Evolving Control: Evolved High Frequency Control for Continuous Control Tasks

Samuel Holt[*]
sih31@cam.ac.uk
University of Cambridge

Todor Davchev
Google DeepMind

Dhruva Tirumala
Google DeepMind

Ben Moran
Google DeepMind

Atil Iscen
Google DeepMind

Antoine Laurens
Google DeepMind

Yixin Lin
Google DeepMind

Erik Frey
Google DeepMind

Markus Wulfmeier[†]
Google DeepMind

Francesco Romano[†]
Google DeepMind

Nicolas Heess
Google DeepMind

Abstract: High-frequency control in continuous action and state spaces is essential for practical applications in the physical world. Directly learning end-to-end high-frequency control struggles with assigning credit to actions across long temporal horizons, compounded by the difficulty of efficient exploration. The alternative, learning low-frequency policies that guide higher-frequency controllers (e.g., proportional-derivative (PD) controllers), can result in a limited total expressiveness of the combined control system, hindering overall performance. We introduce *EvoControl*, a novel bi-level policy learning framework for learning both a slow high-level policy (using PPO) and a fast low-level policy (using Neuroevolution) for solving continuous control tasks. Learning with Neuroevolution for the lower-policy allows robust learning for long horizons that crucially arise when operating at higher frequencies. This enables *EvoControl* to learn to control interactions at high frequency, benefitting from more efficient exploration and credit assignment than direct high-frequency torque control without the need to hand-tune PD parameters. We empirically demonstrate that *EvoControl* can achieve a higher evaluation reward for continuous-control tasks compared to existing approaches, specifically excelling in tasks where high-frequency control is needed, such as those requiring safety-critical fast reactions.

Keywords: Safety, High-frequency Control, Control

1 Introduction

High-frequency control is paramount for ensuring the safety and reliability of systems that operate in the real world [1, 2, 3, 4]. Failures to respond in real-time to unexpected collisions, disturbances, or human interaction can lead to catastrophic consequences in safety-critical applications such as surgery [5], autonomous driving [6], and industrial automation [7].

Recent work falls into two main categories. First, directly learning end-to-end high-frequency (often torque-based [8]) policies, labeled *direct torque control*, presents significant learning challenges as this increases the number of state transitions within a fixed time window, resulting in longer trajectories with complex temporal dependencies that hinder exploration and credit assignment [9, 10, 11]—and can often lead to suboptimal behavior (Section 5.2).

Second, a common alternative is to learn a slower high-level policy that outputs target positions or velocities, which are then tracked by a faster (higher-frequency) low-level fixed controller, such as a proportional-derivative (PD) controller [12]. This bi-level approach, labeled as *fixed controllers*,

[*]Work done during an internship at Google DeepMind.
[†]Equal advising.

Workshop on Safe and Robust Robot Learning for Operation in the Real World (SAFE-ROL) at CoRL 2024.

is prevalent in real-world continuous control tasks [8, 13] and simplifies learning by reducing the effective time horizon of the high-level policy [14]. For real-time control, this composition affords the high-level policy larger inference time, key for handling rich observations, such as images (i.e., at 30Hz) or using larger networks, such as Visual-Language-Models [15]. While composing with a low-level PD controller that operates at higher-frequencies $\sim 500Hz$ on robotic platforms, often only observing at high-frequency direct robot proprioceptive observations such as robot joint positions, velocities and torques [16]. However, *fixed controllers* are unable to produce fast interaction behavior beyond simple state-goal-tracking limiting their expressiveness, and require manual tuning of their PD parameters for each task.

An effective method for high-frequency control therefore aims to have the following three core properties: **(P1) Efficient Exploration:** Throughout learning be able to efficiently explore the state space as well as a high-level policy with a *fixed controller*. **(P2) High-Frequency Interaction Control:** Enable the learning of a low-level controller capable of complex, adaptive behaviors at high frequencies. **(P3) Automate Controller Tuning:** Reduce manual tuning of the low-level PD controller parameters.

Contributions: ① We introduce *EvoControl*, a novel bi-level policy learning framework for learning both a slow (e.g. 30Hz) high-level policy and a fast (e.g. 500Hz) low-level robot proprioceptive controller using PPO and Neuroevolution, respectively, for continuous-control tasks (Section 3). ② Theoretically, we show that there exist some MDPs where higher frequency actions can be more optimal (Proposition 2.1). Empirically, we demonstrate that EvoControl can achieve a higher evaluation reward for standard continuous-control tasks at high-frequency compared to existing approaches, excelling in tasks where high frequency control is needed, such as in safety critical applications of unmodeled interactions (Section 5.1). ③ We gain insight and understanding of how EvoControl can achieve efficient exploration compared to direct torque control at high-frequency, learn fast interactions, and demonstrate robustness to mistuned PD parameter settings.

2 Problem

We follow the standard continuous control reinforcement learning (RL) setting with the inclusion of an optional low-level controller.

States & Actions. We denote the environments state space as $\mathcal{S} \subset \mathbb{R}^{d_s}$ and its action space as $\mathcal{U} \subset \mathbb{R}^{d_u}$. At time $t \in \mathbb{R}$, the system's state is represented by $\mathbf{s}_t \in \mathcal{S}$, and its action by $\mathbf{u}_t \in \mathcal{U}$. Considering action (e.g. actuator) limits the action space is constrained to a box in Euclidean space: $\mathcal{U} = [\mathbf{u}_{\text{min}}, \mathbf{u}_{\text{max}}]$.

Environment Dynamics. The transition dynamics for continuous control environments can be described by an underlying unknown *differential equation* of $\dot{s}_t = \frac{ds_t}{dt} = f(s_t, u_t)$. The transition function, which describes the evolution of the state over a discrete time step Δ_t, can be approximated using the Euler method $s_{t+\Delta_t} \approx s_t + \Delta_t f(s_t, u_t)$. Given an action u_t and current state s_t, $s_{t+\Delta_t} \sim P(s_{t+\Delta_t}|s_t, u_t)$ is implicitly defined by this approximation. More sophisticated numerical integration methods can also be used. We expand on the problem setup in Appendix A.

Policies. The agent can be represented as a single policy $\pi : \mathbb{R}^{d_s} \to \mathbb{R}^{d_u}$, that observes the current observation at time t and samples an action $u_t \sim \pi(s_t)$ and then applies this action to the environment at a given fixed Δ_t.To formalize a bi-level policy, we decompose π into two components: a slow, high-level policy, ρ, and a fast, low-level policy, β. Both policies interact with the environment as described in Algorithm 1. At time step k, ρ outputs a latent action $a_k \sim \rho(s_k)$ (e.g., a target position or velocity). Operating at a higher frequency, β receives a_k and generates low-level actions u_{k+i} (e.g., motor torques) at a finer-grained time step i to achieve the target specified by ρ. With β operating at frequency $1/\Delta_t$, ρ's latent action is executed by β for G steps, making ρ's effective frequency $1/(G\Delta_t)$.

Algorithm 1 Bi-Level Policy Interaction (Single High-Level Step)

$a_k \sim \rho(s_k)$ ▷ High-level action
for $i = 0$ to $G - 1$ **do**
  $u_{k+i} \sim \beta(s_{k+i}, a_k)$ ▷ Low-level action
  $s_{k+i+1} \sim f(s_{k+i}, u_{k+i}, \Delta_t)$

Objective. The environment produces a reward r_t sampled from an unknown reward function $r(s_t, u_t)$. The overall objective of the agent is to maximize the expected future discounted reward $\mathbb{E}_{s_0:T, a_0:T-1, r_0:T-1}\left[\sum_{i=0}^{T-1} \gamma^i r_i\right]$, where $0 \leq \gamma < 1$ is the discount factor.

2.1 Higher Frequency Actions Can Be More Optimal

In some environments controlling with a higher-frequency action can be more optimal—a point we make formally in Proposition 2.1, with a full proof in Appendix B.

Proposition 2.1. *There exist Markov Decision Processes (MDPs) where the optimal control policy, maximizing the expected cumulative reward over a fixed episode duration T, requires an action frequency approaching infinity.*

This highlights the need for high-frequency control in certain environments. This is analogous to the Pulse Width Modulation (PWM) sampling theorem [17], where variable pulse widths enable perfect signal reconstruction from discrete samples, similar to how high-frequency actions enable optimal control in our MDP setting. See Appendix B.1 for a continuous control safety critical intuitive example.

2.2 Background: Fixed PD Controllers

Table 1: Common Fixed Low-Level PD Controllers

Method	a_k	Control Law
PD Absolute Position	$q^d = a_k$	$\tau(t) = K_p(q^d - q) + K_d(\dot{q}^d - \dot{q})$
PD Delta Position	$\delta q^d = a_k$	$\tau(t) = K_p((q_k + \delta q^d) - q) + K_d(\dot{q}^d - \dot{q})$
PD Velocity	$\dot{q}^d = a_k$	$\tau(t) = K_p(\dot{q}^d - \dot{q}) + K_d(0 - \ddot{q})$
PD Integrated Velocity	$\dot{q}^d = a_k$	$\tau(t) = K_p((q^d + \int \dot{q}^d dt) - q) + K_d(0 - \dot{q})$

In continuous control and robotics, hierarchical structures composed of a learned high-level policy (ρ) and a fixed low-level controller (e.g., a PD controller [18]) are common. This hierarchical decomposition reduces the number of decision steps for the high-level policy by a factor of G within a fixed episode duration T, where G is the number of low-level actions executed per high-level action. The high-level policy outputs a target a_k, often a desired position or velocity[3], which the low-level PD controller then tracks. The controller computes a control signal u_t based on the error between the target a_k and the measured system state s_t, $u_t = K_p(a_k - s_t) + K_d(\dot{a}_k - \dot{s}_t)$ where $e_t = a_k - s_t$ is the tracking error, and $K_p, K_d \in \mathbb{R}^+$ are constant proportional and derivative gains. Common PD controller designs using proprioceptive states (joint positions q_t, velocities $\dot{q}_t$, and torques τ_t) are summarized in Table 1. We provide an expanded background on PD controllers in Appendix C.

3 EvoControl: Evolved low-level controller framework

We now propose *EvoControl*, a novel bi-level policy learning framework for learning both a slow high-level policy and a fast low-level policy for continuous control tasks. The key idea is to stabilize the bi-level on-policy learning of a higher-level policy by initially learning with a fixed-low-level PD controller and then annealing to a gradually neuroevolved high-frequency controller—Figure 1 provides a block diagram.

First, we formulate the bi-level policy optimization problem and its challenges. Then, we discuss our approach and the advantages of using Neuroevolution for lower-level policy optimization. Specifically our framework can be applied starting with different semantically meaningful high-level actions a_k from the high-level policy, such as position/velocity targets, commonly seen in existing PD controllers, as outlined in Table 1.

3.1 Promise and Challenges of Policy Hierarchies

Hierarchical reinforcement learning (HRL) promises to tackle complex tasks by leveraging temporal abstraction with multiple policy levels [19, 14, 20]. Diverse low-level trajectories induced by high-level actions improve exploration [14, 21, 22], as exemplified by fixed low-level controllers (e.g., PD) acting as effective temporally extended actions [23]. However, simultaneously learning both policy levels with latent actions can destabilize learning due to the co-dependency between them: changing low-level policies create non-stationarity for the high-level, while the low-level requires informative high-level actions [24, 25, 26]. Moreover, bi-level optimization presents challenges in

[3]The state s_t can encompass a wide range of proprioceptive information beyond joint positions (q_t). We present the target a_k and tracking error in terms of position/velocity to align with common PD controller formulations.

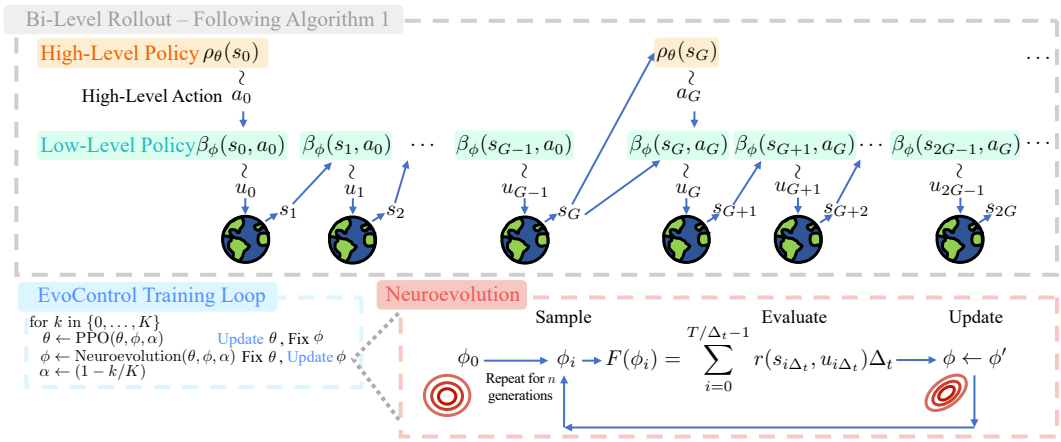

Figure 1: **Top: Bi-level Policy Interaction.** The high-level policy ρ_θ outputs latent action a_k, which guides the low-level policy β_ϕ for G steps. **Bottom: EvoControl Training Loop.** EvoControl trains both the high-level (ρ_θ, with parameters θ) and low-level (β_ϕ, with parameters ϕ) policies over the course of training divided into K discrete sections. Each section k first optimizes θ with PPO (fixed ϕ), then optimizes ϕ with Neuroevolution (fixed θ). Neuroevolution maintains a population of ϕ parameters, evaluates their fitness (episodic return, F), and updates the parameter distribution to maximize average fitness. This process, robust to long horizons, enables learning of complex low-level behaviors. To stabilize learning, β_ϕ is initially a PD controller (β_{PD}, $\alpha = 1$) and transitions to a learned controller as α anneals towards 0.

determining optimal latent actions and low-level rewards. While subgoal-based rewards are intuitive, optimizing the overall episodic return (R) is often desirable but difficult due to the credit assignment problem, especially with high-frequency low-level policies [9]. This motivates our framework for directly optimizing R.

3.2 Efficient High-Level Policy Exploration

In the following we outline how we train both the slow high-level policy ρ and the fast low-level policy β, by training each level in $K \in \mathbb{Z}_+$ stages, with the other fixed, which assists to mitigate the issues of instability and long-horizon credit assignment.

To deliver on the key advantage of efficient high-level state-action exploration for the bi-level control approach, as outlined in Figure 1, we seek a stable way of initially learning the high-level policy, to overcome the non-stationary challenge of learning with a continually updating lower-level policy. A key approach we choose, which, as we will see later, also assists in stably learning a lower-level policy, is to represent the lower-level policy as a convex combination of a fixed PD controller β_{PD} and a learned neural network actor policy $\beta_{\phi_{\text{NN}}}$ with parameters ϕ, and start initially training with only the PD controller, with $\alpha = 1, \alpha \in [0, 1]$ and anneal α to 0 over the course of training. Specifically, we formulate the bi-level policies during training in Algorithm 1 as:

$$\rho_\theta(s_k), \qquad \beta_\phi(s_{k+i}, a_k) = \alpha\beta_{\text{PD}}(s_{k+i}, a_k) + (1 - \alpha)\beta_{\phi_{\text{NN}}}(s_{k+i}, a_k)$$

This bring two immediate advantages, 1) the high-level action a_k output from the high-level policy has an initial semantic meaning, grounding it and allowing a user to select the most appropriate PD controller for the given task, and 2) as α is annealed throughout training from 1 to 0, we inherit the effective state-action exploration properties of having a fixed PD controller initially (Section 5.2), and yet can still retain the flexibility of learning more complex lower-level behavior, beyond just sub-goal/state tracking.

To optimize the high-level policy, we employ Proximal Policy Optimization (PPO) [27], a highly effective on-policy reinforcement learning algorithm for continuous control tasks. The high-level policy is represented by a continuous control agent consisting of a neural network with separate critic and actor heads. The actor head parameterizes a multivariate Gaussian distribution with a diagonal covariance matrix, with parameters θ. Specifically, the high-level policy is defined as:

$$a_k \sim \rho_\theta(s_k) = \mathcal{N}(\mu_\theta(s_k), \Sigma_\theta(s_k))$$

where $\mu_\theta(s_k)$ is the mean vector and $\Sigma_\theta(s_k) = \text{diag}(\sigma^2_{\theta,1}(s_k), \sigma^2_{\theta,2}(s_k), ..., \sigma^2_{\theta,n}(s_k))$ is the diagonal covariance matrix, with $\sigma^2_{\theta,l}(s_k)$ representing the variance for the l-th action dimension at time step k.

3.3 Neuroevolving a Fast Low-level Policy

We seek to learn more complex lower-level behavior, beyond prior work of simple goal-reaching low-level policies [25]. However, directly optimizing the episodic return (R) is challenging due to the extended credit assignment horizon, exacerbated by the higher-frequency low-level controller and its increased number of steps (G) per high-level latent action a_k.

Policy gradient methods, while a natural choice for policy optimization, are known to struggle with the long horizons encountered in high-frequency control [9][4]. This difficulty is compounded by the credit assignment problem [10], where the impact of individual low-level actions on the overall return becomes increasingly diffused over longer horizons [28]. Furthermore, the increased temporal density of actions at higher frequencies can lead to heightened sensitivity to policy parameter variations, making optimization landscapes more challenging to navigate and potentially leading to suboptimal solutions. Empirical evidence supporting these challenges is presented in Section J.1.

Instead we seek a learning method that is invariant to the long-horizon credit assignment issue, and has the properties that enable it to discover ideally a globally optimal low-level policy. Motivated by this, we use Neuroevolution, a gradient free, black box, global optimization approach [29, 30]. Neuroevolution leverages Evolutionary Strategies (ES) to optimize the lower-level policy neural network parameters ϕ, by maintaining a population of parameters represented by a distribution $p_\eta(\phi)$, itself parameterized by η and maximizes the average fitness value $\mathbb{E}_{\phi\sim p_\eta} F(\phi)$ over the population by searching for ϕ with stochastic gradient ascent [30]. The core idea rests upon optimizing the score function estimator of

$$\nabla_\phi \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[F(\phi+\sigma\epsilon)] = \frac{1}{\sigma}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[F(\phi+\sigma\epsilon)\epsilon],$$

where $\mathcal{N}(0, I)$ is the standard multivariate normal distribution, and σ is a step size parameter. This estimator allows for gradient estimation without explicit backpropagation by sampling perturbations of ϵ. Therefore Neuroevolution maintains a population of parameters, evaluates their fitness, and generates a new population through selection, mutation (adding noise $\sigma\epsilon$), and recombination, as determined by a specific Neuroevolution algorithm used [30]. ES, while less sample efficient compared to RL, are particularly well-suited for scenarios where gradient-based methods struggle, such as those with delayed-rewards, noisy environments, or long-horizon tasks [30]. Specifically, we use Policy Gradients with Parameter-Based Exploration (PGPE) [31], which optimizes a population of policy parameters by maximizing average fitness using a score function estimator, eliminating the need for backpropagation and making it robust to long horizons and noisy environments. We empirically demonstrate its effectiveness within EvoControl in Section 5.1.

One consideration is what to select as the fitness function $F(\phi)$ for the parameters. Given the bi-level setup, we seek to optimize the episodic return for the combined bi-level policy as a rollout in the environment, i.e. $F(\phi) = R$. Doing so becomes a long horizon optimization problem, especially when the lower-level operates at a higher frequency. Interestingly, directly optimizing the parameters with gradient descent is infeasible due to stochastic noise on the state of the environment and potentially many steps of gradient propagation through an entire bi-level policy rollout of the environment. Crucially, to reduce variance of the fitness function F and improve learning convergence of the lower-level policy, we sample the mode of the probabilistic high-level actor, and parameterize the lower-level actor as a deterministic low-level policy neural network.

Another advantage of using Neuroevolution for lower-level policy optimization is the inherent parallelism of fitness evaluations. This parallelism can lead to faster wall-clock time convergence compared to gradient-based methods, even though evolutionary strategies (ES) generally require more samples [30]. In our experiments, we find that this trade-off is beneficial: the increased

[4] Low-level controllers often operate at 500Hz-1KHz, resulting in $G = 50 - 100$ low-level actions respectively for each high-level action at 10Hz.

sample complexity is outweighed by the ability to stably learn a high-frequency low-level policy (Appendix J.2).

The process of annealing with a PD controller further improves learning a lower-level policy that can be directed with a high-level latent action as input a_k that is directed towards solving the task. Although initially the higher-level policy will provide a latent-action a_k that would be applicable to a particular PD controller that it was initially trained with in the early stages of training, by enabling full optimization of the lower-level policy we can evolve a better performing lower-level policy, lessening the reliance on a tuned PD controller. In practice we find this effective, even if the PD controller is mistuned (Section 5.2). We provide pseudocode for EvoControl in Appendix G.1.

4 Related Work

Here we provide the existing approaches to continuous control, and provide an extended related work in Appendix D.

Fixed Low-Level Controllers: Combining learned low-frequency high-level policies with fixed high-frequency controllers (e.g., PD) [13] simplifies exploration [9, 32] but suffers from limitations. PD parameters require task-specific tuning, and fixed controllers struggle with complex high-frequency interactions (e.g., collisions). Current real-world RL applications are often limited to low-frequency control due to this reliance on analytical controllers [33, 34, 35, 36]. EvoControl, conversely, learns flexible high-frequency behaviors, beyond simple state-goal tracking behavior of fixed-controllers.

Direct Torque Control: End-to-end high-frequency torque control [9, 37, 38] offers flexibility but faces the curse of dimensionality, hindering exploration [33, 9]. EvoControl mitigates this with a hierarchical structure, enabling efficient exploration while retaining low-level adaptability.

Hierarchical Reinforcement Learning (HRL): Methods like options [14, 39] and hierarchical actor-critic [40, 41], including recent deep learning extensions [42, 43, 44, 45], decompose tasks but often focus on discrete skills/subgoals. HRL typically learns simpler low-level policies limited by subgoal attainment, not overall episodic return. EvoControl, inspired by HRL, tackles continuous high-frequency control with meaningful exploration, focusing on a fast, complementary low-level policy, uniquely combining PPO and neuroevolution within its hierarchy.

5 Experiments and Evaluation

In this section, we evaluate EvoControl and verify that it can achieve a higher evaluation reward for the same number of high-level policy steps compared to existing training of a high-level policy either with *fixed controllers* or *direct torque control*.

Benchmark Environments. We evaluate performance on twelve high-dimensional continuous control environments (Appendix E). Ten are standard Gym MuJuCo tasks [46, 47], including locomotion (e.g., Ant, HalfCheetah, Humanoid), and manipulation tasks (e.g., Reacher, Pusher). We also introduce a safety-critical Reacher variant with a randomly positioned obstacle that penalizes collisions. We simulate all environments at 500Hz, with each high-level policy baseline running at 31.25Hz, simulating realistic sensor limitations (e.g., camera-based state estimation).

Table 2: EvoControl Ablation of PD Controllers.

Controller Variant	β_{NN} Obs.	β_{NN} action
EvoControl (Full State)	$s_t, a_k, e_t, q_t, \dot{q}_t, t/T$	τ
EvoControl (Residual State)	$e_t, t/T$	τ
EvoControl (Target + Proprioceptive)	$a_k, q_t, \dot{q}_t, e_t, t/T$	τ
EvoControl (Target)	$a_k, q_t, \dot{q}_t, t/T$	τ
EvoControl (Learned Gains)	$s_t, a_k, q_t, \dot{q}_t, t/T$	K_p, K_d
EvoControl (Delta Position)	$s_t, a_k, e_t, q_t, \dot{q}_t, t/T$	τ

Benchmark Methods. We compare EvoControl against established baselines, using the same high-level PPO[5] policy (ρ) learning algorithm across all, varying only the low-level policy (β). We consider *fixed controllers*: PD Position, PD Position Delta, and PD Integrated Velocity [8]; *direct torque control* at both high (500Hz) and low (31.25Hz) frequencies [49]; a Random policy (30Hz); and several EvoControl ablations with varying state information provided to the low-level neural network controller (Table 2). Here, the EvoControl variants using position-based controllers are annealed from their corresponding PD controllers. Method implementation details are provided in Appendix F.

[5]PPO is able to achieve state-of-the-art performance solving Gym MuJuCo tasks [48, 47].

Table 3: Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±6.56	61.2±0.441	91.6±1.23	100±2.96	100±0.974	99.9±0.03	100±1.53e-06	100±8.47	100±1.8	85.2±2.87	75.7±0.633
Fixed Cont. - PD Position Delta	2.4±1.91	2.76±0.0888	100±1.35	96.6±1.71	2.96±0.0397	53.8±1.57	100±1.53e-06	0±0	40.9±3.23	15.2±7.6	90.2±0.239
Fixed Cont. - PD Int. Velocity	3.59±1.78	2.46±0.0932	74.7±0.903	83.4±1.13	0±0	49.7±1.55	86.5±2	0±0	0±0	0±0	85.9±2.55
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0	17.2±0.316	1.42±0.533	10.4±2.19	10.3±0.586	0±0	0±0	1.34±7.89	2.08±5.84	45.3±6.74	0±0
Direct Torque Cont. - Low Freq. (31.25Hz)	54.5±7.15	100±1.21	72±0.64	98±2.55	80.6±2.56	100±0.0311	100±1.53e-06	73.2±12.9	59.2±3.72	100±1.94	100±2.68
EvoControl (Full State)	**368±73.2**	**157±18.3**	**274±20.6**	**123±19**	**116±18.4**	**101±0.859**	100±0	**362±12.5**	**114±7.51**	**106±2.75**	**203±137**
EvoControl (Residual State)	**182±16.1**	**182±6.31**	**101±5.54**	**170±18.2**	**212±145**	99.2±1.25	100±0	**375±94.8**	**106±25.8**	**104±3.42**	**205±57.4**
EvoControl (Target + Proprio.)	**319±35.1**	**168±14.7**	**171±155**	**165±7.19**	**165±150**	99.7±1.19	100±0	**353±28.4**	96.8±78.6	**105±4.71**	**178±63.7**
EvoControl (Target)	**293±87**	**162±23.5**	**283±250**	**164±21.2**	**205±147**	99.6±0.949	100±0	**353±43.4**	**112±1.73**	**105±1.65**	**188±88.2**
EvoControl (Learned Gains)	**266±104**	**113±10.5**	**206±302**	**150±15.1**	**117±2.44**	99.5±2.48	100±0	**330±17.8**	**116±1.62**	**105±2.45**	**196±118**
EvoControl (Delta Position)	**362±47.7**	**133±34.6**	**225±82.9**	**119±18**	**105±4.64**	**101±0.394**	100±0	**267±30.2**	65.5±21	99.1±12.7	**183±34.4**

Evaluation. We train each policy (high-level ρ and low-level β) for 1M high-level steps. Post-training, we evaluate performance using 128 rollouts (different random seeds) per trained policy, calculating the return for each 1,000-step episode. We repeat this process for three training seeds per baseline. Results are reported as the mean normalized score $\mathcal{R}$ [50] across all 384 evaluation rollouts (3 training seeds x 128 evaluation rollouts), scaled from 0 (random policy performance) to 100 (best non-EvoControl baseline). Further evaluation details are provided in Appendix H.

5.1 Main Results

We evaluated all benchmark methods across all of our environments with results tabulated in Table 3. EvoControl on average achieves high average evaluation normalized return $\mathcal{R}$ on all environments. Specifically, EvoControl can both achieve a high average return $\mathcal{R}$, while learning a slow (31.25Hz) high-level policy, and able to solve a environment task with a fast (500Hz) learned low-level policy. Furthermore, EvoControl is able to outperform direct torque control at high frequency, and outperform the same high-level policy learning algorithm with position PD controllers, and we provide insights in Section 5.2.

5.2 Insight Experiments

Next, we analyze why EvoControl performs better than learning a policy with standard *fixed controllers* or *direct torque control*. We highlight the importance of controlling an environment at high frequency when required, without sacrificing learning convergence ability.

Does EvoControl Possess Efficient Exploration? (P1). To explore if the benchmarked methods during training possess efficient exploration we analyze the learning curves for the Reacher 1D environment and their state space visitation frequency histograms in Figure 2 (with implementation details in Appendix I.1). We observe that EvoControl can initially achieve the same efficient state exploration due to temporal abstractions, similar to that of a fixed PD position controller, and then can further achieve a higher evaluation reward throughout training. This suggests that the higher-frequency control enabled by EvoControl, as theoretically motivated by Proposition 2.1, contributes to finding more optimal actions. This is reminiscent of the principle behind Pulse Width Modulation (PWM), where higher frequency allows for finer control and more accurate signal representation [17]. While a direct equivalence to PWM is not claimed, the ability of high-frequency actions to improve control, as demonstrated in Proposition 2.1, provides a theoretical underpinning for EvoControl's improved performance. Crucially, we observe performing direct high-frequency torque control suffers poor state exploration, given the same number of training steps (Appendix I.1).

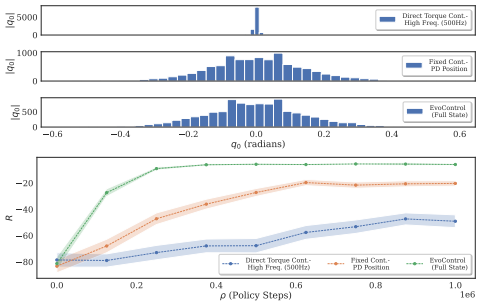

Figure 2: **Top 3 Sub-Plots: State visitation histogram for Reacher 1D.** Empirically demonstrating that PPO with high-frequency-direct torque control suffers from less efficient exploration compared to using PPO at a low-frequency with a fixed-high-frequency PD controller. EvoControl can achieve the same efficient exploration as a PD controller. **Bottom: Evaluation return R versus ρ policy steps on Reacher 1D.** PPO at high-frequency with direct-torque control experiences slower convergence, compared to learning PPO at a low-frequency with a fixed PD controller. EvoControl can evolve its lower-level controller throughout training, leading to a higher evaluation reward in comparison—additional plots are provided in Appendix I.1.

Can EvoControl Learn High-Frequency Interaction Control? (P2). High-frequency control can be crucial for safety-critical tasks like collision avoidance where rapid responses to unexpected contacts are paramount. To investigate such a setting, we adapted the Reacher 1D environment to introduce a random object in 25% of the episodes which blocks the arm from reaching its intended goal, and add both an observation for any measured contact force and a reward penalty for this contact force (*Safety Critical Reacher*). We tabulate the normalized performance of all baselines in Table 4. We observe that EvoControl is able to observe and react faster at high-frequency to un-modelled collisions, compared to a low-frequency policy with a fixed-high-frequency state tracking PD controller. Critically such a collision detection and avoidance environment exemplifies our intuition from Proposition 2.1, that higher-frequency actions can be more optimal. As intuitively a well performing policy, requires a change in behavior as soon as any un-modelled collision is detected, intuitively similar to fast automatic reflexes for a low-level system controller with a high-level system, beyond simple goal-state tracking. We provide experimental details in Appendix I.2.

Table 4: Normalized evaluation return $\mathcal{R}$ for the benchmark methods on the Safety Critical Reacher Environment—using the same normalization as in Table 3. EvoControl can learn to control at a higher frequency than that of the low-level policy frequency, even though it starts with using a PD position control to learn the high-level controller; whereas learning at a low-frequency with a PD position controller cannot observe the collision at a fast enough frequency receiving a low evaluation reward for this environment.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Safety Critical Reacher $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±19.2
Fixed Cont. - PD Position Delta	0±0
Fixed Cont. - PD Int. Velocity	67.1±11.4
Fixed Cont. - Random	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0
Direct Torque Cont. - Low Freq. (31.25Hz)	35.9±17.1
EvoControl (Full State)	**205±15.6**
EvoControl (Residual State)	**124±50.1**
EvoControl (Target + Proprio.)	**237±41.2**
EvoControl (Target)	**121±41.9**
EvoControl (Learned Gains)	**169±123**
EvoControl (Delta Position)	**213±37.5**

Can EvoControl Automate Tuning of PD parameters? (P3). A widespread limitation of any fixed-PD-low-level controller is the inherent sensitivity of its state-tracking performance to that of its fixed gains (K_p, K_d). In practical scenarios such gains require careful manual tuning to each task and environment of operation. Therefore having an approach that can be more robust to tuning PD gains than just their inherent sensitivity is practically useful. Empirically, evaluating on the Reacher 1D task, with varying the $K_p = \{0.001, 0.1, 1.0, 10.0\}$ gain, we observe EvoControl achieving a higher average normalized return $\mathcal{R}$ than a PD controller, crucially as the PD controllers become less tuned $K_p \to 0$ their performance decreases, highlighting the sensitivity of these fixed controllers. This highlights EvoControls robustness to PD parameters, which could arise due to the PD controller provides a semantically meaningful initial latent action a_k that the lower-level policy can then refine. Ablating the initial PD controller annealing in EvoControl destabilizes learning of both the high-level and low-level policies, confirming its importance (Appendix J.4).

Table 5: Normalized evaluation return $\mathcal{R}$ for the benchmark methods on Reacher 1D—using the same normalization as in Table 3. EvoControl is more robust to the tuning of the underlying PD controller than existing fixed PD controllers, which can degrade as their PD controller parameter (K_p) becomes less tuned for the task.

Same PPO high-level alg. ρ with a Low-Level Policy β of	$K_p = 0.001$ $\mathcal{R}\uparrow$	$K_p = 0.1$ $\mathcal{R}\uparrow$	$K_p = 1.0$ $\mathcal{R}\uparrow$	$K_p = 10.0$ $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	0±0	13.5±7.39	81.8±2.81	100±0.893
Fixed Cont. - PD Position Delta	0±0	0±0	14.6±7.31	78.8±3.58
Fixed Cont. - PD Int. Velocity	0±0	0±0	0±0	80.5±4.17
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	43.6±6.48	43.6±6.48	43.6±6.48	43.6±6.48
Direct Torque Cont. - Low Freq. (31.25Hz)	96.2±1.87	96.2±1.87	96.2±1.87	96.2±1.87
EvoControl (Full State)	**99.1±3.13**	**99.2±7.07**	**102±3.25**	**101±0.36**
EvoControl (Residual State)	**100±4.55**	**99.5±5.62**	**100±3.29**	**102±2.18**
EvoControl (Target + Proprio.)	**98.6±3.5**	**97.9±1.48**	**101±4.43**	**101±0.894**
EvoControl (Target)	94.6±6.85	**96.5±10.1**	**101±1.58**	**100±0.995**
EvoControl (Learned Gains)	91.8±5.37	93.4±6.69	**101±2.35**	**100±3.04**
EvoControl (Delta Position)	95.8±8.29	**97.4±1.92**	95.4±12.2	**97.6±8.52**

6 Conclusion and Future Work

In this paper we present *EvoControl*, a novel bi-level policy learning framework for learning both a slow high-level policy and a fast low-level controller using PPO and Neuroevolution, respectively, for continuous-control tasks. Theoretically, we show the existence of MDPs where infinitely high action frequency is necessary for optimal control. Empirically, EvoControl outperforms existing high-frequency control methods, particularly in tasks requiring fast reactions. Moreover the limitations of the current approach, are that EvoControl still relies on the existence of a fixed-PD controller for the task (common in robotics applications, Appendix J.4) and can require more computational complexity compared to only performing PPO, which can be readily parallelized in practice with modern accelerated compute platforms, both could be readily improved. In addition, promising future directions include exploring more complex nested hierarchies, direct low-level to high-level information flow, and ensembles of policies (Appendix K).

Acknowledgments

We thank the anonymous reviewers, and area chairs, and specifically Francesco Nori, Leonard Hasenclever, Steven Bohez, Thomas Lampe, Nimrod Gileadi and Jose Enrique Chen for their insightful comments and suggestions that ultimately improved this work. This work was supported by Google DeepMind.

References

[1] N. Hogan. Impedance control: An approach to manipulation. In *1984 American control conference*, pages 304–313. IEEE, 1984.

[2] S. Oh, H. Woo, and K. Kong. Frequency-shaped impedance control for safe human–robot interaction in reference tracking application. *IEEE/ASME Transactions On Mechatronics*, 19 (6):1907–1916, 2014.

[3] S. Venkataraman and S. Gulati. Terminal slider control of robot systems. *Journal of Intelligent and Robotic Systems*, 7:31–55, 1993.

[4] E. Dantec, M. Taix, and N. Mansard. First order approximation of model predictive control solutions for high frequency feedback. *IEEE Robotics and Automation Letters*, 7(2):4448–4455, 2022.

[5] K. J. Kuchenbecker, J. Gewirtz, W. McMahan, D. Standish, P. Martin, J. Bohren, P. J. Mendoza, and D. I. Lee. Verrotouch: High-frequency acceleration feedback for telerobotic surgery. In *Haptics: Generating and Perceiving Tangible Sensations: International Conference, EuroHaptics 2010, Amsterdam, July 8-10, 2010. Proceedings, Part I*, pages 189–196. Springer, 2010.

[6] J. Guo, U. Kurup, and M. Shah. Is it safe to drive? an overview of factors, metrics, and datasets for driveability assessment in autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 21(8):3135–3151, 2019.

[7] M. Vasic and A. Billard. Safety issues in human-robot interactions. In *2013 ieee international conference on robotics and automation*, pages 197–204. IEEE, 2013.

[8] E. Aljalbout, F. Frank, M. Karl, and P. van der Smagt. On the role of the action space in robot manipulation learning and sim-to-real transfer. *IEEE Robotics and Automation Letters*, 2024.

[9] X. B. Peng and M. Van De Panne. Learning locomotion skills using deeprl: Does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–13, 2017.

[10] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

[11] W. Dabney, G. Ostrovski, and A. Barreto. Temporally-extended ϵ-greedy exploration, 2020.

[12] B. Wei. A tutorial on robust control, adaptive control and robust adaptive control—application to robotic manipulators. *Inventions*, 4(3):49, 2019.

[13] P. Song, Y. Yu, and X. Zhang. A tutorial survey and comparison of impedance control on robotic manipulation. *Robotica*, 37(5):801–836, 2019.

[14] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[15] Y. Ma, Z. Song, Y. Zhuang, J. Hao, and I. King. A survey on vision-language-action models for embodied ai. *arXiv preprint arXiv:2405.14093*, 2024.

[16] R. P. Borase, D. Maghade, S. Sondkar, and S. Pawar. A review of pid control, tuning methods and applications. *International Journal of Dynamics and Control*, 9:818–827, 2021.

[17] J. Huang, K. Padmanabhan, and O. M. Collins. The sampling theorem with constant amplitude variable width pulses. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(6): 1178–1190, 2011.

[18] D. E. Oku, E. P. Obot, and C. Author. Comparative study of pd, pi and pid controllers for control of a single joint system in robots. *Int. J. Eng. Sci*, 7(9):V2, 2018.

[19] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, 10, 1997.

[20] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '11, page 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011. ISBN 0982657161.

[21] S. Li, J. Zhang, J. Wang, Y. Yu, and C. Zhang. Active hierarchical exploration with stable subgoal representation learning. *arXiv preprint arXiv:2105.14750*, 2021.

[22] W. McClinton, A. Levy, and G. Konidaris. Hac explore: Accelerating exploration with hierarchical reinforcement learning. *arXiv preprint arXiv:2108.05872*, 2021.

[23] S. Chiaverini, B. Siciliano, and L. Villani. A survey of robot interaction control schemes with experimental comparison. *IEEE/ASME Transactions on mechatronics*, 4(3):273–285, 1999.

[24] X. Yang, Z. Ji, J. Wu, Y.-K. Lai, C. Wei, G. Liu, and R. Setchi. Hierarchical reinforcement learning with universal policies for multistep robotic manipulation. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):4727–4741, 2021.

[25] O. Nachum, S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

[26] J. Wöhlke, F. Schmitt, and H. van Hoof. Hierarchies of planning and reinforcement learning for robot navigation. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 10682–10688. IEEE, 2021.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[28] P. Dayan and G. E. Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

[29] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

[30] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[31] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[32] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

[33] R. Martín-Martín, M. A. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. In *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 1010–1017. IEEE, 2019.

[34] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, and A. M. Agogino. Deep reinforcement learning for robotic assembly of mixed deformable and rigid objects. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2062–2069. IEEE, 2018.

[35] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine. Residual reinforcement learning for robot control. In *2019 international conference on robotics and automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[36] T. Davchev, K. S. Luck, M. Burke, F. Meier, S. Schaal, and S. Ramamoorthy. Residual learning from demonstration: Adapting dmps for contact-rich manipulation. *IEEE Robotics and Automation Letters*, 7(2):4488–4495, 2022.

[37] N. Wahlström, T. B. Schön, and M. P. Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015.

[38] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *Advances in neural information processing systems*, 28, 2015.

[39] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[40] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. van de Wiele, V. Mnih, N. Heess, and J. T. Springenberg. Learning by playing - solving sparse reward tasks from scratch. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[41] G. Vezzani, D. Tirumala, M. Wulfmeier, D. Rao, A. Abdolmaleki, B. Moran, T. Haarnoja, J. Humplik, R. Hafner, M. Neunert, et al. Skills: Adaptive skill sequencing for efficient temporally-extended exploration. *arXiv preprint arXiv:2211.13743*, 2022.

[42] D. Rao, F. Sadeghi, L. Hasenclever, M. Wulfmeier, M. Zambelli, G. Vezzani, D. Tirumala, Y. Aytar, J. Merel, N. M. O. Heess, and R. Hadsell. Learning transferable motor skills with hierarchical latent mixture policies. *ArXiv*, abs/2112.05062, 2021.

[43] S. Salter, M. Wulfmeier, D. Tirumala, N. Heess, M. Riedmiller, R. Hadsell, and D. Rao. Mo2: Model-based offline options. In *Conference on Lifelong Learning Agents*, pages 902–919. PMLR, 2022.

[44] M. Wulfmeier, A. Abdolmaleki, R. Hafner, J. Tobias Springenberg, M. Neunert, N. Siegel, T. Hertweck, T. Lampe, N. Heess, and M. Riedmiller. Compositional transfer in hierarchical reinforcement learning. *Robotics: Science and Systems XVI*, Jul 2020. doi:10.15607/rss.2020.xvi.054. URL `http://dx.doi.org/10.15607/rss.2020.xvi.054`.

[45] M. Wulfmeier, D. Rao, R. Hafner, T. Lampe, A. Abdolmaleki, T. Hertweck, M. Neunert, D. Tirumala, N. Siegel, N. Heess, and M. Riedmiller. Data-efficient hindsight off-policy option learning, 2020.

[46] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[47] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[48] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International conference on learning representations*, 2019.

[49] S. Chen, B. Zhang, M. W. Mueller, A. Rai, and K. Sreenath. Learning torque control for quadrupedal locomotion. In *2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids)*, pages 1–8. IEEE, 2023.

[50] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[51] J. R. Norris. *Markov chains*. Number 2. Cambridge university press, 1998.

[52] N. Chentanez, M. Müller, M. Macklin, V. Makoviychuk, and S. Jeschke. Physics-based motion capture imitation with deep reinforcement learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, pages 1–10, 2018.

[53] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):143, 2018.

[54] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. Panne. Learning locomotion skills for cassie: Iterative design and sim-to-real. In *Conference on Robot Learning*, pages 317–329. PMLR, 2020.

[55] O. Nachum, S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018.

[56] S. Abramowitz and G. Nitschke. Scalable evolutionary hierarchical reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 272–275, 2022.

[57] D. Reda, T. Tao, and M. van de Panne. Learning to locomote: Understanding how environment design matters for deep reinforcement learning. In *Proceedings of the 13th ACM SIGGRAPH conference on motion, interaction and games*, pages 1–10, 2020.

[58] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

[59] O. Sigaud. Combining evolution and deep reinforcement learning for policy search: A survey. *ACM Transactions on Evolutionary Learning*, 3(3):1–20, 2023.

[60] K. Suri, X. Q. Shi, K. N. Plataniotis, and Y. A. Lawryshyn. Maximum mutation reinforcement learning for scalable control. *arXiv preprint arXiv:2007.13690*, 2020.

[61] S. Khadka and K. Tumer. Evolution-guided policy gradient in reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[62] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. Stanley, and J. Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *Advances in neural information processing systems*, 31, 2018.

[63] P. Li, J. Hao, H. Tang, X. Fu, Y. Zhen, and K. Tang. Bridging evolutionary algorithms and reinforcement learning: A comprehensive survey on hybrid algorithms. *IEEE Transactions on Evolutionary Computation*, 2024.

[64] C. Bodnar, B. Day, and P. Lió. Proximal distilled evolutionary reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3283–3290, 2020.

[65] H. Zheng, P. Wei, J. Jiang, G. Long, Q. Lu, and C. Zhang. Cooperative heterogeneous deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:17455–17465, 2020.

[66] T. N. Mundhenk, M. Landajuela, R. Glatt, C. P. Santiago, D. M. Faissol, and B. K. Petersen. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*, 2021.

[67] S. Elfwing, E. Uchibe, K. Doya, and H. I. Christensen. Evolutionary development of hierarchical learning structures. *IEEE transactions on evolutionary computation*, 11(2):249–264, 2007.

[68] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. AraÃšjo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[69] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[70] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, et al. Jax: Autograd and xla. *Astrophysics Source Code Library*, pages ascl–2111, 2021.

[71] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[72] Y. Tang, Y. Tian, and D. Ha. Evojax: Hardware-accelerated neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 308–311, 2022.

[73] C. Lu, J. Kuba, A. Letcher, L. Metz, C. Schroeder de Witt, and J. Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468, 2022.

[74] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI gym. *CoRR*, abs/1606.01540, 2016.

Appendix

Table of Contents

A Expanded Problem

In the following we expand the problem setup from the main paper.

States & Actions. We denote the environments state space as $\mathcal{S} \subset \mathbb{R}^{d_s}$ and its action space as $\mathcal{U} \subset \mathbb{R}^{d_u}$. At time $t \in \mathbb{R}$, the system's state is represented by $\mathbf{s}_t \in \mathcal{S}$, and its action by $\mathbf{u}_t \in \mathcal{U}$. Considering action (e.g. actuator) limits the action space is constrained to a box in Euclidean space: $\mathcal{U} = [\mathbf{u}_{\text{min}}, \mathbf{u}_{\text{max}}]$.

Environment Dynamics. The transition dynamics for continuous control environments can be described by an underlying unknown *differential equation* of $\dot{s}_t = \frac{ds_t}{dt} = f(s_t, u_t)$. The transition function, which describes the evolution of the state over a discrete time step Δ_t, can be approximated using the Euler method $s_{t+\Delta_t} \approx s_t + \Delta_t f(s_t, u_t)$. Given an action u_t and current state s_t, $s_{t+\Delta_t} \sim P(s_{t+\Delta_t}|s_t, u_t)$ is implicitly defined by this approximation. More sophisticated numerical integration schemes (e.g., Runge-Kutta methods) can be employed for higher accuracy. In stochastic environments, the dynamics function f can be considered to be stochastic, leading to a probability distribution over next states given the current state and action. We consider the setting where there is an additional observation function that maps the current environment state to an underlying observation $z_t = g(s_t) + \epsilon$, where ϵ is optional observation noise, e.g. Gaussian noise with zero mean $\epsilon_t \sim \mathcal{N}(0, \sigma_\epsilon^2)$[6]. To simplify notation we use observation and state interchangeably and clarify the specifics when needed.

Policies. The agent can be represented as a single policy $\pi : \mathbb{R}^{d_s} \to \mathbb{R}^{d_u}$, that observes the current observation at time t and samples an action $u_t \sim \pi(s_t)$ and then applies this action to the environment at a given fixed Δ_t. In the case of a stochastic policy, the action is sampled from a distribution conditioned on the state: $u_t \sim \pi(\cdot|s_t)$.

To formalize a bi-level policy, we decompose π into two components: a slow, high-level policy, ρ, and a fast, low-level policy, β. Both policies interact with the environment as described in Algorithm 1. The high-level policy ρ operates at a lower frequency and outputs a high-level (latent) action, $a_k \sim \rho(s_k)$, at time step k. This latent action often represents a desired high-level target, such as a target position or velocity. The low-level policy β operates at a higher frequency and receives the high-level action a_k as input. β then generates the low-level actions, u_{k+i}, at a finer-grained time step i, corresponding to direct motor torques or other low-level control signals. These low-level actions aim to achieve the high-level target specified by ρ. We denote the high-level time steps as k and low-level time steps as i to maintain this distinction. The fast, low-level policy β operates at a frequency of $1/\Delta_t$, where Δ_t is the low-level time step. The slow, high-level policy ρ guides the low-level policy over a longer horizon. Specifically, ρ issues a latent action which is executed by β for G steps. This means ρ effectively operates at a frequency of $1/(G\Delta_t)$ with a time step of $G\Delta_t$.

Objective. The environment produces a reward r_t sampled from an unknown reward function $r(s_t, u_t)$, $r : \mathcal{S} \times \mathcal{U} \to \mathbb{R}$. The overall objective of the agent is to maximize the expected future discounted reward $\mathbb{E}_{s_0:T, u_0:T-1, R_0:T-1}\left[\sum_{i=0}^{T-1} \gamma^i r_i\right]$, where $0 \leq \gamma < 1$ is the discount factor.

Markov Decision Process (MDP). We can model the environment as a Markov Decision Process (MDP), defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{U}, P, r, \gamma \rangle$, where:

- $\mathcal{S} \subset \mathbb{R}^{d_s}$ is the continuous state space.
- $\mathcal{U} \subset \mathbb{R}^{d_u}$ is the continuous action space.
- $P(s_{t+\Delta_t}|s_t, u_t)$ is the state transition probability distribution, implicitly defined by the dynamics function f and the discretization scheme (e.g., Euler method).
- $r : \mathcal{S} \times \mathcal{U} \to \mathbb{R}$ is the reward function, providing a scalar reward $r_t = r(s_t, u_t)$ at each time step.
- $\gamma \in [0, 1)$ is the discount factor, determining the importance of future rewards.

[6]For the standard MuJoCo Brax environments we use, the joint velocity observation has Gaussian noise added to it, following the standard implementation of the environments [47].

B Proof of Proposition 2.1: Optimality of High-Frequency Control

Proposition B.1. *There exist Markov Decision Processes (MDPs) where the optimal control policy, maximizing the expected cumulative reward over a fixed episode duration T, requires an action frequency approaching infinity.*

Full Set of Assumptions:

1. **Reward Magnitudes:** $r_{\text{good}} > 0$ and $r_{\text{bad}} < 0$. This ensures that maximizing time in s_{good} and minimizing time in s_{bad} maximizes the cumulative reward.
2. **Transition Probabilities:** The transition probabilities are derived from a continuous-time Markov process with constant transition rates. When discretized with time step Δ_t, the transition probabilities are accurate up to first order in Δ_t, with error terms of $O(\Delta_t^2)$ converging uniformly to zero as $\Delta_t \to 0$. This is a standard assumption when discretizing continuous-time Markov processes, justifiable by the Taylor expansion of the matrix exponential representing the continuous-time transition probabilities (see, e.g., Norris [51], Chapter 2). Specifically, if the transition rate from s_i to s_j is q_{ij}, then $P(s_j|s_i, a, \Delta_t) = q_{ij}\Delta_t + O(\Delta_t^2)$ for $i \neq j$ and $P(s_i|s_i, a, \Delta_t) = 1 - \sum_{j \neq i} q_{ij}\Delta_t + O(\Delta_t^2)$. We also assume $0 < \Delta_t \leq 1/p$ to ensure probabilities remain within [0, 1].
3. **Time Horizon:** The time horizon $T > 0$ is fixed. We consider the limit as $\Delta_t \to 0$ while holding T constant. For any considered fixed $\Delta_t > 0$, T is an integer multiple of Δ_t. This simplifies notation without loss of generality. Furthermore, for any fixed Δ_t, the number of time steps T/Δ_t is assumed to be sufficiently large to allow the system to closely approximate its stationary distribution under the low-frequency policy. This ensures the accuracy of the approximation used for the low-frequency policy's expected reward by allowing the transient behavior of the Markov chain to become negligible, thus permitting the application of the ergodic theorem [51].

Proof. **Markov Decision Process (MDP) Definition:** The MDP is derived from a continuous-time Markov process, discretized with a time step Δ_t.

- **State Space:** $S = \{s_{\text{good}}, s_{\text{bad}}\}$
- **Action Space:** $A = \{a_{\text{maintain}}, a_{\text{recover}}\}$
- **Time Step:** Δ_t (time between actions), $0 < \Delta_t \leq 1/p$
- **Transition Probabilities (derived from continuous-time rate $p > 0$):**

$$P(s_{\text{good}}|s_{\text{good}}, a_{\text{maintain}}, \Delta_t) = 1 - p\Delta_t + O(\Delta_t^2) \tag{1}$$
$$P(s_{\text{bad}}|s_{\text{good}}, a_{\text{maintain}}, \Delta_t) = p\Delta_t + O(\Delta_t^2) \tag{2}$$
$$P(s_{\text{good}}|s_{\text{good}}, a_{\text{recover}}, \Delta_t) = 1 \tag{3}$$
$$P(s_{\text{bad}}|s_{\text{good}}, a_{\text{recover}}, \Delta_t) = 0 \tag{4}$$
$$P(s_{\text{bad}}|s_{\text{bad}}, a_{\text{maintain}}, \Delta_t) = 1 \tag{5}$$
$$P(s_{\text{good}}|s_{\text{bad}}, a_{\text{maintain}}, \Delta_t) = 0 \tag{6}$$
$$P(s_{\text{good}}|s_{\text{bad}}, a_{\text{recover}}, \Delta_t) = 1 \tag{7}$$
$$P(s_{\text{bad}}|s_{\text{bad}}, a_{\text{recover}}, \Delta_t) = 0 \tag{8}$$

- **Reward Function:** $r(s_{\text{good}}, a) = r_{\text{good}} > 0$; $r(s_{\text{bad}}, a) = r_{\text{bad}} < 0$. Note that the reward is only a function of the state.
- **Discount Factor:** $\gamma = 1$ (undiscounted)
- **Episode Duration:** T

- **Initial State:** s_{good}

Cumulative Reward Definition: The cumulative reward $R(\pi, T)$ for a policy π over a time horizon T is defined as:

$$R(\pi, T) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T/\Delta_t - 1} r(s_t, a_t)\right]$$

where the expectation is taken over the trajectories induced by policy π.

Proof Steps:

1) Optimal Policy ($\pi^*(\Delta_t)$ and π_{HF}): For any fixed $\Delta_t > 0$, the optimal policy $\pi^*(\Delta_t)$ is to apply a_{recover} in s_{bad} and a_{maintain} in s_{good}. This is because the recovery is instantaneous and always improves the reward, so any delay in recovery reduces the cumulative reward. The high-frequency policy π_{HF} is defined as the limiting behavior of $\pi^*(\Delta_t)$ as $\Delta_t \to 0$, which maintains the same action choices.

2) Expected Cumulative Reward (High-Frequency, π_{HF} as $\Delta_t \to 0$): As $\Delta_t \to 0$, the time spent in s_{bad} approaches 0 because the recovery is instantaneous. Thus, the expected cumulative reward approaches:

$$R(\pi_{HF}, T) = T r_{\text{good}} \tag{9}$$

3) Expected Cumulative Reward (Low-Frequency, $\pi^*(\Delta_t)$ with fixed $\Delta_t > 0$): For a fixed Δ_t, let μ_{good} and μ_{bad} be the stationary distribution probabilities of being in states s_{good} and s_{bad} respectively under the optimal policy $\pi^*(\Delta_t)$. Solving the balance equations of the Markov chain gives $\mu_{\text{bad}} = \frac{p\Delta_t}{1+p\Delta_t}$ and $\mu_{\text{good}} = \frac{1}{1+p\Delta_t}$. Using the Taylor series expansions for small Δ_t, we have $\mu_{good} \approx 1 - p\Delta_t$ and $\mu_{bad} \approx p\Delta_t$. By the ergodic theorem [51], as $T/\Delta_t \to \infty$, the expected cumulative reward is:

$$\begin{aligned}
R(\pi^*(\Delta_t), T) &\approx T[\mu_{\text{good}} r_{\text{good}} + \mu_{\text{bad}} r_{\text{bad}}] && (10)\\
&\approx T[(1 - p\Delta_t) r_{\text{good}} + (p\Delta_t) r_{\text{bad}}] && (11)\\
&\approx T[r_{\text{good}} - p\Delta_t r_{\text{good}} + p\Delta_t r_{\text{bad}}] && (12)\\
&\approx T r_{\text{good}} - T p\Delta_t r_{\text{good}} + T p\Delta_t r_{\text{bad}}] && (13)\\
&\approx T r_{\text{good}} - T p\Delta_t (r_{\text{good}} - r_{\text{bad}}). && (14)
\end{aligned}$$

4) Comparison: The difference in expected rewards is:

$$\begin{aligned}
\Delta R(T, \Delta_t) &= R(\pi_{HF}, T) - R(\pi^*(\Delta_t), T) && (15)\\
&\approx T r_{\text{good}} - [T r_{\text{good}} - T p\Delta_t (r_{\text{good}} - r_{\text{bad}})] && (16)\\
&\approx T r_{\text{good}} - T r_{\text{good}} + T p\Delta_t (r_{\text{good}} - r_{\text{bad}}) && (17)\\
&\approx T p\Delta_t (r_{\text{good}} - r_{\text{bad}}). && (18)
\end{aligned}$$

Since $r_{\text{good}} > r_{\text{bad}}$ and $p > 0$, we have $\Delta R(T, \Delta_t) > 0$ for any fixed, sufficiently small $\Delta_t > 0$. Therefore, $R(\pi_{HF}, T) > R(\pi^*(\Delta_t), T)$ for sufficiently small Δ_t. As Δ_t can be arbitrarily small, the optimal action frequency $1/\Delta_t$ can be arbitrarily large, demonstrating the existence of MDPs where the optimal control policy requires an action frequency approaching infinity. □

Explanation of Assumptions: The reward assumption creates a meaningful optimization problem. The transition probability assumption reflects a standard discretization of a continuous-time Markov process. Assumption 3 regarding T simplifies the analysis and allows the use of the ergodic theorem to accurately approximate the expected reward of the low-frequency policy by its stationary distribution, since T is large enough that the transient phase becomes negligible. This clarifies that we compare policies where the low-frequency policy has had sufficient time to express its long-term behavior. The assumption of a fixed T as $\Delta_t \to 0$ provides a consistent basis for comparing high and low frequency policies [51].

B.1 Intuitive Continuous Control Safety-Critical Example

Consider a safety-critical task involving a one-degree-of-freedom robot arm. The arm's state is its joint angle θ_t, and the action is the motor torque τ_t. The goal is to reach a target angle θ_{goal} from an initial angle θ_0 within a fixed episode duration T. An immovable obstacle may appear in a random subset of episodes (e.g., 25% of the time), obstructing the direct path to the goal. The reward function encourages reaching the target angle while penalizing contact forces with the obstacle:

$$R = \int_0^T \left(r_{\text{goal}} e^{-|\theta_t - \theta_{\text{goal}}|} - r_{\text{collision}} F_t \right) dt$$

where r_{goal} and $r_{\text{collision}}$ are positive weighting constants, and F_t is the magnitude of the contact force between the arm and the obstacle at time t (0 if no contact). The policy receives the observation $O_t = (\theta_t, \dot{\theta}_t, \tau_{t,\text{measured}})$, where $\tau_{t,\text{measured}}$ is the measured torque, reflecting contact forces if any.

A standard approach might use a position PD controller with a low-frequency high-level policy that provides the target angle θ_{target}. However, this approach faces limitations. If the obstacle is present, the PD controller will exert a continuous force against it, incurring significant penalties. The low-frequency policy might only detect the collision after a substantial delay, making it difficult to react effectively.

A high-frequency policy, on the other hand, can detect the collision much faster and take corrective action. Upon detecting a sudden increase in $\tau_{t,\text{measured}}$, it can immediately reduce the motor torque, minimizing the contact force F_t. Furthermore, a sophisticated high-frequency policy can learn to approach the target cautiously, probing for the obstacle with small torques. If contact is detected, it can adjust its trajectory to reach the goal while avoiding further collisions.

This intuitive example illustrates how high-frequency control can be crucial for safety-critical tasks. It enables faster reaction to unexpected events and allows for more nuanced control strategies that consider the full reward structure, including collision avoidance. This motivates the development of methods like EvoControl, capable of effectively learning such high-frequency policies. This example highlights scenarios where high-frequency control offers a significant advantage over traditional low-frequency control coupled with fixed controllers, especially in tasks requiring rapid responses and nuanced interaction behaviors.

C Expanded Background: Fixed PD Controllers

Low-level PD controllers are extensively used within robotics applications, specifically when combined with a learned high-level policy ρ. This hierarchical structure simplifies the learning problem and effectively reduces the number of decision steps for the high-level policy. This reduction is achieved by allowing the high-level policy to operate at a timestep of $G\Delta_t$, where G is the number of low-level actions executed per high-level action, effectively reducing the number of high-level actions within a fixed episode duration T.

In continuous control and robotics, these hierarchical structures, composed of a learned high-level policy (ρ) and a fixed low-level controller (e.g., a PD controller [18]), are common [9, 13, 52, 53, 54]. The high-level policy outputs a target a_k which the low-level PD controller tracks using a control signal based on the error between a_k and the measured system state s_t.

Commonly, PD control is designed to track a second-order signal, such as position or velocity. The control signal, u_t, is given by:

$$u_t = K_p(a_k - s_t) + K_d(\dot{a}_k - \dot{s}_t), \tag{19}$$

where $K_p, K_d \in \mathbb{R}^+$ are constant proportional and derivative gains, and $e_t = a_k - s_t$ represents the tracking error.

Specifically for robotics, proprioceptive observed states can be represented as joint positions (q_t), joint velocities ($\dot{q}_t$), and torques (τ_t). This leads to several common PD control designs, summarized in

Table 1. These designs differ in how the high-level target a_k is interpreted and used in the control law. For instance, in "PD Absolute Position," a_k directly specifies the desired joint position (q^d), while in "PD Delta Position," a_k represents a change in joint position (δq^d) relative to the current position. The state s_t in the control law can encompass a wider range of proprioceptive information beyond just joint positions (q_t). We present the target a_k and tracking error in terms of position/velocity for clarity and to align with common PD controller formulations.

D Extended Related Work

Table 6: **Comparison with related bi-level learning approaches in RL.** Our method, EvoControl, can achieve efficient state-action space exploration, whilst learning high-frequency interaction behavior, and avoids tuning of PD parameters.

Approach	Ref.	π	Low-level β Reward	High-level Action Duration Δ_ρ	High-level Action ρ	(P1) Efficient Exploration	(P2) High-Frequency Interaction Control	(P3) Automate Controller Tuning PD Parameters
Fixed Controllers	[13]	$\{\rho_{\text{nn}}, \beta_{\text{pd\_controller}}\}$	$-\|\|s(t) - s_{\text{desired}}(t)\|\|_2$	$G\Delta_t$	$\{q_d, \dot{q}_d, \tau_d\}$	✓	✗	✗
Direct Torque Control	[9]	$\{\rho\}$	$R = \sum_{i=0}^{T/\Delta_t - 1} r(s_{i\Delta_t}, u_{i\Delta_t})\Delta_t$	Δ_t	$a(t)$	✗	✓	✓
HRL: Skills	[14, 42]	$\{\rho_{\text{manager}}, \{\beta_0, \beta_1, \ldots, \beta_n\}\}, n \in \mathcal{Z}_n$	$R = \sum_{i=0}^{T/\Delta_t - 1} r(s_{i\Delta_t}, u_{i\Delta_t})\Delta_t$	$z_n \in \mathcal{Z}_n$	✗	✓	✓	
HRL: Sub Goals	[55]	$\{\rho, \beta\}$	$-\|\|s(t) - s_{\text{desired}}(t)\|\|_2$	$G\Delta_t$	$s_{\text{desired}}(t)$	✓	✗	✓
EvoControl	**(Ours)**	$\{\rho, \beta\}$	$R = \sum_{i=0}^{T/\Delta_t - 1} r(s_{i\Delta_t}, u_{i\Delta_t})\Delta_t$	$G\Delta_t$	$z \in \{q_d, \dot{q}_d, \tau_d\}$	✓	✓	✓

Existing approaches to continuous control in robotics primarily fall into two categories: those employing fixed low-level controllers and those utilizing direct torque control learned end-to-end. EvoControl aims to addresses limitations inherent in both approaches, and we summarize the key differences in Table 6.

Fixed Low-Level Controllers: A common strategy involves combining a learned high-level policy (often operating at low frequency, e.g., 10-30Hz) with a fixed, high-frequency low-level controller (e.g., a Proportional-Derivative (PD) controller operating at 500Hz or higher) [13, 52, 53, 54]. The high-level policy generates setpoints (e.g., desired positions or velocities), and the low-level controller tracks these setpoints by adjusting actuator torques. While prevalent, this approach suffers from several drawbacks. The low-level controller's parameters require careful tuning, and its fixed nature limits its ability to handle high-frequency interactions such as unexpected collisions or disturbances. Furthermore, recent work applying RL algorithms to the physical world often restricts itself to relatively low-frequency control ($\sim$20Hz) due to the reliance on analytical impedance controllers [33, 34, 35]. Even hierarchical approaches employing analytical controllers often limit high-level policy frequencies [36]. EvoControl, in contrast, aims to achieve efficient exploration while also enabling the learning of flexible and complex high-frequency behaviors in the low-level policy.

Evolutionary Strategies: Direct evolutionary strategies have been shown to provide an alternative for solving reinforcement learning environments; however the direct application of them, as shown by others are that they can be sample inefficient, get stuck in global minima; however excel at discovering good performing long-horizon tasks, sparse reward tasks and delayed reward tasks, as they often optimize the episodic return, rather than the intermediate temporal difference return [30]. There exist works formulating hierarchical ES for both levels, however still under-perform gradient-based RL policy methods [56]. EvoControl through it's novel combination of a PPO learned high-level policy, and a Neuroevolved low-level policy empirically outperforms the ablation version of using Neuroevolution for both the high-level and low-level in EvoControl, as shown in Appendix J.8.

Direct Torque Control: Alternatively, some methods learn an end-to-end policy that directly outputs joint torques at a high frequency [9, 37, 38]. This approach, while potentially offering greater adaptability, faces significant challenges. High-frequency control suffers from the curse of dimensionality imposed by the increased number of time steps in long horizons. The resulting explosion in the number of possible action sequences significantly hinders exploration and can lead to suboptimal policies [33, 9]. EvoControl mitigates these challenges by employing a hierarchical structure, enabling more efficient exploration while retaining the adaptability afforded by direct torque control at the low level. Moreover, the related work of Peng and Van De Panne [9] compares learning policies with four different action spaces of direct torque control, PD position control, PD velocity control and a muscle activation's for the task of imitating gaits for planar walking robot environments (continuous control). Their findings correlate with ours, in that they observed on

average faster learning convergence and higher task reward using a low-level high-frequency (fast) controller, such as PD controller compared to performing direct torque control. Additionally Peng and Van De Panne [9] due to having no prior controller parameters for the environments that they wanted to control, Peng and Van De Panne [9] similarly performed a related approach where they optimized the fixed low-level controller parameters throughout training a high-level policy. However, all of their low-level controllers used are simple, few parameter (2-7) controllers, such as a PD controller, and such fixed simple controllers are all only capable of sub-goal simple tracking behavior. Whereas EvoControl, can represent the fast lower-level policy with a neural network policy and learn this throughout training the high-level policy, learning fast adaptive behavior of the low-level policy, that goes beyond simple sub-goal tracking behavior. Furthermore, the related work of Reda et al. [57] studied environment design for continuous control tasks, and found that varying the control frequency of performing direct torque control in standard Mujoco Gym like environments (e.g. Ant, Hopper) could yield better learning and overall policy return, however requires tuning the control frequency (or discrete action repeats of the simulation timestep Δ_t) for each environment and task to get the best performance—likely due to matching the inherent control frequency of the dynamics of the environment. They also studied the use of learning with PD controllers, and determined that PD controllers can aid in converging faster to good policy, however can get stuck in lower-reward solutions (local minima), motivating the need for a method to practically perform high-frequency torque control. In summary, EvoControl can overall learn a high-frequency policy π by learning both a slow-high-level policy ρ combined with a fast-low-level policy β, learning adaptive low-level behavior of an equivalent high-frequency policy, avoiding the difficulties of learning a direct torque control high-frequency policy directly. Furthermore, we provide empirical evidence for the difficulty of learning a direct torque control high-frequency policy, as even with an ever increasing number of training steps, such a policy may converge to local minima Appendix J.2.

Hierarchical Reinforcement Learning (HRL): EvoControl draws inspiration from the HRL paradigm, which decomposes complex tasks into simpler subtasks managed by separate policies. Existing HRL methods such as options frameworks [14, 39] and hierarchical actor-critic architectures [40, 41] have been successfully applied to improve exploration and learning efficiency. However, these methods typically focus on discrete skill selection or subgoal decomposition [58], while EvoControl explicitly addresses the challenges of learning a low-level controller for continuous high-frequency control, enabling semantically meaningful exploration in different control modes. Related work in the RHPO/HO2/MO2/HeLMS family [42, 43, 44, 45] has also explored hierarchical approaches. However, unlike typical HRL, which focuses on skill discovery, EvoControl targets learning a fast low-level policy that complements the slow high-level policy. Uniquely, EvoControl combines PPO and Neuroevolution within its hierarchical framework for efficient exploration and complex high-frequency control.

Hybrid Combinations of RL and ES: Existing related work has looked into combining evolutionary strategies ES to improve RL algorithms, specifically using them to collect diverse data as ES methods show superior exploration capabilities compared to on-policy and off-policy RL algorithms, and also take updates for the RL agent itself [59, 60, 61, 62, 63, 64]. Specifically, Suri et al. [60], Khadka and Tumer [61] use RL (SAC/DDPG) with ES data collection to collect diverse trajectories into the replay buffer to update the RL agent. Suri et al. [60] propose automatic mutation tuning to improve the ES component, and demonstrate improved performance on 10 out of 15 continuous-control environments compared to the equivalent RL method baselines. Khadka and Tumer [61] also uses Neuroevolution (ES) to collect diversified trajectories, and use these trajectories in a replay buffer to train an off-policy RL agent. They further, update the Neuroevolution data collection agent with snapshots of the trained RL agent throughout training, and demonstrate on continuous control tasks that this can lead to higher reward evaluation and faster convergence in higher-dimensional state-action challenging environments. Furthermore, Zheng et al. [65] proposes a transfer approach that has a pool of agents containing three classes of agents: on-policy agents, off-policy agents, and a population-based ES agents. All agents explore and collect trajectories into a replay buffer, with the on-policy and ES agent initially transferred from the weights off-policy (global) agent; the trajectories of the on-policy agents are then more frequently sampled when used to update the off-policy global agent, and use a

threshold to control the frequency of policy parameter transfer. There also exist alternative solutions to combine ES and RL, such as seeding ES with an RL agent for symbolic regression [66]. Moreover, Elfwing et al. [67] proposed a task decomposition method, using MAXQ, to break down a complex task into a hierarchy of subtasks on small dimensional state-action space problems, and used a genetic programming algorithm to learn the hierarchical task decomposition automatically. Specifically in Elfwing et al. [67], each hierarchy corresponds to a different subtask that can be performed. Unlike EvoControl, all of these related works do not consider the problem of learning at higher frequencies, operating their continuous control environments at default large discrete time steps (e.g. 20-100Hz [46]), and do not focus on an hierarchical approach of having a high-level policy outputting a latent action and a low-level policy following this latent action for G steps (providing temporal action abstractions)—which limits the practical deployment of their agent, as if rich features are used as inputs to the agent such as images or the use of larger architectures, such as Vision Language Transformers, the inference time of the agent would increase (e.g. $\approx 30Hz$ for images from video), limiting the agents use where high-frequency control is necessary for an environment. Conversely, EvoControl enabled from its hierarchical structure decomposing a high-level policy and a low-level policy, the low-level policy can remain a simple neural network agent of a smaller size being able to run with a fast inference time, and hence fast control operation of $500 - 1KHz$, and still gain the benefit of having a higher-level policy that can still take as input rich features arriving at a lower-frequency such as images.

E Environment Selection and Implementation Details

Benchmark environments. We compare against ten standard continuous-control environments [46, 47], and also a safety critical continuous control environment. Specifically we use the continuous control suite from Brax[7] [47], which consists of ten standard continuous control environments, such as locomotion based robot control tasks such as Ant, HalfCheetah and larger state-action space environments such as Humanoid (e.g. controlling a humanoid robot with a state-action dimension of 60 to walk forwards with a given velocity). All the Brax environments are released under the Apache-2.0 license. Furthermore, within this standard suite of tasks is manipulation based environments such as Reacher and Pusher, where pusher is a 7 degree of freedom (DOF) robotic arm, with the task to push a movable object on a table to a desired goal location. Moreover, we also construct a safety inspired environment, adapting a single arm version of the Reacher environment, where introduce a random un-modeled contact that incurs a large negative reward when the robot manipulator collides with the object. To compare the frequency element, we set the frequency of each environment to 500Hz, and then motivated by a low-lever controller running at lower frequency such as 31.25Hz ($G = 16$) (a realistic assumption when involving cameras to determine state), we set this as the low-level frequency.

E.1 Standard Gym MuJuCo Tasks

We use Brax [47], a differentiable physics engine, which provides efficient implementations of the Ant, HalfCheetah, Hopper, Humanoid, HumanoidStandup, InvertedDoublePendulum, Pusher, Reacher, and Walker2d environments. These environments encompass a range of locomotion and manipulation tasks, providing a diverse testbed for evaluating EvoControl. For each environment, we set the simulation timestep Δ_t to 0.002 (500Hz operation). High-level policies operate at a frequency of 31.25Hz, achieved by executing each high-level action for $G = 16$ simulation steps. To ensure a fair comparison across different control modes, we remove the action magnitude penalization from the default reward function of each environment. The low-level policy receives the high-level action concatenated to a subset of the environment observation state as its own observation, and the exact input specification for each EvoControl variation is provided in Table 2. This allows the low-level controller to condition its actions on the target specified by the high-level policy. The low-level

[7] The Brax continuous control environments are all publicly available from https://github.com/google/brax.

action space is the same as the high-level action space. All environments have a fixed episode length of low-level timesteps of 1,000 environment steps. To increase the realism of the simulation, we run the Brax environments with the backend of MJX, that is a MuJoCo environment in Jax with XLA. This enables us to even modify the MuJoCo xml definition file (to create the Safety Critical Reacher) environment. For all MuJoCo environments, we incorporated fixed PD controllers. We tuned the PD gains for each environment individually. Specifically, we set the proportional gain (K_p) to 1.0. This value was chosen as the environments, by default, accept actions with a magnitude of 1, representing a normalized torque input. To determine the optimal derivative gain (K_d), we leveraged MuJoCo's dampratio parameter, setting it to 1.0 (critically damped). We then empirically observed the K_d value that corresponds to this dampratio within the simulation. These tuned K_p and K_d values were used consistently throughout our experiments unless explicitly stated otherwise, providing a standardized and well-tuned PD baseline for comparison with EvoControl. This approach ensured that the PD controllers were appropriately configured for each environment's dynamics, providing a strong benchmark for evaluating the performance of learned low-level policies.

E.2 Reacher 1D

The Reacher 1D environment is a simplified version of the standard Reacher environment. We remove the second arm link, creating a 1DOF task suitable for detailed analysis. The goal is randomly placed within the reachable workspace of the single arm link. The high-level state space consists of the angle and angular velocity of the arm, and the 2D position of the target. The high-level action is the desired angle. The low-level state comprises the high-level state concatenated with the high-level action, and the low-level action is the torque applied to the joint. To ensure reproducibility we provide the full environment MuJoCo xml specification below.

```
<mujoco model="reacher_1d">
  <compiler angle="radian" inertiafromgeom="true"/>
  <default>
    <joint armature="1" damping="1.0" limited="true"/>
    <geom conaffinity="0" contype="0" friction="1 0.1 0.1" rgba="0.4 0.33 0.26 1.0"/>
  </default>
  <option gravity="0 0 0" timestep="0.002" />
  <custom>
    <!-- brax custom params -->
    <numeric data="0 0.1 -0.1" name="init_qpos"/>
    <numeric data="1000 1000" name="constraint_stiffness"/>
    <numeric data="1000" name="constraint_limit_stiffness"/>
    <numeric data="3 0.1" name="constraint_vel_damping"/>
    <numeric data="0.1" name="constraint_ang_damping"/>
    <numeric data="0.0" name="ang_damping"/>
    <numeric data="0" name="spring_mass_scale"/>
    <numeric data="1" name="spring_inertia_scale"/>
    <numeric data="5" name="solver_maxls"/>
  </custom>
  <worldbody>
    <light diffuse=".5 .5 .5" pos="0 0 3" dir="0 0 -1"/>
    <!-- Arena -->
    <geom conaffinity="0" contype="0" name="ground" pos="0 0 0" size="1 1 10" type="plane" rgba="1 1 1 1"/>
    <geom conaffinity="0" fromto="-.3 -.3 .01 .3 -.3 .01" name="sideS" size=".02" type="capsule"/>
    <geom conaffinity="0" fromto=" .3 -.3 .01 .3  .3 .01" name="sideE" size=".02" type="capsule"/>
    <geom conaffinity="0" fromto="-.3  .3 .01 .3  .3 .01" name="sideN" size=".02" type="capsule"/>
    <geom conaffinity="0" fromto="-.3 -.3 .01 -.3 .3 .01" name="sideW" size=".02" type="capsule"/>
    <!-- Arm -->
    <geom conaffinity="0" contype="0" fromto="0 0 0 0 0 0.02" name="root" size=".011" type="capsule"/>
      <body name="body0" pos="0 0 0.01">
        <joint axis="0 0 1" limited="true" name="joint0" pos="0 0 0" type="hinge" range="-3.13 3.13"/>
        <geom fromto="0 0 0 0.2 0 0" name="link0" size=".01" type="capsule"/>
        <body name="fingertip" pos="0.11 0 0">
          <geom name="fingertip" pos="0 0 0" size=".01" type="sphere"/>
        </body>
      </body>
    <!-- Target -->
    <body name="target" pos="0 0 0.01">
      <joint armature="0" axis="1 0 0" damping="0" limited="true" name="target_x" pos="0 0 0" range="-.2 .2" stiffness="0" type="
          slide"/>
      <joint armature="0" axis="0 1 0" damping="0" limited="true" name="target_y" pos="0 0 0" range="-.2 .2" stiffness="0" type="
          slide"/>
      <geom conaffinity="0" contype="0" name="target" pos="0 0 0" size=".009" type="sphere"/>
    </body>
  </worldbody>
  <actuator>
    <motor ctrllimited="true" ctrlrange="-1.0 1.0" gear="200.0" joint="joint0"/>
  </actuator>
</mujoco>
```

E.3 Safety Critical Reacher

The Safety Critical Reacher environment builds upon the Reacher 1D environment by introducing a safety aspect. In 25% of the episodes, a randomly positioned obstacle is introduced, which the arm

must avoid. A contact force sensor is added to the observations, and a penalty is applied to the reward for any contact force exceeding a threshold. This encourages the development of low-level controllers capable of reacting quickly to avoid collisions. The high-level state space adds a contact force sensor to the Reacher 1D state, while action spaces for both high and low level controllers remain the same as the Reacher 1D environment. This environment directly tests the hypothesis that higher-frequency actions can lead to significantly better performance in safety-critical scenarios, aligning with the intuition presented in Proposition 2.1. The faster reaction time allowed by a high-frequency low-level controller is crucial for effective collision avoidance. To ensure reproducibility we provide the full environment MuJoCo xml specification below. We also use the following reward:

$$r = -\left\|q_0 - \frac{\pi}{2}\right\| - 3.1415927 \cdot \mathbb{I}(\|f_c\| > 0) \tag{20}$$

where q_0 is the joint angle (where q_0 indicates the first dimension of q at time t), f_c is the contact force between the arm and the obstacle, and $\mathbb{I}(\cdot)$ is an indicator function that equals 1 if the condition inside is true, and 0 otherwise. Where we used a fixed goal location of $\pi/2$, and initial starting state of $q_0 = 0$.

```
<mujoco model="safety_critical_reacher">
  <compiler angle="radian" inertiafromgeom="true"/>
  <default>
    <joint armature="1" damping="1" limited="true"/>
    <geom friction="1 0.1 0.1" rgba="0.4 0.33 0.26 1.0"/>
  </default>
  <option gravity="0 0 0" timestep="0.002" />
  <custom>
    <!-- brax custom params -->
    <numeric data="-1.57 0.11 0.0 -0.3" name="init_qpos"/>
    <numeric data="1000 1000" name="constraint_stiffness"/>
    <numeric data="1000" name="constraint_limit_stiffness"/>
    <numeric data="3 3" name="constraint_vel_damping"/>
    <numeric data="0.1" name="constraint_ang_damping"/>
    <numeric data="0.0" name="ang_damping"/>
    <numeric data="0" name="spring_mass_scale"/>
    <numeric data="1" name="spring_inertia_scale"/>
    <numeric data="5" name="solver_maxls"/>
  </custom>
  <worldbody>
    <light diffuse=".5 .5 .5" pos="0 0 3" dir="0 0 -1"/>
    <!-- Arena -->
    <geom conaffinity="0" contype="0" name="ground" pos="0 0 0" size="1 1 10" type="plane" rgba="1 1 1 1"/>
    <geom conaffinity="0" contype="0" fromto="-.3 -.3 .01 .3 -.3 .01" name="sideS" size=".02" type="capsule"/>
    <geom conaffinity="0" contype="0" fromto=" .3 -.3 .01 .3  .3 .01" name="sideE" size=".02" type="capsule"/>
    <geom conaffinity="0" contype="0" fromto="-.3  .3 .01 .3  .3 .01" name="sideN" size=".02" type="capsule"/>
    <geom conaffinity="0" contype="0" fromto="-.3 -.3 .01 -.3 .3 .01" name="sideW" size=".02" type="capsule"/>
    <!-- Arm -->
    <geom conaffinity="0" contype="0" fromto="0 0 0 0 0 0.02" name="root" size=".011" type="capsule"/>
      <body name="body0" pos="0 0 0.01">
        <joint axis="0 0 1" limited="true" name="joint0" pos="0 0 0" range="-1.570 3.1415" type="hinge"/>
        <geom fromto="0 0 0 0.2 0 0" name="link0" size=".01" type="capsule"/>
        <body name="fingertip" pos="0.11 0 0">
          <geom conaffinity="0" contype="0" name="fingertip" pos="0 0 0" size=".01" type="sphere"/>
        </body>
      </body>
    <!-- Random Collision Capsule -->
    <body name="obstacle-body" pos="0 0 0.01">
      <joint axis="1 0 0" damping="0" limited="true" name="obstacle_x" pos="0 0 0" range="-.2 .2" stiffness="0" type="slide"
            armature="1e10"/>
      <joint axis="0 1 0" damping="0" limited="true" name="obstacle_y" pos="0 0 0" range="-.2 .2" stiffness="0" type="slide"
            armature="1e10"/>
      <joint axis="0 0 1" damping="0" limited="true" name="obstacle_z" pos="0 0 0" range="-.3 .3" stiffness="0" type="slide"
            armature="1e10"/>
      <geom pos="0 0 0" size=".02" fromto="0 0 -0.1 0 0 0.1" type="capsule" name="obstacle"/>
    </body>
  </worldbody>
  <contact>
    <pair geom1="obstacle" geom2="fingertip" condim="1" />
    <pair geom1="obstacle" geom2="link0" condim="1" />
 </contact>
  <actuator>
    <motor ctrllimited="true" ctrlrange="-1.0 1.0" gear="200.0" joint="joint0"/>
  </actuator>
</mujoco>"""
```

F Benchmark Method Implementation Details

Benchmark methods. We seek to compare against competitive established baselines, using the same high-level PPO policy (ρ) learning algorithm with the same high-level architecture across all baselines, varying only the low-level policy (β). We compare with *fixed controllers* baselines, which are deterministic PD controllers of: PD Position, PD Position Delta, and PD Integrated Velocity [8]. We also compare against *direct torque control* baselines at both high (500Hz, i.e. the simulation timestep) and low (31.25Hz) frequencies; and a Random policy (31.25Hz). Moreover, we seek to investigate the EvoControl framework, and hence benchmark against several different variations from

varying the observation for the low-level policy, from the full-state to a restricted partially observed state (only observing the robot joint positions q or velocities $\dot{q}$), following EvoControl types with their corresponding observations as outlined in Table 2. Additionally, the EvoControl variants using position-based controllers are annealed from their corresponding PD controllers. We provide more detailed implementation information for each benchmark method in the following.

F.1 High-Level Policy and PPO Implementation

The focus of the paper is on enabling high-frequency control with a learning based method, therefore to provide a thorough competitive implementation of all the benchmark methods we use the same high-level policy neural network architecture and learning algorithm of PPO [27] across all the benchmark methods for all the main results. We did also perform additional ablations of training the high-level policy with Neuroevolution instead for all the benchmark methods, which can be seen in Appendix J.8.

We use the standard PPO implementation [27]. We used the fixed PPO hyper-parameters from PureJaxRL, which are derived from the PPO continuous-control environment parameters from CleanRL [68] which are themselves derived from those from stable baselines [69]. These hyper-parameters have been determined to provide good performance across a range of continuous-control environments. These parameters are specifically 'learning_rate'=3e-4,'num_envs'=1024,'num_steps'=10 (number of environment steps per rollout),'total_timesteps'=1e6, 'update_epochs'=4 (number of PPO update epochs per iteration), 'num_minibatches'=8 (number of minibatches for each PPO update),'gamma'=0.99 (discount factor),'gae_lambda'=0.95 (generalized Advantage Estimation parameter),'clip_eps'=0.2,'ent_coef'=0.0,'vf_coef'=0.5, and 'max_grad_norm'=0.5 (gradient clipping threshold).

The PPO implementation uses batched environments for efficient data collection, accumulating 'num_envs' $\times$ 'num_steps' transitions before performing updates. This facilitates parallel environment interaction and accelerated training.

The high-level policy architecture (the same for all benchmark methods) ρ_θ (with parameters θ) is represented by an actor-critic network implemented using Flax (a Jax based neural network library). Both the actor and critic share a common base network consisting of two hidden layers with 256 units each and tanh activation's (this architecture was initially provided by PureJaxRL to provide effective performance). The actor head outputs the parameters of a multivariate Gaussian distribution (mean and diagonal covariance)—as outlined in Section 3. The critic head outputs a scalar value estimating the state-value function.

F.2 PD Controller Implementation

We implement standard PD controllers as described in Table 1. For all environments, we tune the PD gains as described in Section E. Briefly, K_p is set to 1.0 and K_d is selected to correspond to a MuJoCo 'dampratio' of 1.0 (critically damped). For the PD Position controller, the high-level action a_k is interpreted as the desired absolute joint position (q^d). The PD Delta Position controller interprets a_k as a change in joint position (δq^d) relative to the joint position at the time of the high-level action q_k, such that $q^d = q_k + \delta q^d$. The Integrated Velocity controller interprets a_k as the desired joint velocity and integrates it to obtain a target position. This integration is performed numerically using the trapezoidal rule. These controllers provide a variety of baseline behaviors for comparison.

F.3 Fixed Controllers

The fixed controllers (PD Position, PD Position Delta, and PD Integrated Velocity) are implemented as deterministic policies. Given a state and the high-level action a_k, they directly compute the low-level control action u_t based on the corresponding control law as described in Table 1.

F.4 Direct Torque Control

For direct torque control, we use two variants: high-frequency (500Hz) and low-frequency (31.25Hz). In the high-frequency variant, the policy operates at the simulation frequency, outputting a torque command at every simulation step. The low-frequency variant operates at the same frequency as the high-level policy in the hierarchical setting. It outputs a torque command every $G = 16$ simulation steps, which is held constant during the intervening steps. Both variants are trained using PPO with the same hyperparameters as the high-level policy, except for the number of environment steps which is adapted based on the direct torque control policy frequency. This allows for a direct comparison of the performance of direct torque control at different frequencies. The same high-level PPO implementation is used to train both the high and low-frequency policies, ensuring that any performance differences are due to the control frequency and not the learning algorithm itself.

G EvoControl Implementation Details

In the following we provide implementation details for EvoControl. We used JAX [70] to implement EvoControl, and present the core training loop in Algorithm 2.

Network Architectures. We use the exact same high-level policy architecture and learning algorithm as the baselines use, from Appendix F.1. Therefore the high-level policy ρ_θ (with parameters θ) is represented by an actor-critic network implemented using Flax. Both the actor and critic share a common base network consisting of two hidden layers with 256 units each and tanh activations. The actor head outputs the parameters of a multivariate Gaussian distribution (mean and diagonal covariance). The critic head outputs a scalar value estimating the state-value function. The low-level policy $\beta_{\phi NN}$ (with parameters ϕ) is a separate neural network, also implemented using Flax. It consists of three hidden layers with 256 units each and tanh activations. The output layer produces the low-level control actions (torques), $\tau = u_t$ (unless otherwise specified, for example K_p, K_d). Specifically the low-level takes as an input observation the EvoControl variant observation, as detailed in Table 2.

Table 7: EvoControl Ablation of PD Controllers.

Controller Variant	β_{NN} Obs.	β_{NN} action
EvoControl (Full State)	$s_t, a_k, e_t, q_t, \dot{q}_t, t/T$	τ
EvoControl (Residual State)	$e_t, t/T$	τ
EvoControl (Target + Proprioceptive)	$a_k, q_t, \dot{q}_t, e_t, t/T$	τ
EvoControl (Target)	$a_k, q_t, \dot{q}_t, t/T$	τ
EvoControl (Learned Gains)	$s_t, a_k, q_t, \dot{q}_t, t/T$	K_p, K_d
EvoControl (Delta Position)	$s_t, a_k, e_t, q_t, \dot{q}_t, t/T$	τ

Low-level Observation. For clarity we reproduce Table 2, here as Table 7. Specifically, the observation for the low-level policy can consist of the current state s_t, the high-level policy latent action a_k, the PD controller error e_t (that is used during the annealing), the robots generalized positions q_t, the generalized velocities $\dot{q}_t$, and the ratio of the percentage of the low-level steps that the current high-level action is being followed for—for example with $G = 16$, $T = G = 16$, and hence $t = i$ (Algorithm 1) or the number of low-level steps out of G that the low-level policy is currently on whilst following the high-level policy.

Annealing Strategy. The annealing parameter α controls the convex combination of the fixed PD controller (β_{PD}) and the learned low-level policy ($\beta_{\phi NN}$). We use a linear decay schedule, starting at $\alpha = 1.0$ and decreasing linearly to $\alpha = 0.0$ over the K training sections, of $\alpha_k = 1 - \frac{k}{K}$, where k is the current training section.

Neuroevolution Details. For Neuroevolution we use Policy Gradients with Parameter-Based Exploration (PGPE) [31] algorithm to optimize the low-level policy $\beta_{\phi NN}$. The neural network's parameter vector ϕ is directly optimized. We use a population size of es_pop_size $= 512$, and each individual is evaluated over es_rollouts $= 16$ rollouts to estimate its fitness (episodic return R). Adam [71] is used within PGPE, and we we use the PGPE hyper-parameters of a center learning rate of 0.05 and a standard deviation learning rate of 0.1. We use es_sub_generations $= 8$ generations per training section k. The parameter distribution's initial standard deviation is 0.1. We use the implementation of PGPE provided by EvoJax [72], in Jax, and their recommended hyper-parameters for PGPE, which were empirically found to work well for continuous control tasks. Furthermore, we set $K = 8$ per 1M high-level ρ steps used to train the high-level policy for, and this was empirically determined to work well in practice.

G.1 EvoControl PseudoCode

For the following pseudocode; we used the same parameters as described above, specifically setting the total number of training sections to $K = 8$ (per 1M high-level ρ policy steps), and then training the

PPO high-level policy for 1M (i.e. 1 Million) high-level steps, therefore performing N PPO updates of $N = \lfloor$ 1e6/(num_envs$\times$num_steps$\times$K)$\rfloor$ = 12. We note that each step of the slow high-level policy when operating with a fast low-level policy is effectively $G = 16$ high-frequency environment timesteps of Δ_t.

Algorithm 2 EvoControl Training

Require: Environment $f(s_t, u_t)$, reward function $r(s_t, u_t)$, high-level policy $\rho_\theta(s_k)$, initial low-level policy $\beta_{PD}(s_i, a_k)$, total training sections K, steps per section N, annealing strategy for α, neuroevolution parameters η, population size P, generations per section G_{evo}, rollouts per individual R_{evo}.
Ensure: Trained high-level policy $\rho_\theta(s_k)$, trained low-level policy $\beta_\phi(s_i, a_k)$.

Initialize $\alpha \leftarrow 1.0$
Initialize low-level policy $\beta(s_i, a_k) \leftarrow \alpha\beta_{\text{PD}}(s_i, a_k) + (1-\alpha)\beta_{\phi NN}(s_i, a_k)$
Initialize neuroevolution strategy (e.g., PGPE) with parameters η
for $k = 1$ to K **do**
    // Train high-level policy ρ_θ with PPO
    for $n = 1$ to N **do**
        Collect rollout data using ρ_θ and β (Algorithm 1)
        Update ρ_θ using PPO
    // Train low-level policy β_ϕ with neuroevolution
    for $g = 1$ to G_{evo} **do**
        $\phi_{\text{pop}} \leftarrow$ Sample P parameter sets from $p_\eta(\phi)$
        for $p = 1$ to P **do**
            $F_p \leftarrow 0$
            for $r = 1$ to R_{evo} **do**
                Collect rollout using ρ_θ (mode) and β_{ϕ_p} (Algorithm 1)
                $F_p \leftarrow F_p +$ rollout return
            $F_p \leftarrow F_p / R_{evo}$
        Update neuroevolution parameters η using fitness values $F_{1:P}$ (e.g., PGPE update)
    $\phi \leftarrow$ best performing parameter set from neuroevolution
    $\beta_{\phi NN}(s_i, a_k) \leftarrow$ neural network with parameters ϕ
    $\beta(s_i, a_k) \leftarrow \alpha\beta_{\text{PD}}(s_i, a_k) + (1-\alpha)\beta_{\phi NN}(s_i, a_k)$
    $\alpha \leftarrow 1 - k/K$

G.2 Detailed Analysis of EvoControl

This appendix provides a detailed analysis of the EvoControl algorithm, addressing the mathematical setting, assumptions, complexity, and properties as requested.

G.2.1 Mathematical Setting and Assumptions

EvoControl operates within the standard continuous control Reinforcement Learning (RL) framework. We consider a Markov Decision Process (MDP) defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{U}, P, r, \gamma \rangle$, with definitions provided in Appendix A.

Assumptions:

- *Markov Property:* The environment dynamics satisfy the Markov property, meaning the next state depends only on the current state and action, not on the history.
- *Stationarity:* The transition probabilities and reward function are stationary (do not change over time).
- *Differentiable Policy:* The high-level policy $\rho_\theta(s_k)$ is parameterized by θ and is differentiable with respect to θ. This allows for gradient-based optimization.
- *Representable Low-Level Policy:* The low-level policy $\beta_\phi(s_i, a_k)$ can be adequately represented by the chosen neural network architecture with parameters ϕ.

G.2.2 Complexity Analysis

Time Complexity: The time complexity of EvoControl is dominated by the PPO updates for the high-level policy (ρ) and the rollout evaluations for Neuroevolution of the low-level policy (β).

PPO Updates: The per-update complexity of PPO scales linearly with the number of environment interactions. For N_ρ high-level steps, with 'num_envs' parallel environments running for 'num_steps' steps each, and PPO updates occurring every K training sections, there are $N_\rho/(K \cdot \text{num\_envs} \cdot \text{num\_steps})$ PPO updates. Each PPO high-level environment step involves G low-level environment steps.

Neuroevolution Rollouts: Each training section involves Neuroevolution of the low-level policy. With a population size of 'es_pop_size', 'es_rollouts' rollouts per individual, and 'es_sub_generations' generations per section, the number of rollouts per section is es_rollouts · es_sub_generations · es_pop_size. Each rollout has 'episode_length' low-level steps.

Let $\mathcal{T}_{\text{env}}$ be the time for a single low-level environment step of duration Δ_t. Then, the total time complexity, *without* parallelization of environment rollouts, is:

$$O\left((N_\rho \cdot G + K \cdot \text{es\_rollouts} \cdot \text{es\_sub\_generations} \cdot \text{es\_pop\_size} \cdot \text{episode\_length}) \cdot \mathcal{T}_{\text{env}}\right)$$

We can also re-express this, as if we train the high-level policy for N_ρ high-level ρ steps, and we train EvoControl with $K = 8$ sections per 1M high-level ρ steps (i.e. $K = (N_\rho \cdot 8)/(1e6)$), then the time complexity can also be expressed as:

$$O((N_\rho \cdot G + \frac{N_\rho \cdot 8}{1e6} \cdot \text{es\_rollouts} \cdot \text{es\_sub\_generations} \cdot \text{es\_pop\_size} \cdot \text{episode\_length}) \cdot \mathcal{T}_{\text{env}})$$

However, both of these time complexity measures are worst case, and do not account for any availability to parallelize environment rollouts, which is common in practice on modern GPUs. If we assume that a user has a GPU/CPU that can parallelize the environment rollouts, then the time complexity can approach:

$$O((N_m \cdot \text{num\_steps} \cdot G + \frac{N_\rho \cdot 8}{1e6} \cdot \text{es\_sub\_generations} \cdot \text{episode\_length}) \cdot \mathcal{T}_{\text{env}})$$

Where $N_m = (N_\rho)/(\text{num\_envs} \cdot \text{num\_steps})$.

We provide thorough additional experiments limiting the computational complexity of the above two approaches, as detailed in Appendix J.3.

Space Complexity: The space complexity is primarily determined by the size of the neural networks for the high-level and low-level policies, the size of the PPO buffer, and the population size for neuroevolution. It is $O(|\theta| + |\phi| + S_{PPO} + P \cdot |\phi|)$, where $|\theta|$ and $|\phi|$ are the number of parameters in the high-level and low-level policies respectively, and S_{PPO} is the size of the PPO buffer.

G.3 Computational Considerations

Building on the previous section, environment rollouts can be parallelized on modern GPUs. Specifically num_envs, es_rollouts, es_pop_size can all be parallelized. A benefit of neuroevolution here, is that the fitness evaluations (within PGPE) are highly parallelizable. We leverage JAX's 'vmap' function for vectorized rollouts, enabling efficient parallel execution on GPUs or CPUs. This can also be readily further optimized, such as distributing the population across multiple devices, to reduce training time (as neuroevolution is a gradient free approach) [30].

Whilst the goal of our work is to provide an initial method that can learn a better low-level controller for use within a high-level policy learning environment, to achieve higher final evaluation reward, we

acknowledge that doing so increases computational complexity compared to policy learning with a traditional fixed PD controller. Therefore to investigate, what happens if we make the number of low-level environment steps equivalent we provide a further ablation in this setting in Appendix J.3.

H Evaluation Metrics

For each environment, and for each baseline we train the joint policy π consisting of a high-level ρ and a low-level policy β for the same number of high-level policy ρ steps, here 1M steps. Once the policies have been trained, we perform 128 evaluation rollouts, each with a different random seed and compute the undiscounted cumulative sum of rewards for each rollout, i.e. the return for the episode, where each episode lasts 1,000 environment steps. We repeat training each baseline policy across three random seeds. We quote each result as the mean across it's random seeds and provide the corresponding 95% confidence intervals throughout for all metrics. Specifically we quote the normalized score $\mathcal{R}$ [50] of the policy in the environment, averaged over 384 random seeds—normalized to the interval of 0 to 100, where a score of 0 corresponds to a random policy performance, and 100 to an existing fixed controller expert policy—which is whichever non-EvoControl baseline scores the highest evaluation environment return.

All experiments were run on a NVIDIA H100 GPU, with 80GB VRAM with a 40 core CPU with 256 RAM. We detail the hyper-parameters in Appendix F for each benchmark method, and how the hyper-parameters were selected, and their origin of source. We did sweep over the learning rate for PPO with the fixed controller PD position baseline, however found the initial hyper-parameters already provided by prior work (PureJaxRL [73], and hence CleanRL [68]) to be the most performant, therefore they were kept constant throughout all experiments.

I Additional Experimental Setup

I.1 Efficient Exploration Experimental Setup

To reproduce this experiment, we used the Reacher 1D environment, as detailed in Appendix E.2. Specifically to investigate the efficiency of exploration, we modified the Reacher 1D environment to have a deterministic goal across new random seeds, such that the goal location is $q_{\text{goal}} = \pi/2.0$, and the initial starting position to $q = 0$. This is to ensure that we can correctly measure exploration, otherwise starting in a random state with a random goal, could already explore the state-action space, just through environment resets—whereas the focus of this insight experiment is to compare the methods exploration instead, hence fixing the environment starting state and end goal state. Specifically we run each baseline approach for 10,240 low-level environment steps each, and collect the state throughout training for these initial steps. We then process the state collected, and plot the state visitation histograms, as shown in Figure 2.

I.2 High-frequency Interaction Control in Safety Critical Reacher

To reproduce this experiment, we follow the same setup for the Safety Critical Reacher environment, as detailed in Appendix E.3.

J Additional Experiments

J.1 Ablation Using PPO to Train the Lower-level Policy

We performed an additional ablation experiment, by training the low-level policy with PPO rather than Neuroevolution. To be comparable we used the same architecture that our existing high-level PPO agent uses, as described in Section 3, and Appendix G. We follow the same setup, of training the high-level policy for 1M high-level environment steps, and now train the low-level policy for the same 1M high-level steps, now training for the low-level for 1M$\times G$ =16M low-level environment

steps—to give this ablation the most competitive performance comparison to EvoControl and the non-EvoControl baselines. We perform a complete re-run across all environments as presented in the main paper main results table. The ablation with PPO training the lower-level policy can be seen in Table 8. We observe that using PPO to train the lower-level policy within this EvoControl ablation performs worse (achieves a lower average evaluation return) than using Neuroevolution to train the lower-level policy β—thus justifying the use of Neurevolution for training the low-level policy.

Table 8: Ablation. Training the lower-level policy with PPO instead of Neuroevolution, training both the high-level policy and the low-level policy for 1M high-level environment steps each, to produce a competitive ablation. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Method Name	High-level ρ with	Low-level β with	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	PPO	PD Position	100±6.56	61.2±0.441	91.6±1.23	100±2.96	100±0.974	99.9±0.03	100±1.53e-06	100±8.47	100±1.8	85.2±2.87	75.7±0.633
Fixed Cont. - PD Position Delta	PPO	PD Position Delta	2.4±1.91	2.76±0.0888	100±1.35	96.6±1.71	2.96±0.0397	53.8±1.57	100±1.53e-06	0±0	40.9±3.23	15.2±7.6	90.2±0.239
Fixed Cont. - PD Int. Velocity	PPO	PD Int. Velocity	3.59±1.78	2.46±0.0932	74.7±0.903	83.4±1.13	0±0	49.7±1.55	86.5±2	0±0	0±0	0±0	85.9±2.55
Fixed Cont. - Random	Random	Direct Torque	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	PPO	Direct Torque	0±0	17.2±0.316	1.42±0.533	10.4±2.19	10.3±0.586	0±0	0±0	1.34±7.89	2.08±5.84	45.3±6.74	0±0
Direct Torque Cont. - Low Freq. (31.25Hz)	PPO	Direct Torque	54.5±7.15	100±1.21	72±0.64	98±2.55	80.6±2.56	100±0.0311	100±1.53e-06	73.2±12.9	59.2±3.72	100±1.94	100±2.68
Ablation: EvoControl (Full State)	PPO	PPO	16.4±59.6	25.5±53.6	**102±55.7**	**142±20.3**	82±78.8	83.8±40	69.4±132	**105±209**	55±27.1	80.1±31.5	82.4±9.64
Ablation: EvoControl (Residual State)	PPO	PPO	12±49.7	41.3±69.2	68.4±102	60.2±121	33.9±100	84.6±64.8	100±0	94.7±351	91.4±12.9	**101±7.8**	**103±80.2**
Ablation: EvoControl (Target + Proprio.)	PPO	PPO	0±0	25.5±34.6	54.1±72.9	56.6±34.2	38.5±52.6	34.6±138	29.8±22.3	91.3±15.3	44.7±44.8	62.3±85.1	20.8±119
Ablation: EvoControl (Target)	PPO	PPO	0±0	17.8±16.7	88.2±3.09	69.8±76.3	45±40.6	13.5±119	27.8±121	**132±146**	59.3±35.6	15.5±33.5	95.1±43.2
Ablation: EvoControl (Delta Position)	PPO	PPO	0±0	2.88±1.86	89±16.9	92.5±10.2	51.2±58.1	98.9±2.64	81.2±81	0±0	43.9±10.7	5.07±10.6	61.6±48.6

J.2 ES Outperforms Direct Torque Control at High-frequency

In the following we provide empirical evidence for EvoControl outperforming the baseline of a high-frequency low-level direct torque control policy. To address any sample complexity concerns, we also find when we limit EvoControl to use the same computational complexity as all baselines, EvoControl still outperforms the baselines, which is evaluated in detail in Appendix J.3. To provide a thorough analysis of the ability to learn a high-frequency low-level direct torque control policy, we performed additional experiments of training the *Direct Torque Cont. - High Freq. (500Hz)* baseline for an increasing number of high-level ρ policy steps. Specifically, as tabulated in Table 9, we train for a larger number of ρ steps, significantly greater than all the baselines were trained for (which is 1M ρ steps)—here being from 1M ρ steps to 10B ρ steps. We observe that even with more high-level ρ steps, which corresponds to significantly more low-level environment steps than that used in EvoControl, *Direct Torque Cont. - High Freq. (500Hz)* still on average achieves a lower normalized return compared to EvoControl. This could suggest that direct high-frequency control with PPO produces policies that get stuck in local minima, and fail to find a better performing global policy at high-frequency as EvoControl is able to do—leveraging Neuroevolution for learning the lower-level high-frequency policy.

Table 9: Additional Experiment. Training *Direct Torque Cont. - High Freq. (500Hz)* baseline for an increased number of high-level ρ policy steps—from from 1M ρ steps to 10B ρ steps. Normalized evaluation return $\mathcal{R}$ for the baseline, across each environment. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment—using the normalization from the main table of results, Table 3.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
1,000,000 Train ρ steps *Direct Torque Cont.* - High Freq. (500Hz)	0±0	17.2±0.336	3.63±0.259	15.6±3.64	17±1.2	0±0	0±0	2.95±7.81	0±0	45.3±6.74	0±0
10,000,000 Train ρ steps *Direct Torque Cont.* - High Freq. (500Hz)	6.3±14.9	63±1.43	93.5±2.82	87.9±3.41	78.3±2	97.9±0.148	42.6±1.48	21.9±9.03	42.4±5.19	87.8±3.62	36.5±1.08
100,000,000 Train ρ steps *Direct Torque Cont.* - High Freq. (500Hz)	137±17	106±1.46	124±3.52	133±4.13	139±3.5	88.2±1.08	63.2±2.86	42.5±13.9	103±2.32	67.6±6.25	91.5±6.51
1,000,000,000 Train ρ steps *Direct Torque Cont.* - High Freq. (500Hz)	125±15.4	101±1.35	141±2.49	127±3.79	119±3.05	87.5±1	84.8±1.34	219±11.1	98.3±2.58	73.5±6.04	149±6.47
10,000,000,000 Train ρ steps *Direct Torque Cont.* - High Freq. (500Hz)	185±16.6	86.3±1.09	155±6.58	143±6.98	71.3±1.3	91.8±1.08	84.6±1.21	75.5±12.7	88.5±3.01	81.2±4.86	219±5.53

J.3 Ablation Equal Computational Complexity for All Baselines

We investigate making the computational complexity the same for all baselines and EvoControl, in two approaches. First, the most direct approach we set the budget of the number of low-level environment steps to be equivalent for all baselines, listed as *equivalent number of low-level environment steps* in Appendix J.3.1. Second, we recognize that modern GPUs allow for environment parallelization. Thus, we investigate only setting the same number of sequential low-level environment steps to be

equivalent for all baselines—where the bottleneck for parallelized rollouts is the number of sequential steps of the parallelized environments. This is listed as *equivalent number of sequential low-level environment steps* in Appendix J.3.2.

J.3.1 Equivalent Number of Low-Level Environment Steps

Here we explicitly set the total number of low-level environment steps for all the baselines to be the same. For EvoControl, that trains its high-level policy with PPO and its low-level policy with Neuroevolution, this means that the high-level policy trained with PPO now receives less high-level update steps compared to the baselines, to account for the low-level step budget used by the low-level Neuroevolution component. This is different from the main results within the paper (Section 5.1) which trained each baseline for 1M high-level steps, thus meaning that the high-level policy ρ was trained for 1M steps, not accounting for the potentially differing number of low-level steps used by updating or using the lower-level policy.

To set the total number of low-level environment steps for all the baselines to be the same, we first compute the total number of low-level steps that EvoControl uses, where we train EvoControl's high-level policy for 1M steps, and then now train the baseline methods for this increased number of equivalent high-level steps. Specifically if we consider a PD position baseline, originally training this for 1M high-level environment steps, with a lower-level PD position controller, operating at a higher frequency with $G = 16$, meaning that the number of low-level environment steps used in the environment are $1\text{M} \times 16 = 16M$. Here EvoControl, when the high-level is trained for 1M steps, the lower-level policy also receives updates, therefore the total number of low-level environment steps used within the training of EvoControl is $K \times$ es_rollouts $\times$ es_sub_generations $\times$ es_pop_size $\times$ episode_length. To simplify the comparison, we explicitly set es_rollouts $= 1$ and leave the other inputs the same as they were for the main results (that of $K = 8$, es_sub_generations $= 8$, episode_length $= 1000$). This leaves the input parameter of es_pop_size that we can vary. Therefore the total number of low-level environment steps used by EvoControl is $1\text{M} \times 16 + 64K \times$ es_pop_size. Therefore, for the following experiments we train all the other non-EvoControl baselines for $1\text{M} \times 16 + 64K \times$ es_pop_size low-level environment steps, by specifically determining how many high-level steps this is by dividing by G and using that as the input as the total number of high-level steps to train for each baseline.

For the results, as discussed, we vary es_pop_size $= \{16, 32, 64, 128, 256\}$ and re-run each non-EvoControl baseline with the equivalent number of low-level steps as EvoControl—which means as EvoControl uses Neuroevolution to update the low-level policy, the high-level policy now receives less equivalent updates compared to the high-level policy of the non-EvoControl baselines. We observe in Tables 10 to 14 that EvoControl even when limited to use the same number of low-level environment steps as all the baselines, on average achieves a higher normalized evaluation return than all the baselines *fixed controllers* and *direct torque control*.

Table 10: Ablation. Equivalent Number of Low-Level Environment Steps, with es_pop_size $= 16$. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 6400 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±1.75	62.1±0.115	97.5±0.246	100±0.629	100±0.144	100±0.00784	100±3.74e-07	100±2.91	100±0.337	86±0.672	83.1±0.0441
Fixed Cont. - PD Position Delta	4.11±0.503	2.84±0.022	100±0.679	96.7±0.391	2.73±0.0111	55.9±0.412	100±3.74e-07	33±3.84	42.2±0.705	18.6±1.74	93.8±0.052
Fixed Cont. - PD Int. Velocity	2.05±0.504	2.53±0.0229	76.1±0.319	86.5±0.315	0±0	48.1±0.388	86.6±0.496	0±0	0±0	0±0	73.1±0.633
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0	24.1±0.233	12.3±0.336	0±0	31.3±0.434	3.34±0.797	60.2±1.07	2.43±2.54	17.4±1.22	72.6±1.21	36.4±0.7
Direct Torque Cont. - Low Freq. (31.25Hz)	46.5±1.42	100±0.359	78.3±0.148	97.1±0.627	81.3±0.292	100±0.00823	99.7±0.0278	80.6±3.86	59.8±0.801	100±0.504	100±0.272
EvoControl (Full State)	**145±2.51**	89.7±0.661	**127±0.596**	96.2±0.651	88.9±0.111	**100±0.0083**	100±3.74e-07	**423±3.41**	**107±0.192**	**106±0.183**	**134±0.491**
EvoControl (Residual State)	**124±2.45**	**125±0.595**	83.5±0.719	**117±0.768**	**101±0.159**	99.6±0.0159	100±3.74e-07	**445±3.61**	**105±0.178**	**105±0.227**	**188±0.831**
EvoControl (Target + Proprio.)	**148±2.81**	99.2±0.719	**146±0.998**	**122±0.707**	99.1±0.452	99.7±0.0748	95.4±0.29	**468±3.31**	**107±0.177**	**106±0.18**	**132±0.498**
EvoControl (Target)	**141±2.79**	**106±0.993**	**135±0.621**	**124±0.713**	**100±0.374**	99.9±0.0148	100±3.74e-07	**420±3.17**	65.3±0.802	**103±0.29**	**137±0.697**
EvoControl (Learned Gains)	79.8±2.43	52.3±0.487	**109±1.1**	85.3±0.732	93.6±0.178	72.5±1.24	100±3.74e-07	**409±3.45**	99.5±0.421	97.9±0.449	**114±0.909**
EvoControl (Delta Position)	**139±2.81**	99.2±0.85	**141±0.382**	97.6±0.764	68.8±0.63	87.6±0.565	100±3.74e-07	**398±3.32**	63.2±0.802	**102±0.337**	**150±0.529**

Table 11: Ablation. Equivalent Number of Low-Level Environment Steps, with es_pop_size $= 32$. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 6400 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±1.23	59.6±0.195	99.5±0.0651	83.2±0.473	100±0.275	99.9±0.00874	100±3.74e-07	100±2.71	100±0.258	86.3±0.645	74.8±0.128
Fixed Cont. - PD Position Delta	3.42±0.447	2.72±0.0204	100±0.266	96.2±0.411	3.03±0.00787	56.6±0.395	100±3.74e-07	50.7±3.73	42.6±0.682	18.5±1.72	93.3±0.072
Fixed Cont. - PD Int. Velocity	4.62±0.453	2.45±0.0214	82.5±0.702	79.4±0.275	0±0	50.6±0.404	96.9±0.226	0±0	0±0	0±0	100±0.519
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0	44.3±0.269	33.8±0.449	27.9±0.605	55±0.664	34.7±0.47	92.7±0.287	3.92±2.35	18.7±1.21	77±1.13	47.5±0.404
Direct Torque Cont. - Low Freq. (31.25Hz)	42.9±1.15	100±0.257	99.4±0.446	100±0.449	96.3±0.133	100±0.0082	100±3.74e-07	91.8±3.56	62.6±0.749	100±0.475	84.1±0.306
EvoControl (Full State)	**158±2.4**	**143±0.502**	**142±0.195**	93±0.663	96.5±0.144	**100±0.00707**	100±3.74e-07	**411±2.87**	**107±0.163**	**106±0.17**	**149±0.484**
EvoControl (Residual State)	**158±2.18**	**112±0.433**	94.2±0.114	**112±0.68**	**123±0.683**	99.5±0.0196	100±3.74e-07	**424±3.31**	97.4±0.435	**105±0.209**	**200±0.515**
EvoControl (Target + Proprio.)	**152±2.39**	**120±0.848**	**127±0.427**	**123±0.614**	**116±0.405**	99.3±0.0701	100±3.74e-07	**444±3.27**	57.3±0.813	**106±0.165**	**145±0.792**
EvoControl (Target)	**154±2.33**	**103±0.521**	**141±0.396**	**122±0.579**	**117±0.609**	88.4±0.494	100±3.74e-07	**404±3.29**	96.4±0.309	**104±0.219**	**136±0.557**
EvoControl (Learned Gains)	**104±2.31**	67.1±0.518	**155±1.98**	96.9±0.657	**100±0.209**	96.2±0.142	100±3.74e-07	**360±2.9**	**102±0.281**	**102±0.31**	**161±0.705**
EvoControl (Delta Position)	**161±2.53**	**105±0.707**	**127±0.91**	**101±0.674**	74±0.479	**100±0.0103**	100±3.74e-07	**429±3.24**	59.5±0.774	**102±0.316**	**149±0.361**

Table 12: Ablation. Equivalent Number of Low-Level Environment Steps, with es_pop_size $= 64$. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 6400 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±5.17	66±0.555	90.2±0.489	92.8±2.9	87.8±1.3	99.6±0.0357	100±1.53e-06	100±9.4	100±1.24	88.7±2.6	68.7±0.285
Fixed Cont. - PD Position Delta	4.93±1.67	2.88±0.0891	84.3±0.854	100±1.79	3.16±0.024	57.1±1.6	100±1.53e-06	3.4±9.98	42.8±2.99	27.9±7.48	85.4±0.513
Fixed Cont. - PD Int. Velocity	5.94±1.8	2.59±0.0938	56.9±2.3	72.6±0.872	0±0	61.3±1.59	99±0.45	0±0	0±0	9.58±7.72	100±0.681
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0	44±0.794	42.9±0.53	63.6±2.37	53±2.15	97.8±0.387	49.9±2.36	9.18±7.69	10.8±5.42	81.1±4.24	32.2±1.27
Direct Torque Cont. - Low Freq. (31.25Hz)	70.4±6.52	100±1.22	100±1.26	87.2±1.35	100±0.204	100±0.0178	100±1.53e-06	83.1±10.2	67.2±3.05	100±1.79	69.2±1.38
EvoControl (Full State)	**188±12**	**165±1.15**	**118±22.4**	**111±15.9**	90.2±6.6	**100±0.306**	100±0	**296±42.6**	**109±1.86**	**104±2.14**	**127±126**
EvoControl (Residual State)	**152±117**	**166±16.9**	**125±118**	**108±43**	**112±14.2**	99.2±0.304	100±0	**302±18.1**	**102±1.08**	**104±2.07**	**169±59.6**
EvoControl (Target + Proprio.)	**183±69.2**	**135±11.1**	**130±55.2**	**126±12.1**	**124±58.8**	87.1±53.3	100±0	**300±42.6**	**107±2.77**	**104±2.23**	**130±12.7**
EvoControl (Target)	**201±27.5**	**164±10.6**	**131±93.5**	**125±4.52**	**109±12.2**	99.5±1.7	100±0	**286±27**	83.5±63.4	**103±3.2**	**122±38**
EvoControl (Learned Gains)	**103±23.1**	90±20.6	**141±845**	**111±28.9**	**102±1.5**	93.9±26	100±0	**263±46.8**	**104±11.4**	**102±4.85**	**149±35.2**
EvoControl (Delta Position)	**168±29.9**	**123±75.2**	95.9±40.8	**104±9.23**	94.2±8.68	**100±0.491**	100±0	**300±61.6**	59±5.93	99.9±7.07	**160±12.4**

Table 13: Ablation. Equivalent Number of Low-Level Environment Steps, with es_pop_size $= 128$. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±9.19	73.2±0.419	86±0.878	89.5±2.72	78.4±1.31	99.7±0.0353	100±1.53e-06	100±8.67	100±0.928	86±2.76	100±0.935
Fixed Cont. - PD Position Delta	3.33±2.59	3.12±0.101	48.8±1.06	89±1.69	3.22±0.0153	70.2±1.95	100±1.53e-06	10.2±9.63	43.6±2.74	19.9±7.52	72.7±1.51
Fixed Cont. - PD Int. Velocity	8.12±2.69	2.82±0.107	42.2±1.94	57.8±0.929	0±0	52.2±1.66	100±1.53e-06	0±0	0±0	0.651±7.83	85.3±0.32
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	4.16±15.4	66.5±2.22	46.6±1.49	54.8±1.15	75.7±2.04	93.4±0.598	49.6±2.28	14.2±5.96	33.5±5.21	83.8±4.27	41.1±1.75
Direct Torque Cont. - Low Freq. (31.25Hz)	53.9±7.67	100±1.74	100±3.31	100±1.87	100±0.397	100±0.029	100±1.53e-06	85.2±6.7	68.1±2.89	100±1.53	88.5±0.724
EvoControl (Full State)	**328±88.2**	**148±68.4**	**105±2.16**	87.1±8	97.1±13.4	**101±0.0993**	100±0	**252±16.8**	**106±0.66**	**105±2.23**	**127±57.7**
EvoControl (Residual State)	**197±40.1**	**191±33**	94.2±120	**120±20.9**	**121±42**	99.8±0.474	100±0	**239±18.6**	**102±4.38**	**104±6.29**	**138±127**
EvoControl (Target + Proprio.)	**274±75.2**	**202±14.7**	**121±51.5**	**127±19.1**	**152±89.5**	90.6±37.6	100±0	**253±44.6**	**107±2.27**	**105±2.33**	**135±23.9**
EvoControl (Target)	**313±6.88**	**204±2.9**	**113±47.8**	**122±10.3**	**135±99**	92.4±31.5	100±0	**245±22.2**	**101±2.17**	**104±1.27**	**128±53.9**
EvoControl (Learned Gains)	**242±90.9**	**116±1.47**	**142±94.7**	**104±18.6**	**104±2.93**	99.6±3.18	100±0	**236±21.6**	**105±9.62**	**103±5.11**	**134±55.8**
EvoControl (Delta Position)	**296±14.1**	**151±50**	61.5±287	98.8±4.74	93.3±24.5	**101±0.13**	100±0	**238±19.7**	60.2±24.1	95.6±19.8	**123±41.4**

Table 14: Ablation. Equivalent Number of Low-Level Environment Steps, with es_pop_size $= 256$. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±5.67	60±0.402	47.3±1.17	100±1.78	96.2±0.628	100±0.0379	100±1.53e-06	100±6.56	100±0.959	85.4±2.66	100±0.253
Fixed Cont. - PD Position Delta	3±1.86	2.71±0.0777	27.6±0.461	98.3±1.56	3.03±0.024	96.3±0.84	100±1.53e-06	8.47±8.21	49.4±2.42	21.9±7.31	57.2±2.07
Fixed Cont. - PD Int. Velocity	6.16±1.79	2.46±0.0836	24.5±0.943	57.3±1.29	0±0	22±0.756	100±1.53e-06	0±0	0±0	3.33±7.57	66.1±0.156
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	28.7±13.2	62.2±1.3	58.7±0.756	65.7±2.62	75.1±2.23	99.5±0.0862	60.9±3.02	19.3±5.42	45.7±4.95	83.4±3.99	43.4±1.66
Direct Torque Cont. - Low Freq. (31.25Hz)	52.4±5.84	100±0.992	100±0.664	94±1.3	100±0.435	100±0.0472	100±1.53e-06	89.8±3.93	81.2±2.41	100±1.34	79.1±0.946
EvoControl (Full State)	**302±40.8**	**144±17.5**	69.9±18.2	97.1±14.5	**109±19.4**	**101±0.524**	100±0	**206±15.8**	89±64.6	**103±2.1**	**123±3.79**
EvoControl (Residual State)	**138±28.7**	**162±22.8**	37±3.31	**124±49**	**129±37.4**	**101±0.561**	100±0	**215±21.9**	99.9±7.45	**103±2.89**	**128±94.1**
EvoControl (Target + Proprio.)	**281±39.5**	**130±7.09**	90.1±11.2	**125±42.8**	**141±70.8**	99.2±5.02	76.9±99.4	**205±11.8**	**104±3.39**	**103±2.14**	**118±30.7**
EvoControl (Target)	**241±42.1**	**150±19.8**	75.1±38.2	**124±21.5**	**118±8.06**	99.7±2.18	100±0	**198±21.5**	75.7±56.7	**102±2.93**	92.7±30.3
EvoControl (Learned Gains)	**197±86.2**	96.3±13.9	59.1±50.9	**111±21.7**	**107±1.18**	**101±0.202**	100±0	**190±45.5**	**104±0.793**	**102±2.65**	**135±40**
EvoControl (Delta Position)	**257±50.6**	**116±23.7**	**102±2.25**	98.4±14.1	98.5±11.2	**101±1.57**	100±0	**206±25.4**	65.3±40.5	95±15	**130±48.7**

J.3.2 Equivalent Number of Sequential Low-level Environment Steps

Here we set the total number of *sequential* low-level environment steps for all the baselines to be the same. This is approach is different from setting the total number of low-level environment steps to be the same, as it acknowledges the more realistic scenario of performing parallel environment rollouts on modern GPUs. With parallelized rollouts, the computational bottleneck becomes the number of *sequential* steps within each environment, as the overhead of increasing the number of parallel environments is negligible compared to increasing the number of sequential steps—assuming a user has a sufficiently large GPU to perform parallelized rollouts of the environment. Such an assumption is common in practice [68], with many implementations of PPO and simulation environments natively supporting parallelized rollouts for the environment [74].

To set the total number of sequential low-level environment steps for all the baselines to be the same, we follow a similar setup as described in Appendix J.3.1, now only accounting for the sequential low-level environment steps that EvoControl uses. Specifically, we first compute the total number of sequential low-level steps that EvoControl uses, where we train EvoControl's high-level policy for 1M steps, and then now train the baseline methods for this increased number of equivalent high-level steps. Specifically if we consider a PD position baseline, originally training this for 1M high-level environment steps, with a lower-level PD position controller, operating at a higher frequency with $G = 16$, meaning that the number of low-level environment steps used in the environment are $1\text{M} \times 16 = 16M$. Here EvoControl, when the high-level is trained for 1M steps, the lower-level policy also receives updates, therefore the total number of sequential low-level environment steps used within the training of EvoControl is $K \times$ es_sub_generations $\times$ episode_length. As es_rollouts and es_pop_size are parallelized, they are not counted in the total number of sequential low-level steps, therefore we leave them as the default values as defined in Appendix G. We leave the other parameters the same as they were for the main results (that of $K = 8$, es_sub_generations $= 8$, episode_length $= 1000$). This leaves the input parameter of es_pop_size that we can vary. Therefore the total number of sequential low-level environment steps used by EvoControl is $1\text{M} \times 16 + 64K$, a fixed amount. Therefore, for the following experiments we train all the other non-EvoControl baselines for $1\text{M} \times 16 + 64K$ low-level environment steps, by specifically determining how many high-level steps this is by dividing by G and using that as the input as the total number of high-level steps to train for each baseline.

For the results, as discussed, we vary es_pop_size $= \{16, 32, 64, 128, 256\}$ and re-run each non-EvoControl baseline with the equivalent number of sequential low-level steps as EvoControl—which means as EvoControl uses Neuroevolution to update the low-level policy, the high-level policy now receives less equivalent updates compared to the high-level policy of the non-EvoControl baselines. We observe in Tables 15 to 19 that EvoControl even when limited to use the same number of sequential low-level environment steps as all the baselines, on average achieves a higher normalized evaluation return than all the baselines *fixed controllers* and *direct torque control*—which remains consistent with the results seen throughout.

Table 15: Ablation. Equivalent Number of Sequential Low-Level Environment Steps, with es_pop_size $= 16$. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±7.29	69.3±0.723	93.7±1.28	100±2.75	100±0.194	99.9±0.0347	100±1.53e-06	100±8.47	100±1.39	85±2.93	82.8±0.176
Fixed Cont. - PD Position Delta	4.81±1.78	3.14±0.101	100±1.55	97.7±1.85	2.6±0.0436	53.9±2.1	100±1.53e-06	0±0	40.3±3.19	15.2±7.6	93.3±0.235
Fixed Cont. - PD Int. Velocity	4.8±1.82	2.8±0.106	75.2±0.875	84±1.34	0±0	48.8±1.93	87.8±1.97	0±0	0±0	0±0	84.1±3.36
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0	19.3±0.355	3.85±0.273	13.4±2.62	15.3±1.23	0±0	0±0	2.95±7.81	0±0	45.3±6.74	1.15±4.13
Direct Torque Cont. - Low Freq. (31.25Hz)	55.6±6.39	100±2.73	75.9±0.496	89.7±2.27	71.9±2.3	100±0.041	100±1.53e-06	75.4±12.8	59.2±3.7	100±1.94	100±0.384
EvoControl (Full State)	**116±62.6**	**104±22**	**127±52.3**	94.7±32.6	88.2±6.4	**100±0.553**	100±0	**272±112**	95.5±65.5	**105±3.31**	**130±63.9**
EvoControl (Residual State)	**163±118**	**153±55**	96.5±5.44	93.8±40.1	95.6±30.9	99.2±2.81	100±0	**314±43.1**	**107±1.46**	**105±2.39**	**194±3.42**
EvoControl (Target + Proprio.)	**121±54.9**	92.1±20.7	**131±16.9**	**117±10.2**	**101±3.66**	99.4±12.5	100±0	**311±59.8**	**110±5.76**	**106±2.39**	**137±1.04**
EvoControl (Target)	**129±78.2**	**115±59.4**	**133±2.96**	**110±13.1**	87.8±5.32	**100±2.15**	100±0	**303±53.2**	71.3±46.9	**104±4.91**	**143±382**
EvoControl (Learned Gains)	46.8±54.2	58.6±8.22	**128±74.8**	82.6±7.06	91.1±8.77	75.6±139	100±0	**228±41.4**	**103±13.9**	97.3±1.76	**111±307**
EvoControl (Delta Position)	**117±33.8**	**104±57.3**	**154±41.2**	84.9±23.5	67.6±56.7	98.1±24.1	100±0	**285±52.1**	58.6±6.77	99.7±3.16	**135±68.1**

Table 16: Ablation. Equivalent Number of Sequential Low-Level Environment Steps, with es_pop_size $= 32$. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±5.89	60.8±0.558	89.5±1.2	89.2±3.27	100±1.83	99.8±0.0361	100±1.53e-06	100±8.47	100±1.8	85±2.93	76.8±0.193
Fixed Cont. - PD Position Delta	2.14±1.8	2.77±0.0891	100±1.93	100±1.7	2.73±0.0726	53.9±2.1	100±1.53e-06	0±0	41±3.24	15.2±7.6	89.6±0.171
Fixed Cont. - PD Int. Velocity	1.82±1.66	2.47±0.0937	73±0.882	85±1.21	0±0	48.8±1.93	86.4±2.01	0±0	0±0	0±0	81.3±3.18
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0	16.9±0.325	3.54±0.253	8.28±2.31	21.9±2.06	0±0	0±0	2.95±7.81	0±0	45.3±6.74	0±0
Direct Torque Cont. - Low Freq. (31.25Hz)	45.1±4.76	100±1.7	79.3±1.22	93.7±2.87	69.6±2.38	100±0.041	100±1.53e-06	73.1±12.7	59.1±3.75	100±1.94	100±0.637
EvoControl (Full State)	**150±46.9**	**147±10**	**132±27.3**	97.8±33.7	**105±17.5**	**100±0.451**	100±0	**305±32.8**	**115±2.56**	**105±1.89**	**138±75.6**
EvoControl (Residual State)	**155±102**	**121±11.3**	93.7±4.42	**116±19**	**121±9.93**	100±1.45	100±0	**323±60.3**	94±66.9	**105±3.69**	**161±327**
EvoControl (Target + Proprio.)	**156±33.6**	**111±49.3**	**141±20.5**	**126±12.3**	**117±22**	99.3±0.367	100±0	**324±40.8**	58.1±2.89	**106±2.35**	**137±79.6**
EvoControl (Target)	**164±56.5**	**116±2.59**	**136±27.6**	**129±4.94**	**110±16.7**	98.7±12.2	100±0	**291±104**	88.7±62.8	**105±0.546**	**139±23**
EvoControl (Learned Gains)	86.3±84.9	59.9±11.8	**162±171**	**105±18.5**	**103±21.7**	99.1±12.1	100±0	**303±57.5**	**110±6.3**	**102±4.43**	**155±160**
EvoControl (Delta Position)	**156±78.1**	**101±35.5**	**111±49.3**	**101±7.07**	99.7±8.03	**100±3.02**	100±0	**302±28.8**	58.9±5.31	**102±9.45**	**127±60.5**

Table 17: Ablation. Equivalent Number of Sequential Low-Level Environment Steps, with es_pop_size $= 64$. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±5.05	64.2±0.518	91.8±1.13	98±2.75	100±0.123	99.9±0.03	100±1.53e-06	100±8.47	100±1.8	85±2.93	82.2±0.371
Fixed Cont. - PD Position Delta	3.6±1.66	2.9±0.0928	100±1.55	100±1.79	2.49±0.0711	53.8±1.57	100±1.53e-06	0±0	41±3.24	15.2±7.6	99.7±0.285
Fixed Cont. - PD Int. Velocity	4±1.56	2.57±0.0976	75.2±0.875	90.1±1.48	0±0	49.7±1.55	86.5±2	0±0	0±0	0±0	93.8±3.03
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0	18.2±0.332	1.31±0.536	39.3±4.2	11.4±0.92	0±0	0±0	4.36±7.75	0±0	45.3±6.74	0±0
Direct Torque Cont. - Low Freq. (31.25Hz)	48.2±5.32	100±1.64	77.5±0.571	95.6±2.64	74.5±1.78	100±0.0311	100±1.53e-06	75.4±12.8	59.1±3.75	100±1.94	100±0.592
EvoControl (Full State)	**175±5.49**	**163±6.17**	**148±32.6**	**115±25.4**	**103±27.3**	**101±0.306**	100±0	**348±3.02**	**114±5.95**	**106±2.17**	**166±85.1**
EvoControl (Residual State)	**143±73.5**	**163±32.4**	**170±162**	**144±7.81**	**116±10.2**	99.4±0.305	100±0	**349±8.24**	**108±1.2**	**105±3.87**	**228±27.1**
EvoControl (Target + Proprio.)	**183±124**	**126±57.2**	**164±48.3**	**137±13**	**114±12.7**	87.3±53.4	100±0	**335±48.2**	**114±2.93**	**106±2.3**	**157±51.3**
EvoControl (Target)	**180±67.6**	**176±12.8**	**132±18.4**	**138±6.19**	**126±66**	99.7±1.7	100±0	**326±41.6**	91.2±72.9	**105±3.25**	**157±79**
EvoControl (Learned Gains)	**107±26.7**	91.2±24.8	**198±195**	**110±29.7**	**104±1.37**	94.1±26.1	100±0	**291±19**	**112±6.83**	**104±1.37**	**211±37.8**
EvoControl (Delta Position)	**180±18.7**	**117±25.1**	**131±36.6**	**112±9.84**	59.4±30.5	**101±0.492**	100±0	**342±70**	61.7±9.32	**101±7.19**	**159±36.7**

Table 18: Ablation. Equivalent Number of Sequential Low-Level Environment Steps, with es_pop_size $= 128$. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheeetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±5.68	67.5±0.452	82.2±1.12	100±2.82	100±0.151	99.9±0.03	100±1.53e-06	100±8.47	100±1.8	85.2±2.87	75.5±0.396
Fixed Cont. - PD Position Delta	3.75±1.68	3.04±0.0975	100±1.17	92.9±1.59	2.4±0.064	53.8±1.57	100±1.53e-06	0±0	41.1±3.24	15.2±7.6	92.9±0.185
Fixed Cont. - PD Int. Velocity	1.81±1.6	2.7±0.103	67.2±0.812	81.4±1.29	0±0	49.7±1.55	86.5±2	0±0	0±0	0±0	86.5±2.67
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0	18.8±0.359	1.27±0.479	10±1.85	15.9±0.789	0±0	0±0	2.95±7.81	0±0	45.3±6.74	0±0
Direct Torque Cont. - Low Freq. (31.25Hz)	36.8±4.92	100±2.11	65.2±0.505	90.4±2.1	87.4±1.02	100±0.0311	100±1.53e-06	73.1±12.7	60.6±3.71	100±1.94	100±0.416
EvoControl (Full State)	**258±60.9**	**124±13.1**	**157±62.4**	99.4±13.6	93.4±5.77	**101±0.0993**	100±0	**346±3.75**	**115±3.34**	**106±2.28**	**184±45.5**
EvoControl (Residual State)	**166±86.1**	**180±23.9**	**128±163**	**147±17.9**	**140±91.5**	99.8±0.474	100±0	**340±30.4**	**109±4.07**	**104±5.95**	**211±39.8**
EvoControl (Target + Proprio.)	**241±45.1**	**198±4.9**	**172±68.9**	**139±13**	**134±106**	90.6±37.6	100±0	**363±125**	**116±2.5**	**106±2.07**	**163±68.3**
EvoControl (Target)	**218±53.3**	**197±5.49**	**165±93.1**	**132±20.7**	**134±99.5**	92.4±31.5	100±0	**335±40.4**	**109±2.03**	**105±1.27**	**155±21.1**
EvoControl (Learned Gains)	**165±85.1**	**107±10.1**	**170±174**	**124±11.7**	98.1±7.44	99.7±3.18	100±0	**309±49.7**	**112±8.66**	**104±5.15**	**194±87.8**
EvoControl (Delta Position)	**216±20.1**	**125±27**	**114±153**	**115±12.1**	94.4±21.9	**101±0.13**	100±0	**334±16.5**	59.4±6.88	95.9±21.7	**190±74.7**

Table 19: Ablation. Equivalent Number of Sequential Low-Level Environment Steps, with es_pop_size $= 256$. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±6.25	63.2±0.599	89.3±1.2	92.6±2.05	100±0.208	99.9±0.03	100±1.53e-06	100±8.47	100±1.77	85±2.93	81.7±0.384
Fixed Cont. - PD Position Delta	2.57±1.74	2.86±0.092	100±1.84	100±1.78	2.54±0.0542	53.8±1.57	100±1.53e-06	0±0	40.8±3.23	15.2±7.6	96.3±0.219
Fixed Cont. - PD Int. Velocity	2.46±1.51	2.55±0.0967	73.1±0.844	86.6±1.33	0±0	49.7±1.55	86.5±2	0±0	0±0	0±0	93.7±2.77
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0	18±0.353	0.882±0.533	7.93±1.94	10.1±0.357	0±0	0±0	4.36±7.75	0±0	45.3±6.74	0±0
Direct Torque Cont. - Low Freq. (31.25Hz)	63±7.07	100±1.54	70.6±0.548	85.6±1.41	80±1.73	100±0.0311	100±1.53e-06	75.4±12.8	58.8±3.74	100±1.94	100±1.06
EvoControl (Full State)	**285±52.7**	**161±19.5**	**190±40.5**	**126±9.29**	**105±29.9**	**101±0.522**	100±0	**349±27.4**	97.9±70.9	**106±2.16**	**169±58.1**
EvoControl (Residual State)	**161±49.5**	**188±1.91**	95.7±5.8	**153±13.7**	**175±160**	**100±0.559**	100±0	**351±54.3**	**112±1.46**	**105±2.97**	**204±60.3**
EvoControl (Target + Proprio.)	**293±108**	**134±21.4**	**202±134**	**165±36.2**	**140±76.2**	98.8±5	76.9±99.4	**338±45**	**114±3.76**	**106±2.18**	**179±30.8**
EvoControl (Target)	**248±25.5**	**165±31.1**	**175±86.6**	**152±31.8**	**150±148**	99.3±2.17	100±0	**356±14.6**	95.6±23.8	**105±2.87**	**148±31.3**
EvoControl (Learned Gains)	**214±12.7**	**111±11.2**	**128±69**	**125±72.2**	**102±8.52**	**100±0.201**	100±0	**327±81.4**	**115±0.874**	**105±2.67**	**229±56.9**
EvoControl (Delta Position)	**281±76.8**	**141±31.4**	**202±848**	**121±8.27**	96.3±16.3	**101±1.56**	100±0	**342±12.7**	71.9±44.6	97.4±15.4	**173±32.4**

J.4 Ablation: No annealing with PD Controller

Here we conduct an ablation of removing the gradual annealing with a PD controller throughout training. Specifically, to do this we set $\alpha = 0$, therefore meaning that within EvoControl the high-level policy ρ output latent action a_k directly goes into the initially un-trained low-level neural network policy β. We observe in Table 20, that EvoControl as presented in the main paper, the inclusion of annealing with a PD controller throughout learning does help EvoControl to achieve a higher normalized return on average compared to no annealing with a PD controller, as shown in the ablation. This provides empirical evidence for it's inclusion, which could be explained by the intuitive arguments presented in the paper, of helping the higher-level policy ρ to learn a stable policy using an initial goal-tracking sub-policy of a PD controller, and then switch to an improved learned low-level controller throughout training. We also note, that although EvoControl performs well with the inclusion of the annealed PD controller, not having the PD controller, it also performs acceptably compared to the baselines. However, for best performance we recommend users to use the annealing with a PD controller.

Table 20: Main table of results (Table 3), with the ablation of EvoControl with no PD controller annealing (by setting $\alpha = 0$). Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	100±6.56	61.2±0.441	91.6±1.23	100±2.96	100±0.974	99.9±0.03	100±1.53e-06	100±8.47	100±1.8	85.2±2.87	75.7±0.633
Fixed Cont. - PD Position Delta	2.4±1.91	2.76±0.0888	100±1.35	96.6±1.71	2.96±0.0397	53.8±1.57	100±1.53e-06	0±0	40.9±3.23	15.2±7.6	90.2±0.239
Fixed Cont. - PD Int. Velocity	3.59±1.78	2.46±0.0932	74.7±0.903	83.4±1.13	0±0	49.7±1.55	86.5±2	0±0	0±0	0±0	85.9±2.55
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0±0	17.2±0.316	1.42±0.533	10.4±2.19	10.3±0.586	0±0	0±0	1.34±7.89	2.08±5.84	45.3±6.74	0±0
Direct Torque Cont. - Low Freq. (31.25Hz)	54.5±7.15	100±1.21	72±0.64	98±2.55	80.6±2.56	100±0.0311	100±1.53e-06	73.2±12.9	59.2±3.72	100±1.94	100±2.68
EvoControl (Full State)	**368±73.2**	**157±18.3**	**274±20.6**	**123±19**	**116±18.4**	**101±0.859**	100±0	**362±12.5**	**114±7.51**	**106±2.75**	**203±137**
EvoControl (Residual State)	**182±16.1**	**182±6.31**	**101±5.54**	**170±18.2**	**212±145**	99.2±1.25	100±0	**375±94.8**	**106±25.8**	**104±3.42**	**205±57.4**
EvoControl (Target + Proprio.)	**319±35.1**	**168±14.7**	**171±155**	**165±7.19**	**165±150**	99.7±1.19	100±0	**353±28.4**	96.8±78.6	**105±4.71**	**178±63.7**
EvoControl (Target)	**293±87**	**162±23.5**	**283±250**	**164±21.2**	**205±147**	99.6±0.949	100±0	**353±43.4**	**112±1.73**	**105±1.65**	**188±88.2**
EvoControl (Learned Gains)	**266±104**	**113±10.5**	**206±302**	**150±15.1**	**117±2.44**	99.5±2.48	100±0	**330±17.8**	**116±1.62**	**105±2.45**	**196±118**
EvoControl (Delta Position)	**362±47.7**	**133±34.6**	**225±82.9**	**119±18**	**105±4.64**	**101±0.394**	100±0	**267±30.2**	65.5±21	99.1±12.7	**183±34.4**
Ablation - No PD controller annealing ($\alpha = 0$) - **EvoControl (Full State)**	**359±87**	**148±27.8**	**213±142**	**121±17.3**	**116±17.1**	**101±0.51**	100±0	**349±15**	65.5±16	**104±4.52**	**218±39.7**
Ablation - No PD controller annealing ($\alpha = 0$) - **EvoControl (Residual State)**	**164±195**	**121±27.7**	**160±181**	**166±28.2**	**122±25.6**	99.7±1.47	100±0	**366±86.2**	68.4±43.5	**102±7**	**185±109**
Ablation - No PD controller annealing ($\alpha = 0$) - **EvoControl (Target + Proprio.)**	**294±46**	**148±3.82**	**242±193**	**173±18.4**	**165±147**	89.3±43.8	100±0	**357±44.2**	63.2±12.6	**104±5.84**	**188±106**
Ablation - No PD controller annealing ($\alpha = 0$) - **EvoControl (Target)**	**298±31.5**	**145±71.4**	**121±109**	**174±8.98**	**128±16.6**	68.7±78.1	100±0.0223	**356±109**	74.4±41.3	91.6±39.9	**195±74.4**
Ablation - No PD controller annealing ($\alpha = 0$) - **EvoControl (Learned Gains)**	**255±58.2**	**140±19.9**	95.7±127	**156±23.9**	**119±5.95**	**100±0.554**	100±0	**340±49.4**	87.9±40.1	92.7±4.95	**179±117**
Ablation - No PD controller annealing ($\alpha = 0$) - **EvoControl (Delta Position)**	**328±110**	**151±67.6**	**255±116**	**127±2.02**	**120±8.73**	**101±0.365**	100±0	**334±30.9**	60±13.3	96.3±13.3	**236±10.8**

J.5 Ablation: Main Results for More High-level Steps

In the following, we increase the high-level number of steps that each baseline is trained for. Initially in the main results presented in the paper we trained all the results for 1M high-level ρ policy steps. Therefore we ask the question, how do all the results compare if we run all the baselines for the high-level steps of 100M and 1B high-level steps. We tabulate these results in Tables 21 to 23.

Table 21: Additional Experiment. Training all the baselines for a larger amount of high-level ρ policy steps of 10M steps. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	60.8±2.01	68.9±0.858	100±1.23	100±0.281	100±0.308	100±0.033	89.6±1.48	100±4.99	93.8±2.48	84.9±2.56	100±0.775
Fixed Cont. - PD Position Delta	0±0	4.13±0.107	45.4±0.0546	86.5±1.22	1.4±0.0198	98.2±0.485	100±1.53e-06	0.226±7.09	65.5±1.68	25.7±7.17	46.5±0.0729
Fixed Cont. - PD Int. Velocity	0±0	4.15±0.145	45.9±0.0263	58.8±0.93	1.59±0.0629	13.4±0.43	99.6±0.244	0±0	0±0	9.68±7.5	53±0.887
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	0.36±4.11	70±1.51	35±1.66	42.1±1.2	73.4±1.91	97.8±0.148	42.6±1.48	10.7±4.39	36.9±3.9	83.8±3.45	29.8±1.18
Direct Torque Cont. - Low Freq. (31.25Hz)	100±4.06	100±1.39	94.6±3.91	96.3±0.355	79.5±2.01	100±0.0336	100±1.53e-06	41.7±3.83	100±1.33	100±1.66	74.8±2.44
EvoControl (Full State)	**208±4.54**	**231±1.11**	**187±0.629**	**104±0.802**	**118±0.172**	**102±8.45e-05**	100±1.53e-06	**223±3.06**	97.1±0.738	99.7±1.4	**222±1.12**
EvoControl (Residual State)	**113±3.74**	**208±1.23**	**128±4.11**	**124±0.865**	**228±4.5**	**101±0.0953**	100±1.53e-06	**228±3.03**	**103±0.646**	**101±0.634**	**150±0.501**
EvoControl (Target + Proprio.)	**178±3.88**	**230±0.842**	**177±0.467**	**169±0.922**	**206±6.11**	83.6±1.65	100±0.000679	**222±2.92**	96.7±0.865	97.8±1.2	**226±1.36**
EvoControl (Target)	**174±4.38**	**231±0.617**	**185±0.815**	**148±1.19**	**254±6.5**	87.3±1.89	100±1.53e-06	**224±2.84**	92.7±1.08	99.5±1.04	**203±1.72**
EvoControl (Learned Gains)	**147±4.81**	**194±1.13**	**172±0.682**	**110±1.06**	**112±0.506**	**102±0.0572**	100±1.53e-06	**200±4.62**	**102±0.651**	99.8±0.836	**186±1.55**
EvoControl (Delta Position)	**187±4.08**	**209±1.55**	**185±0.188**	98.6±0.824	**115±0.389**	**102±6.05e-05**	100±1.53e-06	**155±5.18**	93.4±1.03	98.4±0.777	**179±0.635**

Table 22: Additional Experiment. Training all the baselines for a larger amount of high-level ρ policy steps of 100M steps. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	62.5±1.4	38.6±0.485	85.1±2.69	100±0.289	36±2.01	100±0.19	98.9±0.418	100±4.58	100±1.97	85.5±2.5	75.7±2.56
Fixed Cont. - PD Position Delta	2.34±0.431	4.14±0.0897	53.9±0.362	94.8±1.17	1.09±0.0471	99.9±0.25	100±1.53e-06	85.6±5.35	78.1±1.84	20.4±7.39	44.3±0.0682
Fixed Cont. - PD Int. Velocity	2.61±1.03	4.08±0.132	24.9±2.14	68.4±1.68	35±1.03	34.6±2.79	44.6±4.01	37.7±6.42	0±0	12.1±7	52.4±1.14
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	29.8±4.11	100±1.82	77.1±2.23	75.1±2.87	100±2.88	86.5±1.06	63.2±2.86	9.06±6.19	84±2.74	65.3±6.04	72.7±2.65
Direct Torque Cont. - Low Freq. (31.25Hz)	100±3.6	60.6±1.53	100±5.62	96.3±0.501	77.7±1.17	99.9±0.281	100±1.53e-06	72.5±4.49	92.7±3.2	100±2.19	100±3.95
EvoControl (Full State)	**188±3.19**	**191±0.581**	**227±0.36**	**159±1.64**	**199±3.11**	**100±3.24e-05**	100±1.53e-06	**278±3.5**	**102±0.708**	**102±0.577**	**264±1.06**
EvoControl (Residual State)	**109±3.06**	**151±2.44**	**186±1.95**	**163±0.807**	**194±6.19**	**100±0.000688**	100±1.53e-06	**284±3.14**	**104±1.47**	**102±0.826**	**211±3.09**
EvoControl (Target + Proprio.)	**155±3.15**	**192±0.605**	**216±1.79**	**192±1**	**241±4.83**	96.7±0.226	100±0.0112	**283±2.86**	**103±0.782**	96.8±1.3	**257±1.58**
EvoControl (Target)	**147±2.77**	**192±0.499**	**213±2.3**	**189±1.66**	**261±2.19**	96.4±0.314	98.9±0.791	**279±3.23**	96.9±1.03	88.1±2.05	**260±0.723**
EvoControl (Learned Gains)	**117±3.68**	**151±1.37**	**215±0.793**	**123±0.874**	63.6±2.6	**100±4.2e-05**	100±1.53e-06	**259±4.17**	**103±0.844**	**101±0.848**	**180±2.09**
EvoControl (Delta Position)	**180±3.45**	**186±1.03**	**225±1.32**	**122±1.86**	**111±0.265**	**100±3.4e-05**	100±1.53e-06	**253±5.79**	**104±0.65**	**102±0.586**	**207±2.45**

Table 23: Additional Experiment. Training all the baselines for a larger amount of high-level ρ policy steps of 1B steps. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	45.4±2.72	29.8±0.662	65.8±1.03	100±0.542	100±0.959	98.1±0.743	98.2±0.595	100±1.6	100±1.84	100±2.82	50.2±2.42
Fixed Cont. - PD Position Delta	1.63±0.459	4.2±0.0834	60.9±0.057	90.9±1.02	1.13±0.0683	100±0.353	100±1.53e-06	46.5±5.03	76.6±1.77	22.1±8.77	33.2±0.0273
Fixed Cont. - PD Int. Velocity	4.86±0.755	0±0	17.2±2.89	56.5±1.92	36.5±0.488	83.6±2.23	19.6±3.86	30.4±2.04	0±0	6.56±8.56	4.81±1.27
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	40.8±3.72	±	100±1.74	86.3±2.36	52.9±3.39	93.8±0.622	95.3±0.751	±	88.6±2.27	82.9±6.85	±
Direct Torque Cont. - Low Freq. (31.25Hz)	100±3.25	100±0.983	±	93.4±0.18	74.4±1.41	95.9±1.17	98.4±0.557	86.9±2.08	±	±	100±0.646
EvoControl (Full State)	**220±4.55**	**197±0.54**	**285±0.424**	**159±0.844**	**185±1.02**	**101±5.28e-05**	100±1.53e-06	**106±1.63**	**104±0.61**	**120±0.664**	**196±0.435**
EvoControl (Residual State)	**134±3.24**	**149±2.34**	**270±1.21**	**159±1.79**	**365±0.986**	**101±0.000788**	100±1.88e-06	**111±1.31**	95.3±2.41	**120±0.727**	**155±0.924**
EvoControl (Target + Proprio.)	**182±3.71**	**194±0.564**	**269±0.707**	**183±1.01**	**340±4.46**	96.5±0.391	100±1.88e-06	**114±1.14**	98.7±1.19	**112±1.43**	**181±0.835**
EvoControl (Target)	**174±4.07**	**194±0.527**	**269±0.71**	**180±0.805**	**355±1.42**	97.5±0.179	100±1.53e-06	**113±1.22**	96.5±1.21	**112±1.5**	**198±0.428**
EvoControl (Learned Gains)	87.3±1.98	39.3±0.435	80.7±0.277	**103±0.745**	83.1±2.83	**101±0.0651**	99.9±0.204	76.9±3.27	94.7±2.51	**103±2.44**	78.9±3.05
EvoControl (Delta Position)	**215±3.81**	**195±0.499**	**269±1.15**	**121±0.526**	**158±1.55**	**101±4.84e-05**	100±1.53e-06	**112±1.35**	**108±0.478**	**120±0.667**	**167±0.592**

J.6 Main Table of Results Additional Metrics

For the main table of results presented in the paper (Table 3), we also provide un-normalized results in Table 24 and the time taken to train the policies in Table 25. Regarding the training time, the wall-clock time for EvoControl can be substantially improved, as the PPO implementation we use to train the high-level policies is implemented in Jax, and was pre-compiled, across a batch of environments, using Jax based environments (Brax). Whereas the Neuroevolution could further be compiled, however for our implementation it was not, and only the population of rollouts was compiled in Jax. Ideally for optimal performance the entire Neuroevolution step could be compiled in Jax leading to speed improvements, however given that all the baselines, including EvoControl could finish training their policies within a time interval of approximately one hour, further optimization was not necessary.

Table 24: Un-normalized evaluation return R for the benchmark methods, across each environment. EvoControl on average achieves a higher evaluation return R than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	1.47e+03±31	1.09e+03±8.22	1.05e+03±12.7	3.52e+03±74	1.88e+05±1.43e+03	9.21e+03±1.97	1e+03±0	-2.36e+03±26.2	-46.6±2.44	-20.3±2.08	895±7.13
Fixed Cont. - PD Position Delta	1.01e+03±9.04	3.6±1.66	1.14e+03±13.9	3.43e+03±42.9	4.51e+04±58.3	6.18e+03±103	1e+03±0	-2.73e+03±26.6	-127±4.38	-70.9±5.5	1.06e+03±2.69
Fixed Cont. - PD Int. Velocity	1.01e+03±8.43	-2.14±1.74	876±9.33	3.1e+03±28.2	4.05e+04±17.5	5.91e+03±102	898±15.1	-2.67e+03±26.2	-229±7.85	-84.7±5.62	1.01e+03±28.7
Fixed Cont. - Random	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	920±41.1	273±5.89	119±5.5	1.28e+03±54.8	5.59e+04±862	722±1.23	106±3.72	-2.67e+03±24.4	-179±7.92	-49.1±4.87	-36.9±24.9
Direct Torque Cont. - Low Freq. (31.25Hz)	1.26e+03±33.8	1.82e+03±22.5	848±6.61	3.47e+03±63.6	1.59e+05±3.76e+03	9.22e+03±2.05	1e+03±0	-2.44e+03±39.9	-102±5.04	-9.55±1.41	1.17e+03±30.1
EvoControl (Full State)	2.74e+03±347	2.88e+03±341	2.94e+03±212	4.08e+03±474	2.11e+05±2.71e+04	9.28e+03±56.5	1e+03±0	-1.55e+03±38.7	-27.6±10.2	-5.44±1.99	2.32e+03±1.54e+03
EvoControl (Residual State)	1.86e+03±76.2	3.35e+03±118	1.15e+03±57.3	5.27e+03±456	3.52e+05±2.13e+05	9.17e+03±82.2	1e+03±0	-1.51e+03±293	-38.5±35	-6.69±2.47	2.35e+03±646
EvoControl (Target + Proprio.)	2.51e+03±166	3.08e+03±274	1.88e+03±1.6e+03	5.15e+03±180	2.83e+05±2.21e+05	9.2e+03±78.3	1e+03±0	-1.58e+03±88	-50.9±107	-5.74±3.4	2.05e+03±716
EvoControl (Target)	2.38e+03±412	2.96e+03±438	3.03e+03±2.68e+03	5.11e+03±530	3.42e+05±2.16e+05	9.2e+03±62.4	1e+03±0	-1.58e+03±134	-30.8±2.35	-5.78±1.19	2.15e+03±993
EvoControl (Learned Gains)	2.26e+03±492	2.06e+03±195	2.23e+03±3.12e+03	4.77e+03±378	2.13e+05±3.58e+03	9.19e+03±163	1e+03±0	-1.65e+03±55	-24.9±2.19	-6±1.77	2.25e+03±1.32e+03
EvoControl (Delta Position)	2.71e+03±226	2.43e+03±644	2.43e+03±856	4e+03±450	1.95e+05±6.82e+03	9.27e+03±26	1e+03±0	-1.84e+03±93.5	-93.4±28.5	-10.2±9.16	2.1e+03±387

Table 25: Time Taken to train in minutes for each baseline against each environment, for the main table of results in Table 3. All the baselines including EvoControl can train their policies on average within an hour. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. As the random baseline does not perform any training, we put a placeholder of 0 for it.

Same PPO high-level alg. ρ with a Low-Level Policy β of	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	0.671±0.0112	1.15±0.0309	0.886±0.0451	0.574±0.281	1.36±0.297	0.536±0.0173	0.503±0.011	0.799±0.285	0.587±0.00373	0.519±0.00276	1.06±0.0154
Fixed Cont. - PD Position Delta	0.668±0.0118	1.08±0.00826	0.879±0.0303	0.576±0.285	1.19±0.283	0.542±0.00226	0.475±0.00815	0.847±0.332	0.595±0.0164	0.473±0.0101	1.16±0.0178
Fixed Cont. - PD Int. Velocity	0.662±0.0253	1.06±0.00543	0.888±0.0169	0.574±0.3	1.11±0.303	0.53±0.0161	0.506±0.0181	0.679±0.27	0.573±0.0182	0.503±0.0125	1.2±0.0349
Fixed Cont. - Random	0±0	0±0	0±0	0±0	0±0	0±0	0±0	0±0	0±0	0±0	0±0
Direct Torque Cont. - High Freq. (500Hz)	0.597±0.0392	0.963±0.0137	0.738±0.0213	0.513±0.273	1.06±0.277	0.457±0.00473	0.424±0.0121	0.679±0.291	0.585±0.02	0.501±0.00844	0.899±0.0231
Direct Torque Cont. - Low Freq. (31.25Hz)	0.663±0.041	1.14±0.0165	0.908±0.00972	0.582±0.277	1.4±0.281	0.583±0.016	0.513±0.0115	0.917±0.274	0.736±0.0267	0.58±0.0301	1.09±0.0476
EvoControl (Full State)	12.8±0.0925	23.3±0.11	14.8±0.0829	20.4±0.505	77.3±0.481	8.33±0.0986	6.93±0.0438	31.4±1.34	5.74±0.0747	4.6±0.0386	25.4±0.226
EvoControl (Residual State)	11.2±0.0999	23.2±0.106	14.3±0.0904	18±0.466	76.1±0.407	7.94±0.0885	7±0.0656	31.1±0.725	5.84±0.331	4.59±0.00888	23.7±0.112
EvoControl (Target + Proprio.)	12.7±0.0899	23.1±0.0987	14.5±0.103	19.5±0.499	76.7±0.205	8.25±0.0224	6.89±0.00579	31.5±0.256	5.71±0.231	4.55±0.0514	24.1±0.154
EvoControl (Target)	11.4±0.0776	22.9±0.156	14.5±0	19.4±0.467	76.7±0.399	8.33±0.139	7.09±0.0406	30.7±1.27	5.66±0.119	4.55±0.0235	24±0.279
EvoControl (Learned Gains)	12.9±0.106	23.1±0.0689	13.3±1.06	20.3±0.517	76.8±0.173	8.09±0.0643	7.13±0.0462	27.4±1.52	5.64±0.16	4.51±0.0625	25.3±0.34
EvoControl (Delta Position)	12.8±0.124	22.3±0.162	14.8±0.175	20.5±0.479	73.2±0.733	8.39±0.0537	6.93±0.0197	29.2±1.25	5.72±0.151	4.59±0.0536	25.7±0.0329

J.7 Learning Curves for All Baselines

We provide the learning curves for all environments presented in the main table of results in the paper, that of Table 3. Specifically, we provide these plots in Figures 3 to 12. We observe that EvoControl on average consistently outperforms the non-EvoControl baselines and achieves a higher evaluation return.

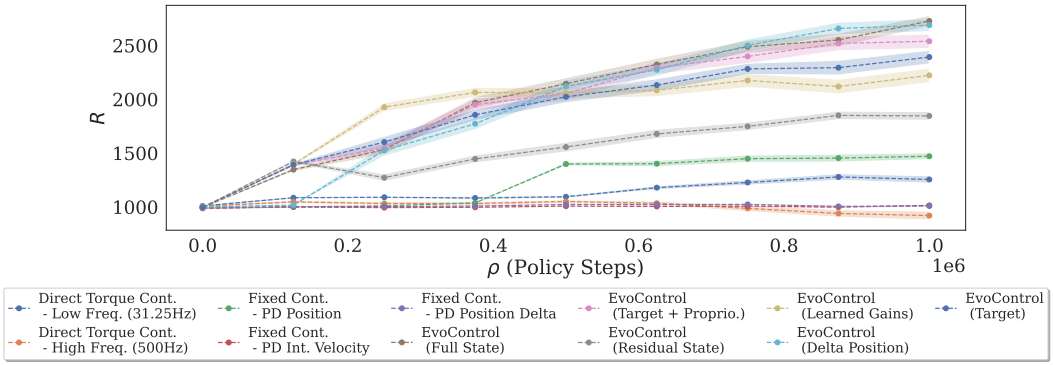

Figure 3: Evaluation return R versus ρ policy steps on Ant, for main table of results Table 3.

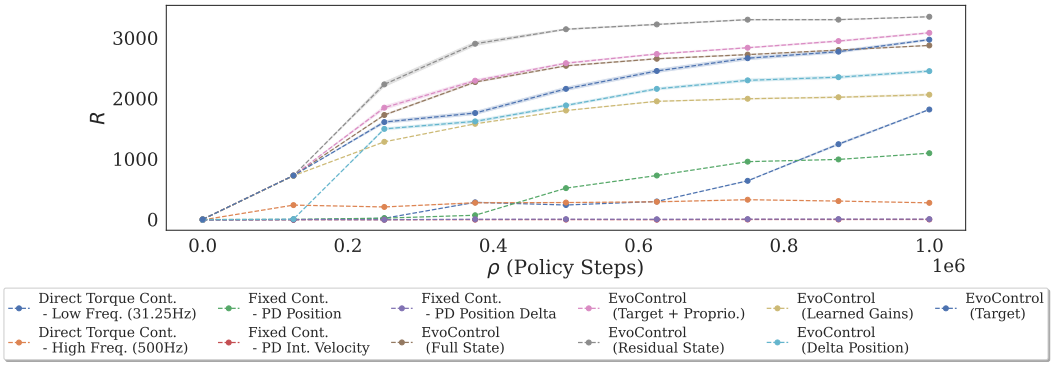

Figure 4: Evaluation return R versus ρ policy steps on Halfcheetah, for main table of results Table 3.

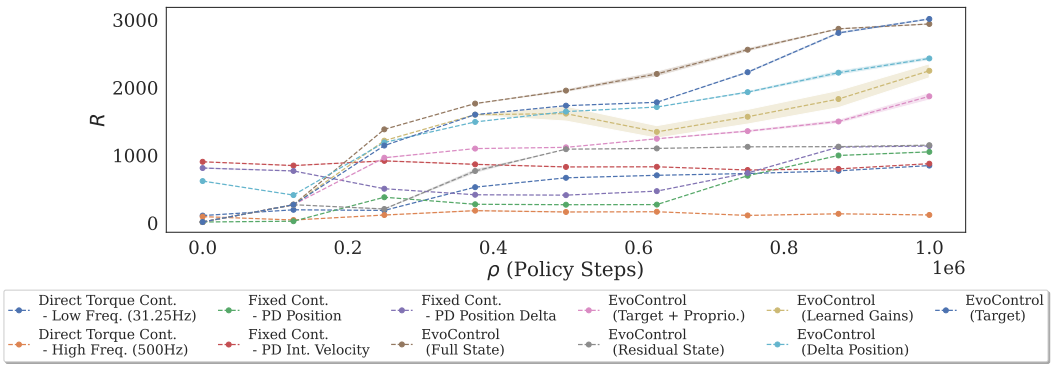

Figure 5: Evaluation return R versus ρ policy steps on Hopper, for main table of results Table 3.

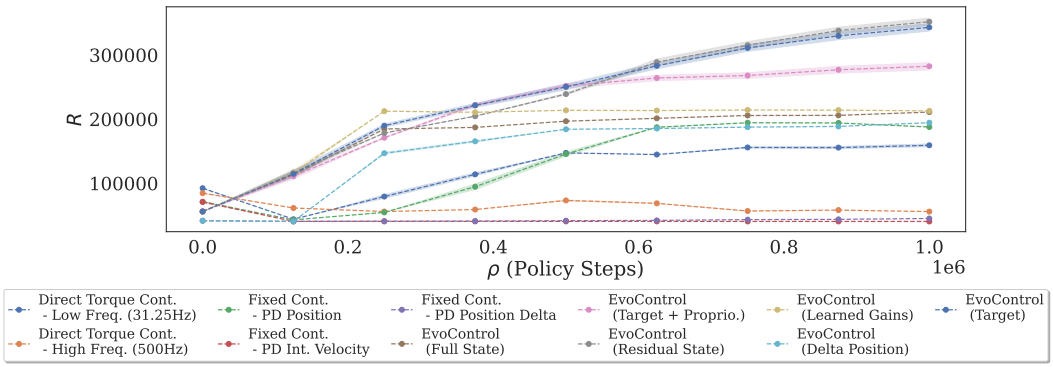

Figure 6: Evaluation return R versus ρ policy steps on Humanoid Standup, for main table of results Table 3.

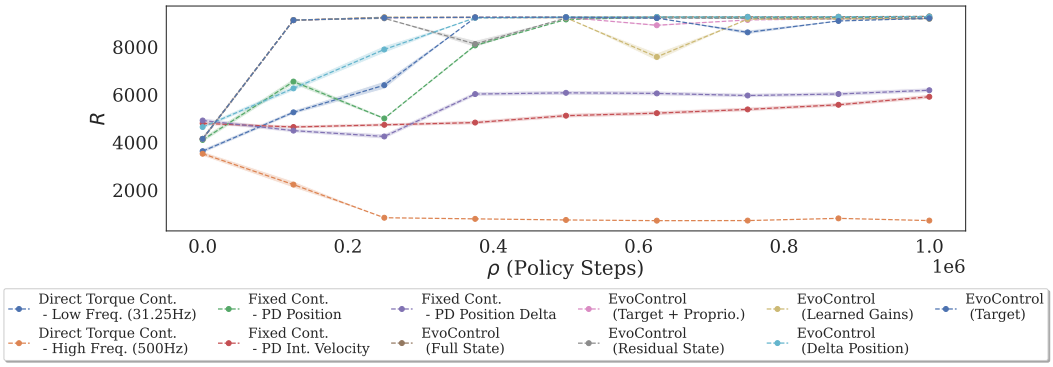

Figure 7: Evaluation return R versus ρ policy steps on Inverted Double Pendulum, for main table of results Table 3.

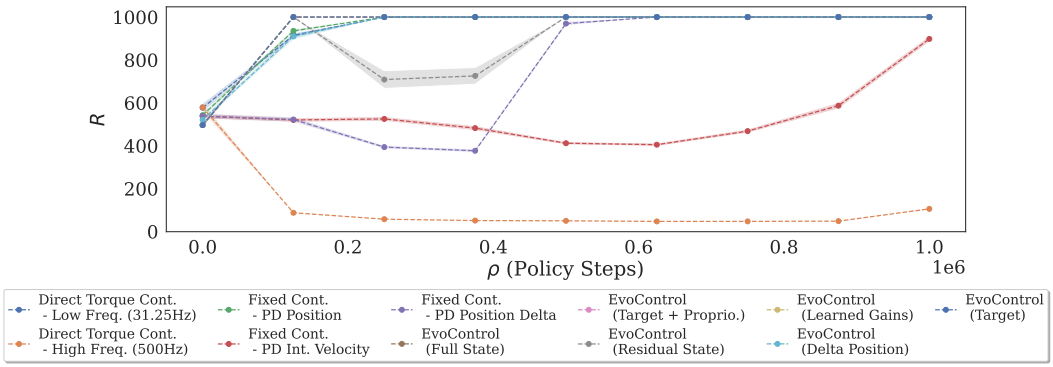

Figure 8: Evaluation return R versus ρ policy steps on Inverted Pendulum, for main table of results Table 3.

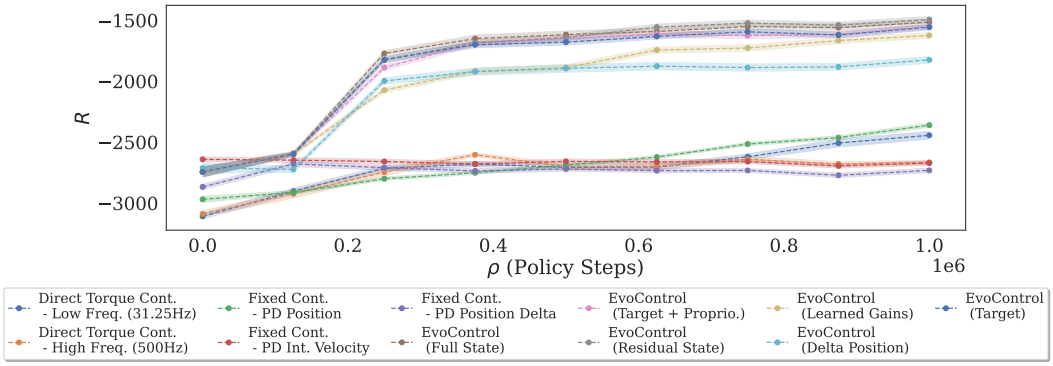

Figure 9: Evaluation return R versus ρ policy steps on Pusher, for main table of results Table 3.

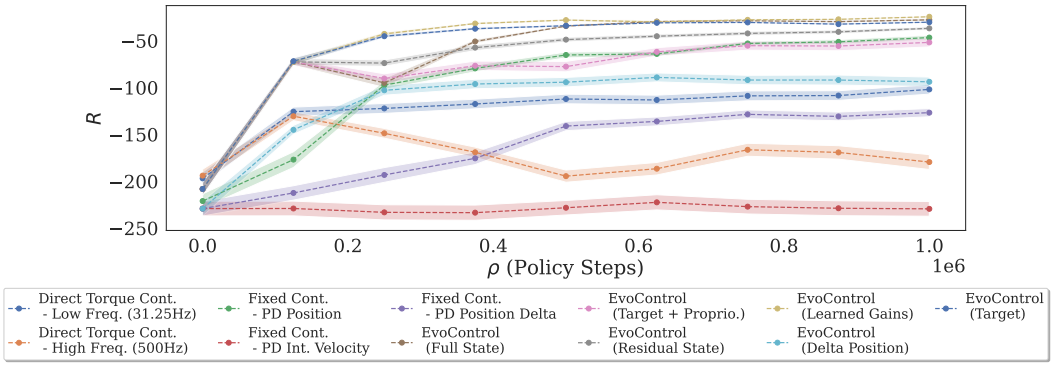

Figure 10: Evaluation return R versus ρ policy steps on Reacher, for main table of results Table 3.

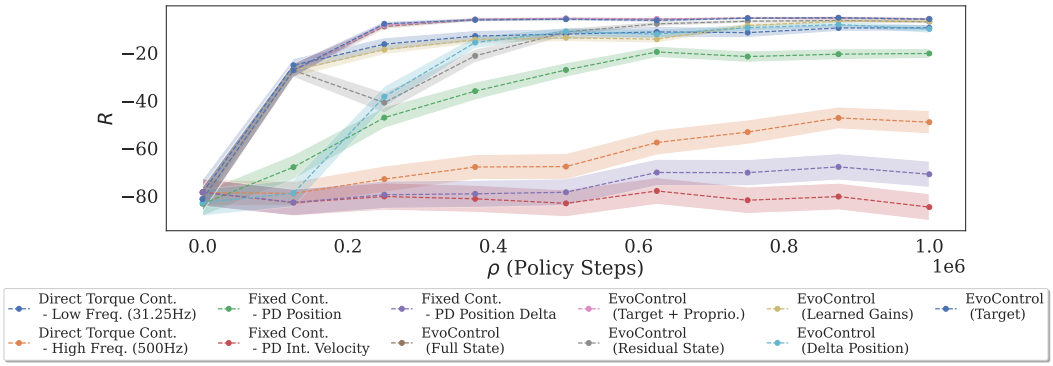

Figure 11: Evaluation return R versus ρ policy steps on Reacher 1D, for main table of results Table 3.

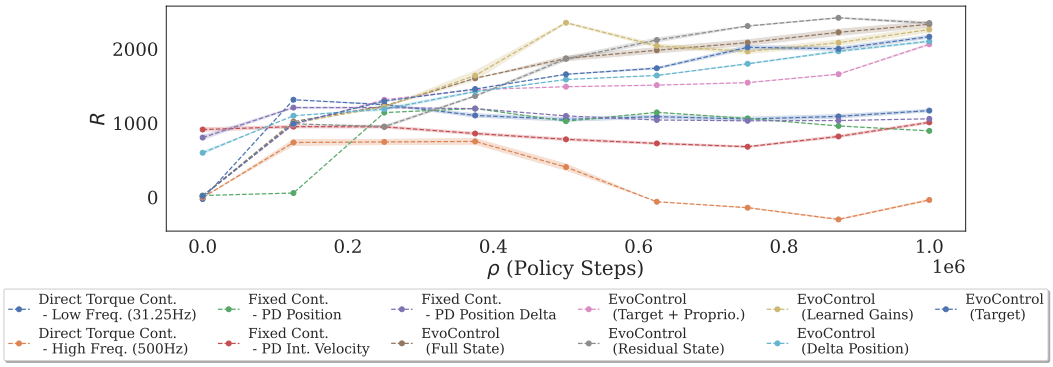

Figure 12: Evaluation return R versus ρ policy steps on Walker2D, for main table of results Table 3.

J.8 Ablation: Training High-Level Policy with Neuroevolution

In the following we perform an ablation whereby we train the high-level policy with Neuroevolution rather than PPO. We provide two variations: first, using Neuroevolution to train the high-level policy with the same fixed low-level controllers for all non-EvoControl baselines (Table 26), and second, within EvoControl modifying training the high-level policy with Neuroevolution instead of PPO.

First, we replace the high-level PPO policy, keeping the same high-level policy architecture and network as originally used in the main paper, and instead using the same Neuroevolution as was used to train the low-level policy. For this we provide two versions, A) that of training the high-level policy with the same number of high-level ρ steps as the core baselines presented in the paper, with 1M ρ steps—labelled *Neuroevolution (1M ρ steps)*. Specifically, this uses $es\_rollouts = 1, K = 8, es\_pop\_size = 256, es\_sub\_generations = 8$. Second, B) training the high-level policy with the same number of environment steps as was used to train the low-level policy within EvoControl—labelled *Neuroevolution (same as EvoControl low-level)*. Specifically this uses more low-level environment steps than A), with $es\_rollouts = 16, K = 8, es\_pop\_size = 512, es\_sub\_generations = 8$. The results are tabulated alongside the original main table of results for ease of comparison in Table 26. We observe on average that EvoControl still outperforms the competitive baselines, and on average achieves a higher normalized evaluation return $\mathcal{R}$ than the baselines of *fixed-controllers* and *direct torque control*.

Table 26: Ablation. Training the high-level policy with Neuroevolution instead of PPO. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Method Name	High-level ρ with	Low-level β with	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher ID $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	PPO	PD Position	100±6.25	63.2±0.599	89.3±1.2	92.6±2.05	100±0.208	99.9±0.03	100±1.53e-06	100±8.47	100±1.77	85±2.93	81.7±0.384
Fixed Cont. - PD Position Delta	PPO	PD Position Delta	2.57±1.74	2.86±0.092	100±1.84	100±1.78	2.54±0.0542	53.8±1.57	100±1.53e-06	0±0	40.8±3.23	15.2±7.6	96.3±0.219
Fixed Cont. - PD Int. Velocity	PPO	PD Int. Velocity	2.46±1.51	2.55±0.0967	73.1±0.844	86.6±1.33	0±0	49.7±1.55	86.5±2	0±0	0±0	0±0	93.7±2.77
Fixed Cont. - Random	Random	Direct Torque	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	PPO	Direct Torque	0±0	18±0.353	0.882±0.533	7.93±1.94	10.1±0.357	0±0	0±0	4.36±7.75	0±0	45.3±6.74	0±0
Direct Torque Cont. - Low Freq. (31.25Hz)	PPO	Direct Torque	63±7.07	100±1.54	70.6±0.548	85.6±1.41	80±1.73	100±0.0311	100±1.53e-06	75.4±12.8	58.8±3.74	100±1.94	100±1.06
EvoControl (Full State)	PPO	Neuroevolution	**368±73.2**	**157±18.3**	**274±20.6**	**123±19**	**116±18.4**	**101±0.859**	**100±0**	**362±12.5**	**114±7.51**	**106±2.75**	**203±1.17**
EvoControl (Residual State)	PPO	Neuroevolution	**182±16.1**	**182±6.31**	**101±5.54**	**170±18.2**	**212±145**	99.2±1.25	**100±0**	**375±94.8**	**106±25.8**	**104±3.42**	**205±57.4**
EvoControl (Target + Proprio.)	PPO	Neuroevolution	**319±35.1**	**168±14.7**	**171±155**	**165±7.19**	**165±150**	99.7±1.19	**100±0**	**353±28.4**	96.8±78.6	**105±4.71**	**178±63.7**
EvoControl (Target)	PPO	Neuroevolution	**293±87**	**162±23.5**	**283±250**	**164±21.2**	**205±147**	99.6±0.949	**100±0**	**353±43.4**	**112±1.73**	**105±1.65**	**188±88.2**
EvoControl (Learned Gains)	PPO	Neuroevolution	**266±104**	**113±10.5**	**206±302**	**150±15.1**	**117±2.44**	99.5±2.48	**100±0**	**330±17.8**	**116±1.62**	**105±2.45**	**196±118**
EvoControl (Delta Position)	PPO	Neuroevolution	**362±47.7**	**133±34.6**	**225±82.9**	**119±18**	**105±4.64**	**101±0.394**	**100±0**	**267±30.2**	65.5±21	99.1±12.7	**183±34.4**
Ablation: Fixed Cont. - PD Position	Neuroevolution (1M ρ steps)	PD Position	**136±1.02**	52.5±0.881	**104±1.1**	**154±0.149**	**118±0.205**	**101±0.0186**	**100±1.08e-06**	**390±1.33**	69.1±0.37	86.7±0.179	**179±1.18**
Ablation: Fixed Cont. - PD Position Delta	Neuroevolution (1M ρ steps)	PD Position Delta	36.1±0.185	4.48±0.00862	**133±0.535**	**137±0.738**	4.61±0.00116	97.7±0.782	**100±0.0594**	**279±3.12**	52±0.174	12.2±0.83	**126±0.512**
Ablation: Fixed Cont. - PD Int. Velocity	Neuroevolution (1M ρ steps)	PD Int. Velocity	**128±2.31**	67.6±0.85	96±0.941	**156±0.381**	**120±0.21**	97.9±1.22	55.2±1.81	**256±1.19**	61.2±0.179	59.7±0.869	**130±1.73**
Ablation: Fixed Cont. - Random	Random	Direct Torque	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Ablation: Direct Torque Cont. - High Freq. (500Hz)	Neuroevolution (1M ρ steps)	Direct Torque	**224±2.01**	**135±0.509**	**117±1.05**	**159±1.2**	**118±0.891**	**102±0.323**	**100±1.08e-06**	**344±0.985**	73.1±0.509	**104±0.17**	**189±3.02**
Ablation: Direct Torque Cont. - Low Freq. (31.25Hz)	Neuroevolution (1M ρ steps)	Direct Torque	**198±3.12**	**119±0.556**	**118±0.349**	**168±0.497**	**115±0.863**	**100±0.0525**	**100±1.08e-06**	**305±2.54**	70.2±0.267	**105±0.113**	**184±2.28**
Ablation: Fixed Cont. - PD Position	Neuroevolution (EvoControl ll steps)	PD Position	**220±0.748**	52.5±0.881	**104±1.1**	**158±0.395**	**119±0.158**	**101±0.0186**	**100±1.08e-06**	**390±1.33**	69.1±0.37	86.7±0.179	**152±1.38**
Ablation: Fixed Cont. - PD Position Delta	Neuroevolution (EvoControl neuro. steps)	PD Position Delta	36.2±0.172	4.48±0.00862	**133±0.535**	**143±0.337**	4.6±0.00358	97.7±0.782	**100±0.0594**	**279±3.12**	52±0.174	12.2±0.83	**122±0.452**
Ablation: Fixed Cont. - PD Int. Velocity	Neuroevolution (EvoControl neuro. steps)	PD Int. Velocity	82.7±1.97	67.6±0.85	96±0.941	**172±0.689**	**118±0.124**	97.9±1.22	55.2±1.81	**256±1.19**	61.2±0.179	59.7±0.869	**140±2.56**
Ablation: Fixed Cont. - Random	Random	Direct Torque	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Ablation: Direct Torque Cont. - High Freq. (500Hz)	Neuroevolution (EvoControl neuro. steps)	Direct Torque	**288±3.08**	**135±0.509**	**117±1.05**	**156±1.02**	**116±0.851**	**102±0.323**	**100±1.08e-06**	**344±0.985**	73.1±0.509	**104±0.17**	**169±2.98**
Ablation: Direct Torque Cont. - Low Freq. (31.25Hz)	Neuroevolution (EvoControl neuro. steps)	Direct Torque	**223±1.56**	**119±0.556**	**118±0.349**	**158±0.574**	**119±0.314**	**100±0.0525**	**100±1.08e-06**	**305±2.54**	70.2±0.267	**105±0.113**	**180±2.17**

Second, within EvoControl we perform the ablation of training the high-level policy with Neuroevolution instead of PPO, using the same high-level policy architecture as used in the main paper. Again, we provide two versions, A) tha of training the A) that of training the high-level policy with the same number of high-level ρ steps as the core baselines presented in the paper, with 1M ρ steps—labelled *Neuroevolution (1M ρ steps)*. Specifically, this uses $es\_rollouts = 1, K = 8, es\_pop\_size = 256, es\_sub\_generations = 8$. Second, B) training the high-level policy with the same number of environment steps as was used to train the low-level policy within EvoControl—labelled *Neuroevolution (same as EvoControl low-level)*. Specifically this uses more low-level environment steps than A), with $es\_rollouts = 16, K = 8, es\_pop\_size = 512, es\_sub\_generations = 8$. The results are tabulated alongside the original main table of results for ease of comparison in Table 27. Crucially we observe that on average EvoControl using PPO to train the high-level outperforms (on average achieves a higher normalized evaluation return $\mathcal{R}$) training the high-level with Neuroevolution, confirming the main framework and EvoControl method presented in the paper, and the advantages of the unique combination of a high-level PPO learned policy with a neuroevolved low-level policy. Furthermore, we also observe that on average these variations of EvoControl outperform the respective non-EvoControl baselines of *fixed-controllers* and *direct torque control*.

Table 27: Ablation. Training the high-level policy with Neuroevolution instead of PPO for EvoControl. Normalized evaluation return $\mathcal{R}$ for the benchmark methods, across each environment. EvoControl on average achieves a higher normalized evaluation return than the baselines of *fixed controllers* and *direct torque control*. Results are averaged over 384 random seeds, with $\pm$ indicating 95% confidence intervals. Returns are normalized to a 0-100 scale, where 0 represents a random policy, and 100 represents the highest reward achieved by a non-EvoControl baseline in each environment. Scores bolded are greater than 100.

Method Name	High-level ρ with	Low-level β with	Ant $\mathcal{R}\uparrow$	Halfcheetah $\mathcal{R}\uparrow$	Hopper $\mathcal{R}\uparrow$	Humanoid $\mathcal{R}\uparrow$	Humanoid Standup $\mathcal{R}\uparrow$	Inverted Double Pendulum $\mathcal{R}\uparrow$	Inverted Pendulum $\mathcal{R}\uparrow$	Pusher $\mathcal{R}\uparrow$	Reacher $\mathcal{R}\uparrow$	Reacher 1D $\mathcal{R}\uparrow$	Walker2D $\mathcal{R}\uparrow$
Fixed Cont. - PD Position	PPO	PD Position	100±6.25	63.2±0.599	89.3±1.2	92.6±2.05	100±0.208	99.9±0.03	100±1.53e-06	100±8.47	100±1.77	85±2.93	81.7±0.384
Fixed Cont. - PD Position Delta	PPO	PD Position Delta	2.57±1.74	2.86±0.092	100±1.84	100±1.78	2.54±0.0542	53.8±1.57	100±1.53e-06	0±0	40.8±3.23	15.2±7.6	96.3±0.219
Fixed Cont. - PD Int. Velocity	PPO	PD Int. Velocity	2.46±1.51	2.55±0.0967	73.1±0.844	86.6±1.33	0±0	49.7±1.55	86.5±2	0±0	0±0	0±0	93.7±2.77
Fixed Cont. - Random	Random	Direct Torque	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Direct Torque Cont. - High Freq. (500Hz)	PPO	Direct Torque	0±0	18±0.353	0.882±0.533	7.93±1.94	10.1±0.357	0±0	0±0	4.36±7.75	0±0	45.3±6.74	0±0
Direct Torque Cont. - Low Freq. (31.25Hz)	PPO	Direct Torque	63±7.07	100±1.54	70.6±0.548	85.6±1.41	80±1.73	100±0.0311	100±1.53e-06	75.4±12.8	58.8±3.74	100±1.94	100±1.06
EvoControl (Full State)	PPO	Neuroevolution	**368±73.2**	**157±18.3**	**274±20.6**	**123±19**	**116±18.4**	**101±0.859**	**100±0**	**362±12.5**	**114±7.51**	**106±2.75**	**203±137**
EvoControl (Residual State)	PPO	Neuroevolution	**182±16.1**	**182±6.31**	**101±5.54**	**170±18.2**	**212±145**	99.2±1.25	**100±0**	**375±94.8**	**106±25.8**	**104±3.42**	**205±57.4**
EvoControl (Target + Proprio.)	PPO	Neuroevolution	**319±35.1**	**168±14.7**	**171±155**	**165±7.19**	**165±150**	99.7±1.19	**100±0**	**353±28.4**	96.8±78.6	**105±4.71**	**178±63.7**
EvoControl (Target)	PPO	Neuroevolution	**293±87**	**162±23.5**	**283±250**	**164±21.2**	**205±147**	99.6±0.949	**100±0**	**353±43.4**	**112±1.73**	**105±1.65**	**188±88.2**
EvoControl (Learned Gains)	PPO	Neuroevolution	**266±104**	**113±10.5**	**206±302**	**150±15.1**	**117±2.44**	99.5±2.48	**100±0**	**330±17.8**	**116±1.62**	**105±2.45**	**196±118**
EvoControl (Delta Position)	PPO	Neuroevolution	**362±47.7**	**133±34.6**	**225±82.9**	**119±18**	**105±4.64**	**101±0.394**	**100±0**	**267±30.2**	65.5±21	99.1±12.7	**183±34.4**
Ablation: EvoControl (Full State)	Neuroevolution (1M ρ steps)	Neuroevolution	**151±42.9**	**134±6.93**	99.6±26.5	**115±33.3**	**110±18.1**	99.6±1.66	100±0	**320±84.2**	55.4±16.4	42.2±56.3	**144±72.4**
Ablation: EvoControl (Residual State)	Neuroevolution (1M ρ steps)	Neuroevolution	**136±135**	99.4±21.3	90.9±2.96	**138±13**	**115±3.68**	67.7±38.9	17.5±66.1	**292±22.3**	52.3±11.5	2.98±2.12	**117±34.9**
Ablation: EvoControl (Target + Proprio.)	Neuroevolution (1M ρ steps)	Neuroevolution	**130±13.9**	**151±32.5**	80.4±47.6	**121±20.7**	**107±22.8**	44±52.2	23±48.6	**296±74.9**	53.9±12.4	7.38±17.6	**126±24.4**
Ablation: EvoControl (Target)	Neuroevolution (1M ρ steps)	Neuroevolution	**135±7.57**	**101±19**	75.6±222	**123±14.3**	**103±22**	65.4±40.5	51.1±42.4	**342±46.1**	52.3±9.81	26±40.1	**130±42.2**
Ablation: EvoControl (Learned Gains)	Neuroevolution (1M ρ steps)	Neuroevolution	**174±38.9**	89±60.3	67.2±160	**111±23.9**	**113±2.31**	46.7±23.3	76.4±101	**252±101**	53.8±8.36	26.2±29.4	**113±18.4**
Ablation: EvoControl (Delta Position)	Neuroevolution (1M ρ steps)	Neuroevolution	**156±15.7**	**130±9.55**	**132±261**	**119±17.6**	**107±17.1**	90.6±40.5	100±0	**322±12.6**	54.9±11.8	31.9±72.6	**112±98.9**
Ablation: EvoControl (Full State)	Neuroevolution (EvoControl ll steps)	Neuroevolution	**176±6.59**	**143±12.4**	94.4±15.6	**123±24.3**	**108±13.4**	95.8±9.58	100±0	**334±52.9**	55.4±16.4	15.7±28.3	**153±43.5**
Ablation: EvoControl (Residual State)	Neuroevolution (EvoControl ll steps)	Neuroevolution	**169±51.8**	**154±41.9**	91.1±8.64	**131±44.1**	**142±58**	42.9±69.4	85.8±59.4	**300±50.1**	52.3±10.1	4.56±8.08	**142±21.8**
Ablation: EvoControl (Target + Proprio.)	Neuroevolution (EvoControl ll steps)	Neuroevolution	**153±13**	**137±19.3**	94.6±0.0	**122±26.4**	**102±23.2**	62.7±18.8	43.8±65.7	**291±16.4**	53.9±12.3	27.3±52.4	**122±26**
Ablation: EvoControl (Target)	Neuroevolution (EvoControl ll steps)	Neuroevolution	**152±61**	**131±109**	63.5±77.1	**120±22.9**	99.6±25.1	48±24.7	20.4±15	**313±74.8**	54.1±14.7	18.8±25.4	**116±11.3**
Ablation: EvoControl (Learned Gains)	Neuroevolution (EvoControl ll steps)	Neuroevolution	**145±166**	90.5±35.2	13.9±80.3	**126±12.6**	**107±21.8**	71.8±18.3	97±12.8	**237±99.7**	53.8±13.9	45.9±19.8	76.1±155
Ablation: EvoControl (Delta Position)	Neuroevolution (EvoControl ll steps)	Neuroevolution	**168±60**	**140±13.5**	**147±27**	**118±8.28**	**107±8.46**	95.8±16.6	100±0	**315±103**	55.6±13.2	50±83.2	**172±67.5**

J.9 Rollout Trajectory Plots of High-level Action for Baselines

In the following we plot rollout trajectories, including the high-level latent action a_k over one evaluation episode in the Reacher 1D environment for all baselines. We take the trained policies, as trained in the main table of results in the paper (Table 3), and evaluate them for one random seed to produce the rollout trajectory plot. To facilitate simpler comparison, we use the same random seed across the baseline trajectory plots. Moreover, we use the Reacher 1D environment as it is straightforward to plot and understand what an optimal policy π should do. In this case the initial reacher arm starts in a random position (as defined on a circle $q_0 \in (-\pi, \pi)$), with a random velocity, and its goal is to move the arm to a randomly sampled goal location (goal $q_{\text{goal}} \in (-\pi, \pi)$)—in the trajectory plots we plot the goal location with the red line, which is constant throughout the episode. Therefore an optimal policy is one that moves the arm to the goal location, quickly and keeps it there to maximize reward, where the reward is defined as $-||\text{xy\_tip\_of\_arm}(q_0) - \text{xy\_tip\_of\_arm}(q_{\text{goal}})||_2^2$. We provide the plots for each respective baseline in Figures 13 to 23. We observe that the EvoControl variants on average consistently outperform the other non-EvoControl baselines and achieve a higher evaluation return R—which is also given on each plot.

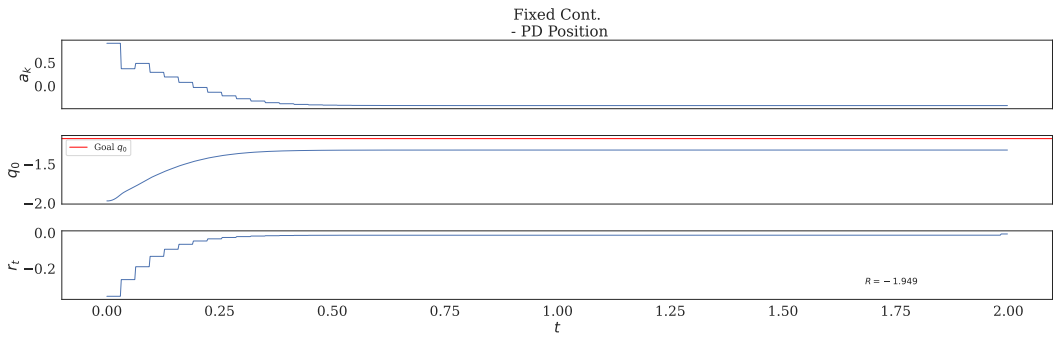

Figure 13: Evaluation Trajectory Rollout for Reacher 1D, for baseline *Fixed Cont.* - PD Position. Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

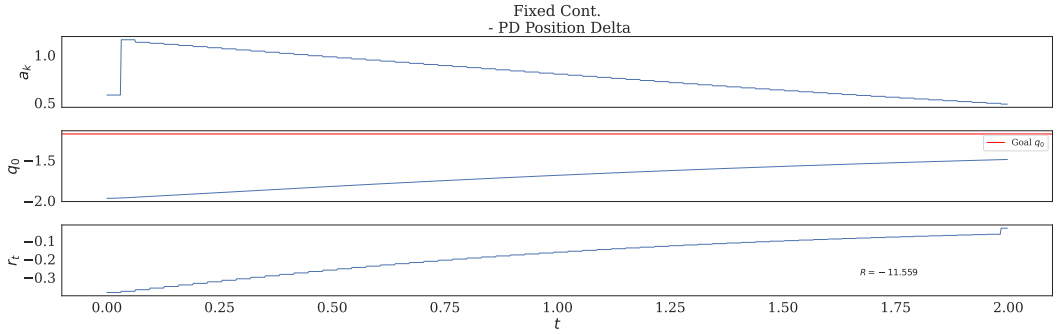

Figure 14: Evaluation Trajectory Rollout for Reacher 1D, for baseline *Fixed Cont.* - PD Position Delta. Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

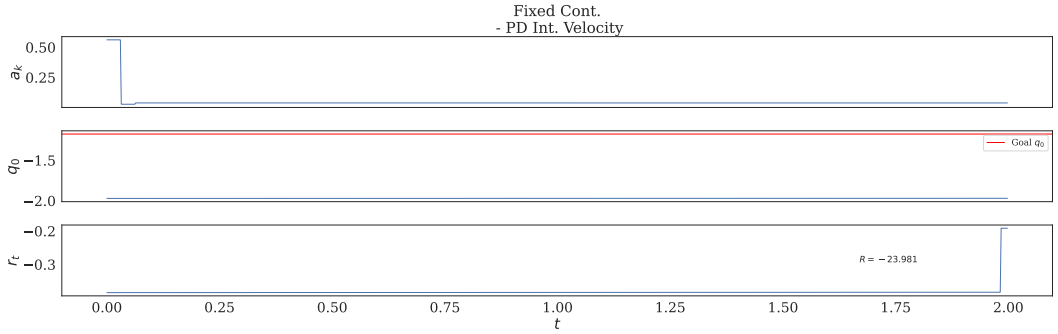

Figure 15: Evaluation Trajectory Rollout for Reacher 1D, for baseline *Fixed Cont.* - PD Int. Velocity. Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

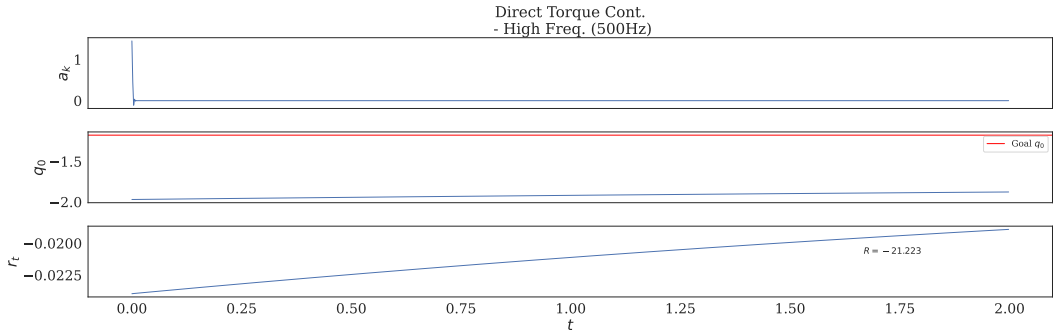

Figure 16: Evaluation Trajectory Rollout for Reacher 1D, for baseline *Direct Torque Cont.* - High Freq. (500Hz). Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

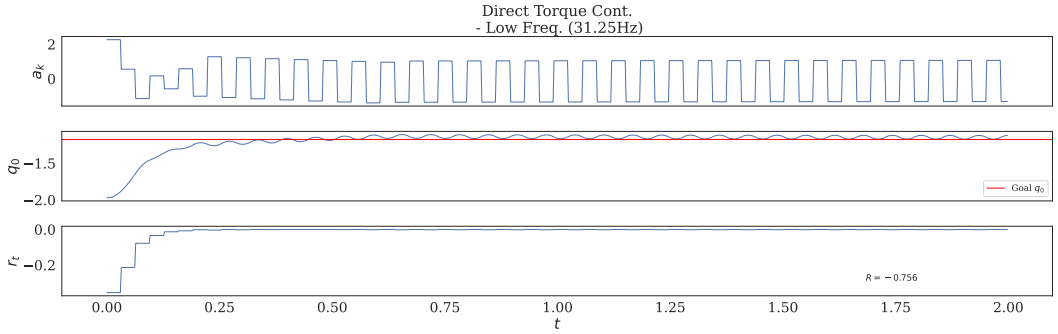

Figure 17: Evaluation Trajectory Rollout for Reacher 1D, for baseline *Direct Torque Cont.* - Low Freq. (31.25Hz). Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

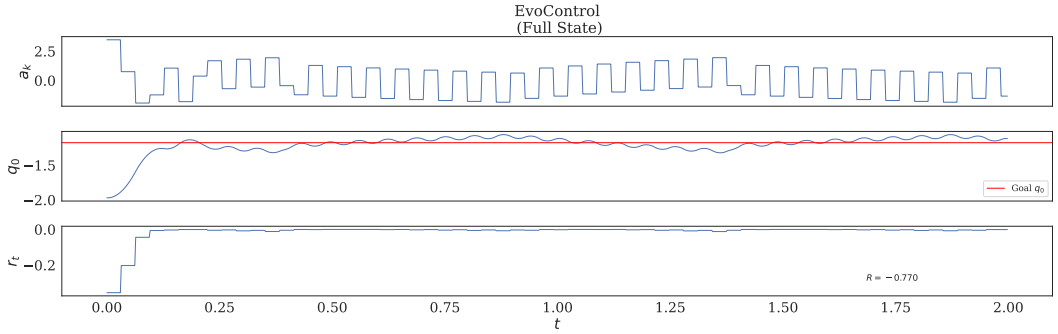

Figure 18: Evaluation Trajectory Rollout for Reacher 1D, for baseline *EvoControl* (Full State). Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

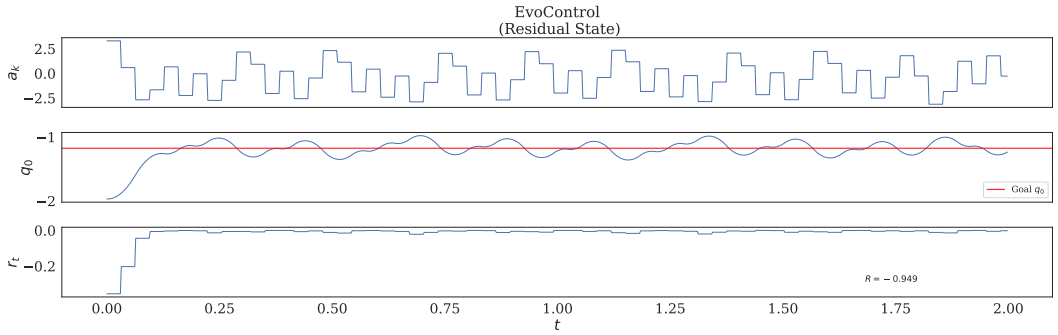

Figure 19: Evaluation Trajectory Rollout for Reacher 1D, for baseline *EvoControl* (Residual State). Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

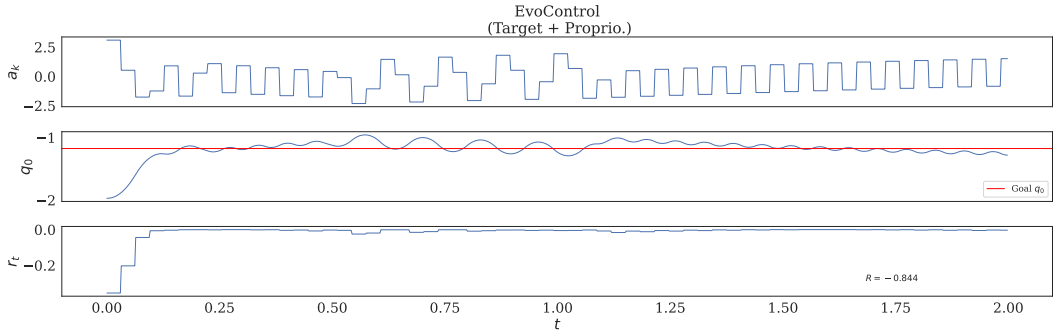

Figure 20: Evaluation Trajectory Rollout for Reacher 1D, for baseline *EvoControl* (Target + Proprio.). Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

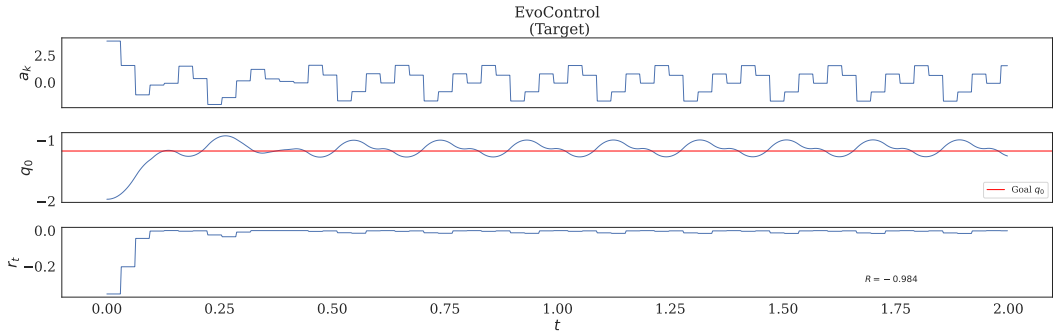

Figure 21: Evaluation Trajectory Rollout for Reacher 1D, for baseline *EvoControl* (Target). Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

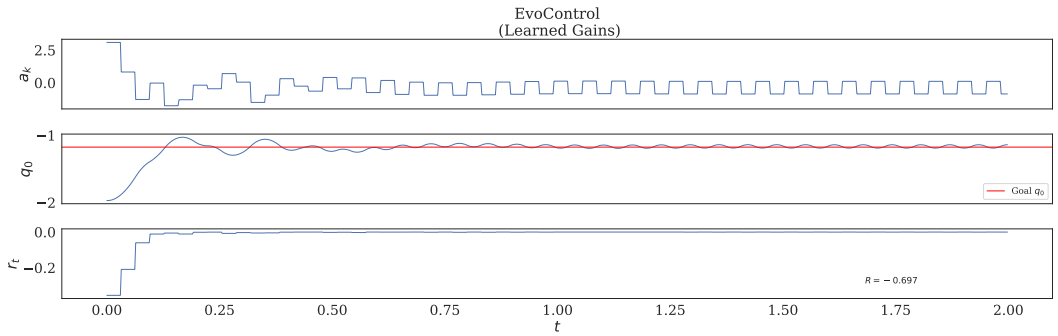

Figure 22: Evaluation Trajectory Rollout for Reacher 1D, for baseline *EvoControl* (Learned Gains). Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

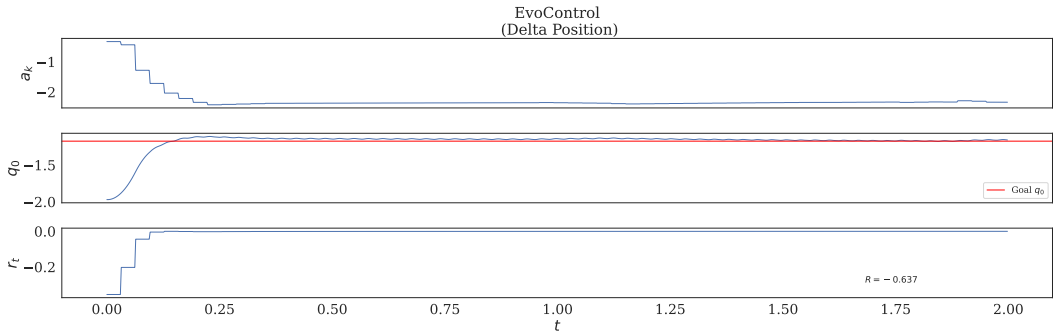

Figure 23: Evaluation Trajectory Rollout for Reacher 1D, for baseline *EvoControl* (Delta Position). Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

J.10 Ablation: Removing Communication Between the Layers in EvoControl

We performed a further ablation, which tests the hypothesis of if within EvoControl, whether there is useful communication between the high-level and low-level policy. Specifically for the EvoControl variants that we considered, in some variations the low-level policy receives only a restricted observation (just the joint positions of the robot, and not the random goal location if one exists), compared to receiving the full observation (which includes any random goal location if one exists for that environment).

Specifically, we consider two variants of EvoControl, where the low-level policy receives the full observation (*EvoControl - (Full State)*), and where the low-level policy only receives a restricted observation without any goal location—necessitating effective communication from the high-level to the low-level (*EvoControl (Target)*). We compare these on the Reacher 1D task, visualizing the rollouts for a random high-level policy, and a zero high-level policy, as observed in Figures 24 and 25. We observe that for both variations after the standard EvoControl training, removing the communication (by making the high-level policy either a random policy or a null policy) significantly reduces the performance (return), and leads to an unstable low-level policy, even when the low-level policy receives the full observation (s_t). Intuitively, it could be the case that the low-level policy learns a form of a PD controller, and the high-level policy, as trained initially with a PD position controller, could converge to treat the low-level policy as a form of PD controller.

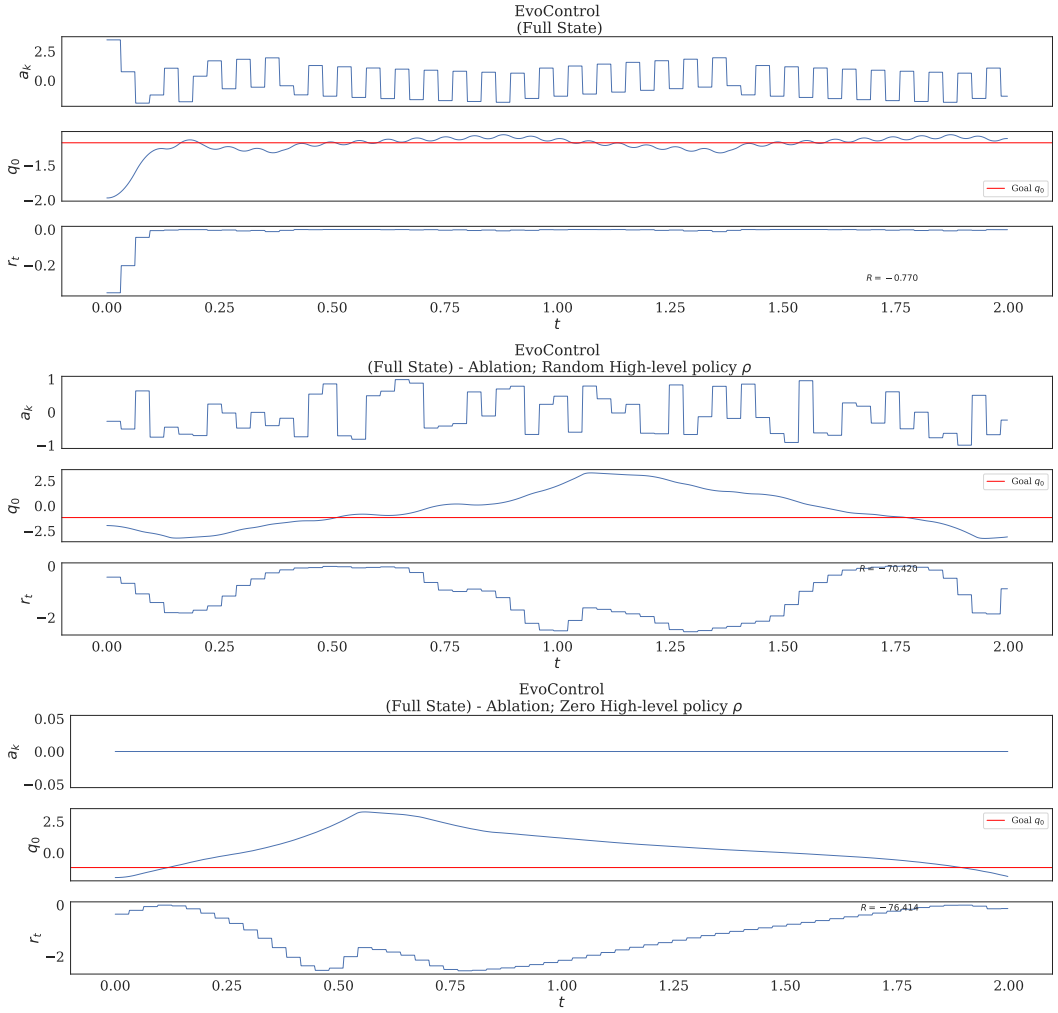

Figure 24: Evaluation Trajectory Rollout for Reacher 1D, for baseline *EvoControl* (Full State), with ablation of a random high-level and a null (zero action) policy. Here the observation for the low-level policy is $s_t, a_k, e_t, q_t, \dot{q}_t, t/T$. We observe that even with the low-level policy receiving the full observation it still relies on the communication from the high-level latent action a_k, and without it, the return significantly reduces. Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

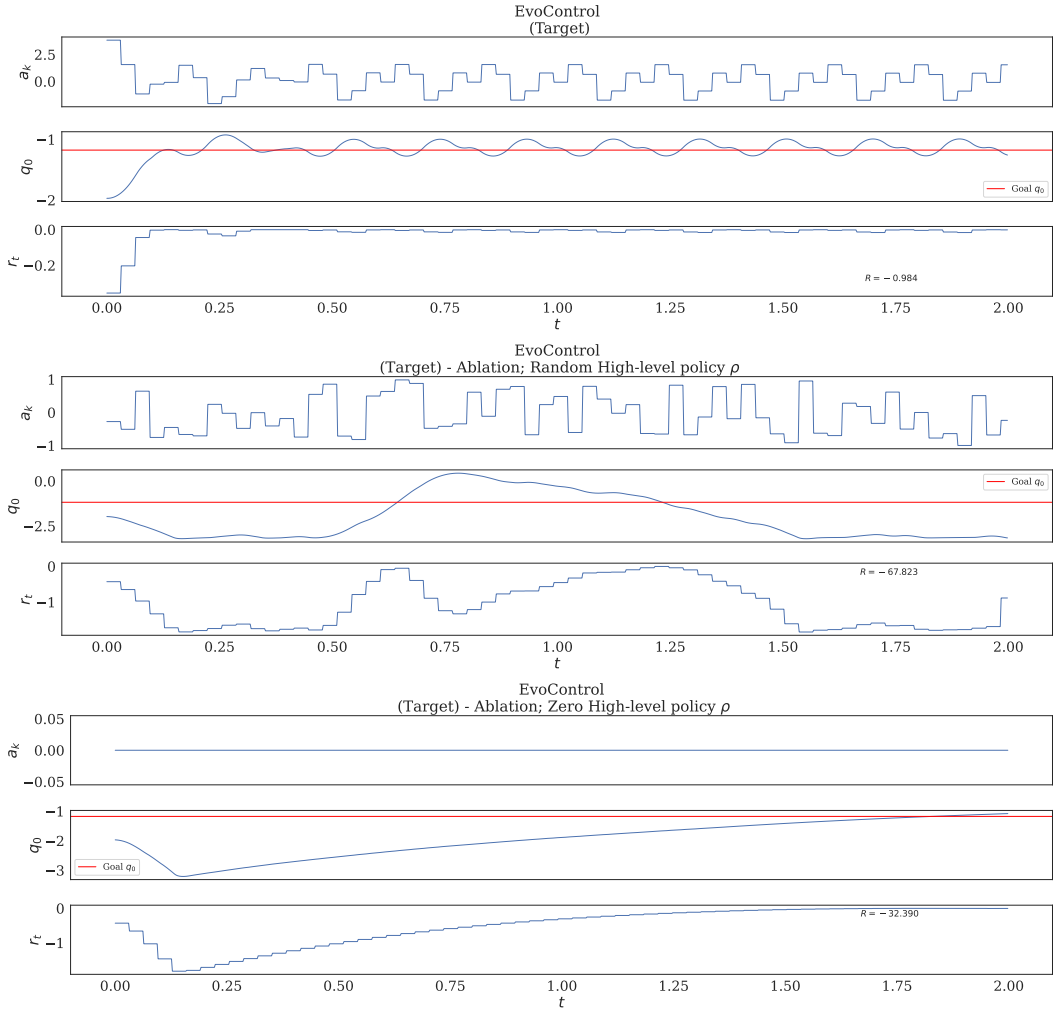

Figure 25: Evaluation Trajectory Rollout for Reacher 1D, for baseline *EvoControl* (Target), with ablation of a random high-level and a null (zero action) policy. Here the observation for the low-level policy is $a_k, q_t, \dot{q}_t, t/T$. We observe that the low-level policy relies on the communication from the high-level latent action a_k, and without it, the return significantly reduces. Environment runs at 500Hz, with an 1,000 low-level environment steps, corresponding to a episode duration of 2.0 seconds. Where a_k is the high-level policy ρ latent action, q_0 is the reacher arm's angle in radians, with the red-line indicating the random goal q_{goal} for the episode, r_t the instantaneous reward and R the total return for the episode.

K Limitations & Future Work

While EvoControl demonstrates promising results for high-frequency continuous control, several limitations and avenues for future research warrant exploration.

- Still relies on the existence of a fixed-PD controller for the continuous-time control task. Although we demonstrate robustness to some degree of PD parameter misspecification (Table 5), the reliance on a PD controller as a starting point poses a limitation. In domains where designing a suitable PD controller is challenging or impossible (e.g., systems with non-actuated joints, highly nonlinear dynamics, or discrete action spaces), applying EvoControl in its current form may be difficult. We do perform an ablation where we show that EvoControl can still learn performant policies without the existence of a fixed-PD controller Appendix J.4, however other approaches to stabilize and initialize policy learning are promising directions for future work.
- EvoControl can require more computational complexity compared to only performing PPO, which can be readily parallelized in practice with modern accelerated compute platforms, and restricting EvoControl to use the same computational complexity, whilst still outperforming the baselines is also possible, Appendix J.3.

In addition, promising future directions include exploring more complex nested hierarchies, direct low-level to high-level information flow, and ensembles of policies.

L Reproducibility Statement

In the following we outline all the sections where the reader can find full information to fully reproduce all the main results. We also clearly state the following of the *assumptions* of the method in Appendix G, *experimental settings* in Appendices E to I, and the *limitations* of the work in Appendix K.

M Ethics Statement

In this paper we present *EvoControl*, a novel bi-level policy learning framework for learning both a slow high-level policy and a fast low-level controller using PPO and Neuroevolution, respectively, for continuous-control tasks, to control continuous control tasks, such as robotic control. However misuse of such a policy learning framework could occur when training on a miss-specified task, and or the behavior of the policy should always be evaluated, checked with a human expert, and appropriate safety controls put in place to avoid any unintended behavior.

N Common Questions and Discussion

N.1 Why Not Simply Pre-train with a PD Controller and Imitate?

One might consider using a PD position controller as a policy to collect rollout trajectories and then using imitation learning to train a neural network to replicate its behavior. However, this approach has limitations. The learned network would only be as capable as the PD controller itself, inheriting its limitations in expressiveness and inability to learn complex, high-frequency interaction behaviors. EvoControl, by directly optimizing the low-level policy with Neuroevolution, aims to surpass the capabilities of the initial PD controller and discover more nuanced and adaptive control strategies. Furthermore, imitation learning requires a substantial amount of demonstration data, while EvoControl learns directly from the environment reward signal.

N.2 Relationship to Pulse-Width Modulation

The benefits of high-frequency control, as highlighted by Proposition 2.1, share a conceptual similarity with Pulse-Width Modulation (PWM) in electrical engineering. In PWM, a high-frequency signal with varying pulse widths is used to effectively represent a lower-frequency analog signal. Similarly, in EvoControl, high-frequency actions generated by the low-level policy can represent and achieve the lower-frequency targets set by the high-level policy with greater precision and responsiveness compared to a fixed-frequency PD controller. While not a direct analogy, this parallel highlights the ability of high-frequency signals to enhance control and achieve desired outcomes more effectively.