0,0]}
    )
```

- Randomises hip and knees within ±reset_noise
- Builds initial Brax pipeline state → position + velocity
- Packs into `State` with obs, *reward*, *done*, *metrics*

BraxPassiveWalker: step

```python
def step(self, state, action):
    # 1) PD controller
    act_scaled = jnp.clip(action,-1,1)*self.action_scale
    q, qd      = state.pipeline_state.q[self.act_idx], state.pipeline_state.qd[self.
         act_idx]
    tau        = self.kp*(act_scaled - q) - self.kd*qd

    # 2) forward simulation
    ps_next    = self.pipeline_step(state.pipeline_state, tau)

    # 3) reward & done
    dx    = ps_next.x.pos[0,0] - state.metrics["prev_x"]
    height_ok = ps_next.x.pos[0,2] > 0.5
    pitch_ok  = abs(quat_to_euler(ps_next.x.rot[0])[1]) < 0.8
    done_flag = jnp.logical_not(height_ok & pitch_ok)

    # 4) pack next state
    return state.replace(
      pipeline_state=ps_next,
      obs=self._get_obs(ps_next),
      reward=dx,
      done=done_flag.astype(jnp.float32),
      metrics={"prev_x": ps_next.x.pos[0,0]}
    )
```

- **PD torques:** $\tau = K_p(a_{\text{target}} - q) - K_d\,\dot{q}$
- **Reward:** forward progress Δx
- **Termination:** torso height < 0.5 m or pitch || 0.8 rad
- **State replace:** updates obs, reward, done, and prev_x

High-Throughput Training

- **Massive sweep:** 120 PPO jobs × 24M steps each (few hours on RTX 4060 Ti).
- **Parallel simulators:** 128 envs in lock-step via `vmap` → $\times$100 speed-up vs. loop.
- **Compile once:** JAX JIT fixes the compute graph, reuses across all envs iterations.
- **Toolchain:** `Brax` for physics, `Flax/Equinox` for networks, `Optax` for optimizers.
- **Reproducible logging:** `msgpack` payloads, deterministic seeds, helper scripts for aggregate plots.

Experimental Sweep Design

Experimental Sweep Design

- **Goals:**
  - Find the single best hyper-parameter configuration
  - Analyse trends over reward scale, learning rate and network size
- **Grid:** 3 seeds × 2 reward scales × 4 LRs × 5 architectures = 120 jobs
- **Metrics:** final episode reward (mean ± std over seeds), wall-clock time, stable completions

Dimension	**Values (count)**
Seeds	{0,1,2} (3)
Reward scale	{0.5,1.0} (2)
Learning rate	{1e-3,5e-4,1e-4,1e-5} (4)
Architectures	{tiny,small,medium,deep,deepXL} (5)
Total runs	120

Table: Sweep grid dimensions (values in braces).

Sweep Results: Reward Scale & Learning Rate Trends

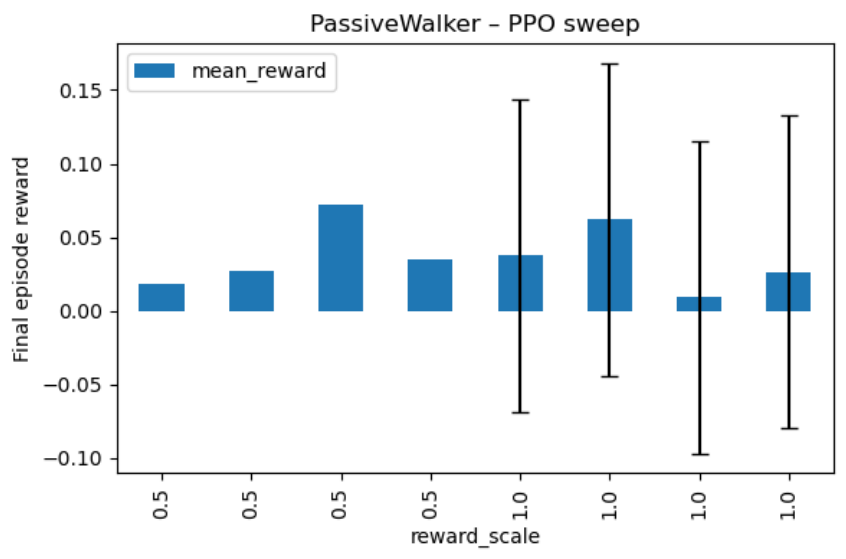


Figure: Mean ± std reward vs. reward scale

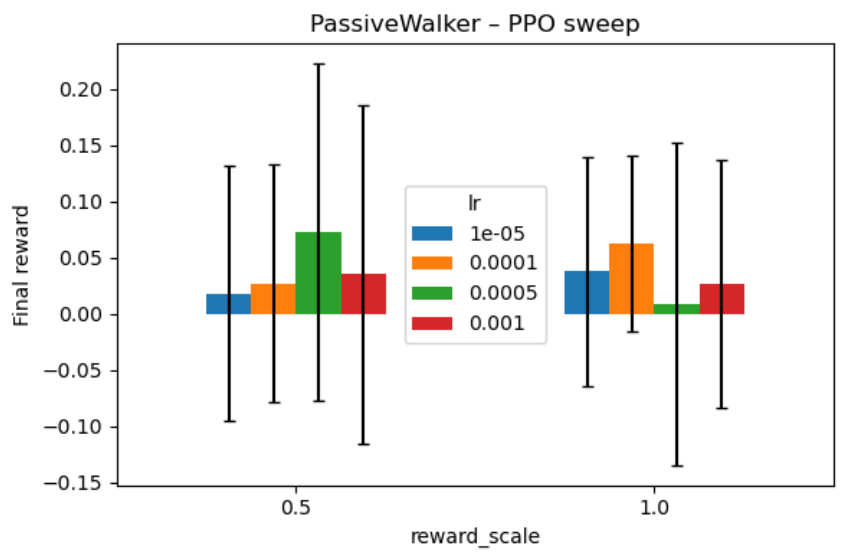


Figure: Mean ± std reward vs. learning rate

Key Insights & Best Configuration

Best Hyper-Parameter Configuration

S2 | R = 0.5 | LR = 1e-3 | Arch = `medium`

- Highest mean reward across seeds
- Shows that moderate reward scaling and a higher learning rate perform best
- Typical wall-clock: ∼2–10 min per job on RTX 4060 Ti
- All 120 jobs completed successfully (no simulator crashes)

Guideline for future experiments: start with `medium`+LR=1e-3+scale=0.5.

Results & Discussion

Behaviour Cloning Performance

- **BC variants:** hip-only & knee-only (context), focus on full BC (hip + knees)
- **Losses compared:** MSE, Huber, L1, Combined

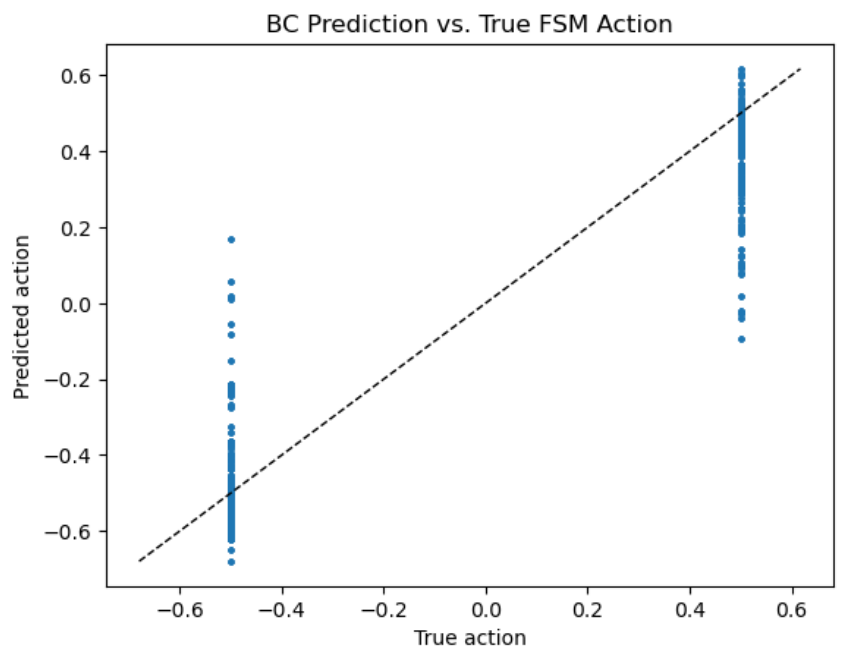


Figure: Hip: BC vs. FSM labels

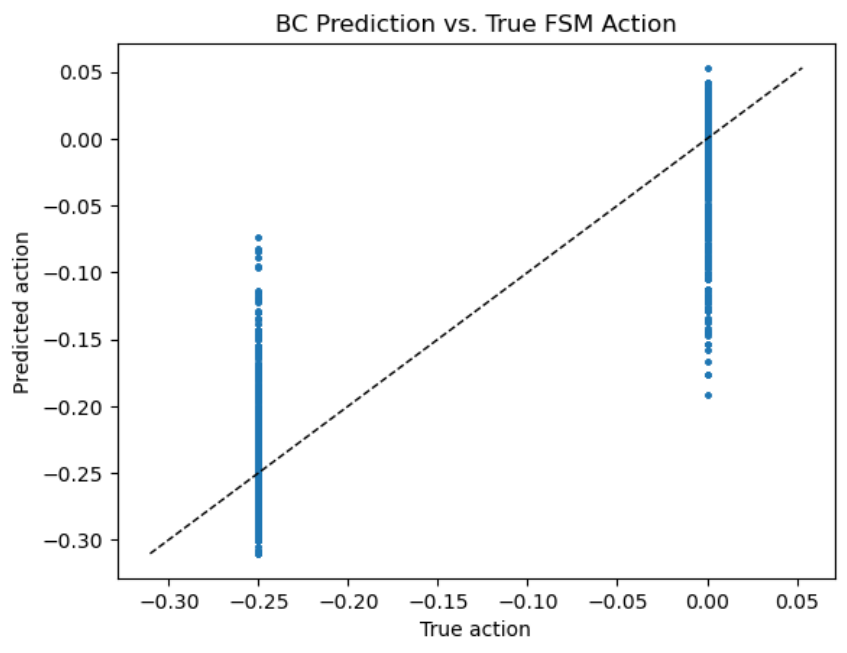


Figure: Knee: BC vs. FSM labels

BC Loss Variant Comparison

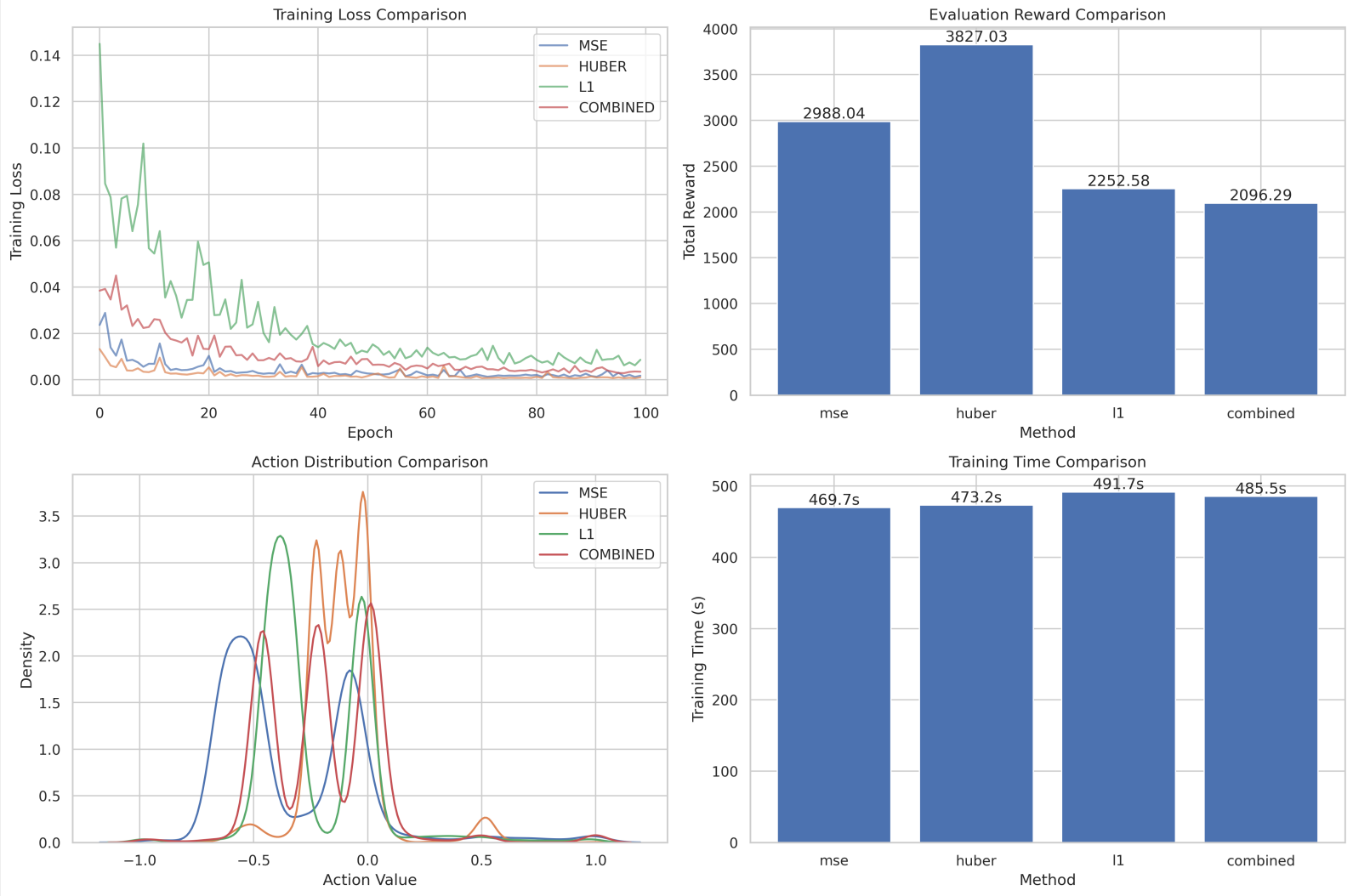


Figure: Training loss, evaluation reward, action-density and runtime for each BC objective

PPO Fine-Tuning Performance

- **BC-seeded PPO** converges in $\sim$50% of the iterations vs. scratch PPO
- **Scratch PPO** achieves similar final gait but needs $\times 2$–$\times 3$ more samples
- Decaying imitation weight $\beta(t)$ stabilises early training

Hyperparameter Sweep: Aggregated Trends

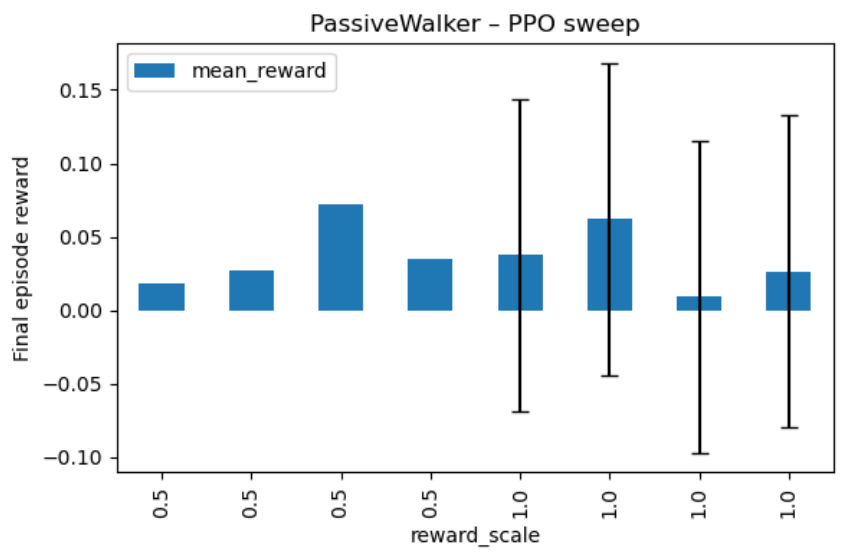


Figure: Mean±std reward vs. reward scale

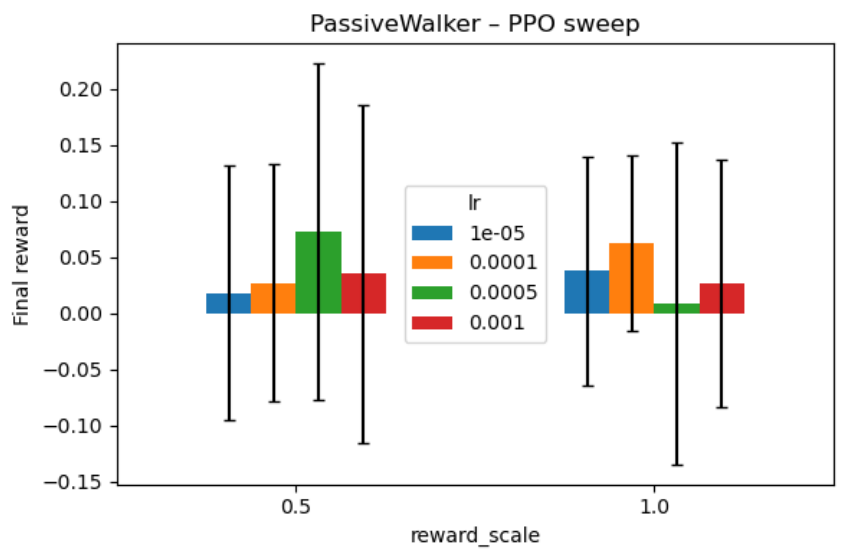


Figure: Mean±std reward vs. learning rate

Discussion & Key Takeaways

- **Behaviour Cloning:** Huber loss offers best trade-off of bias vs. robustness
- **Imitation-seeded PPO:** drastically cuts sample complexity and speeds convergence
- **Sweep guidelines:** medium networks, LR=1e-3, scale=0.5 for strong performance
- **High-throughput pipeline:** vectorised JAX/Brax makes such large sweeps practical

Impact & Ethical Issues

Impact

- **Accelerated Research:** “Walk-from-day-one” imitation jump-starts RL in legged locomotion.
- **Open-Source Toolkit:** JAX/Brax pipeline + scripts for large-scale sweeps released on GitHub.
- **Broader Applications:** Assistive robotics, search-and-rescue, exploratory platforms.
- **Efficiency Gains:** Vectorised simulation achieves $\times 100$ speed-up vs. naive loops.

Ethical Issues

- **Real-World Safety:** Simulator-to-robot gap demands rigorous hardware safety checks.
- **Bias in Demonstrations:** FSM expert encodes narrow gait patterns—limits generalisation.
- **Compute Footprint:** 120-job sweep deep training consume GPU hours—track budgets.
- **Dual-Use Risks:** Legged controllers may be repurposed for surveillance or military.
- **Transparency & Reproducibility:** Full code, configs, logs published; encourage peer verification.

Project Management & Timeline

Project Management

- **Solo Effort:** All design, implementation, experiments by the author
- **Learning Curve:** JAX/MuJoCo/Brax resources sparse—many tools mastered from scratch
- **Challenges:** XML→Brax compatibility, PD tuning, large-scale sweep orchestration
- **Resources:** Initial work on CPU laptop; final deep runs on RTX 4060 Ti workstation
- **Deliverables:** Well-organized GitHub repo with code, data, scripts, and documentation

Timeline

Phase	Activities
FSM & Data Collection	Expert FSM design, MuJoCo demos, dataset logging
Behaviour Cloning	MLP architecture, loss variants, BC training
PPO Fine-Tuning	BC-seeded & scratch PPO, ablation studies
Brax Port & Hyperparameter Sweep	MJCF→Brax conversion, 120-job PPO grid
Write-up & Release	Report chapters, presentation, open-source push

Conclusion & Future Work

Conclusion

- **Pipeline Success:** FSM→BC→PPO→Brax achieved robust passive walking “from day one.”
- **Sample Efficiency:** Imitation regularisation halved RL training steps vs. scratch.
- **Scalability:** JAX/Brax vectorisation made 24 M-step sweeps feasible in hours.
- **Open Science:** All code, models, results publicly available for community reuse.

Future Work

- **Complex Terrain:** Extend to uneven ground, stairs, and variable slopes.
- **Hardware Validation:** Transfer policies to real robot—study sim-to-real gaps and safety.
- **Domain Randomisation:** Improve robustness to mass, friction, and sensor noise variations.
- **Advanced Architectures:** Graph-based or attention-powered controllers for multi-limb coordination.
- **Energy Efficiency:** Incorporate power/regret into reward for practical deployment.

Additional Resources

- **Code & Models:** `https://github.com/yunusdanabas/passive_walker_rl.git`
- **Contact:** yunusdanabas@sabanciuniv.edu