CO-DESIGN GYM: A UNIFIED BENCHMARK FOR EMBODIMENT-POLICY CO-OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Finding an optimal behaviour policy within a given environment is a widely studied problem in domains as diverse as games, robotics, energy infrastructure, communication networks, and multi-agent systems. Numerous benchmarks have been developed to support such research, but the vast majority assume that the agent's embodiment (design) is fixed, focusing instead on policy learning alone. Lifting this assumption gives rise to a broader class of problems in which optimizing embodiment and policy separately is highly suboptimal. An agent's embodiment strongly shapes which control policies can be discovered, while the optimal embodiment is in turn defined by the policies it admits. To help the research community study this class of problems explicitly and systematically, we introduce CO-DESIGN GYM – a suite of benchmark environments for jointly optimizing embodiment and policy. Our environments span domains such as robotic manipulation and locomotion, multi-robot cooperation, deformable and soft dynamics, video games, electricity grids, wireless networks, F1 racing, multi-agent warehouses, and optimal control, offering 20 environment families (domains), with over 85 distinct co-design presets in total. We further contribute a systematic evaluation of representative co-design algorithms, characterizing the current state of the art. Together, these contributions lay the groundwork for cumulative, comparable progress in co-design.

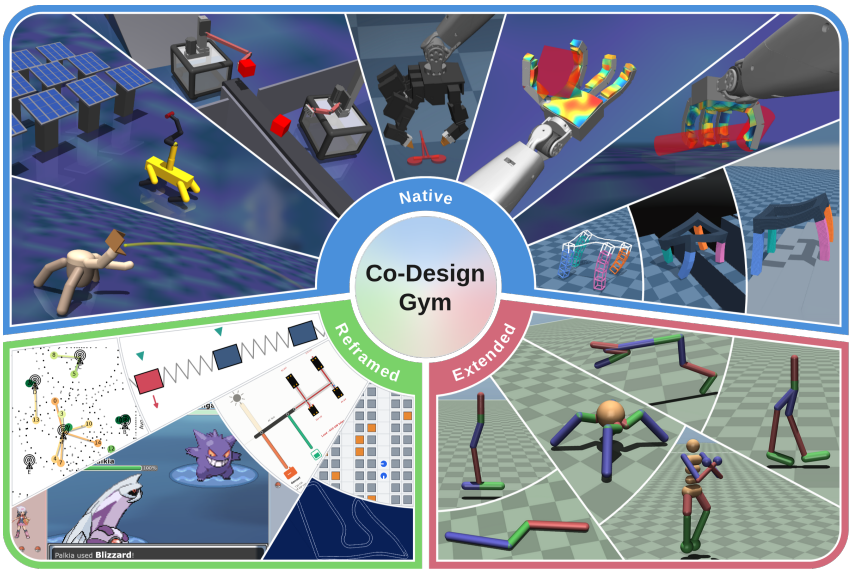

Figure 1: Co-Design Gym overview with three environment categories. **Native:** new domains we developed to study the dynamic interplay between embodiment and policy, covering manipulation, locomotion and deformable body dynamics. **Extended:** classic MuJoCo control tasks from OpenAI Gym extended to support morphology variations. **Reframed:** existing problems from games, energy, networking, racing, logistics and control, recast as co-design tasks. *Co-Design Gym is available at https://anonymous.4open.science/r/codesign-gym-C695/README.md*

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

An agent's ability is a function of both its *embodiment* and its behavioral *policy*. An embodiment determines what an agent is *able to do*: it is an interface for transforming perception into action upon an external environment. The policy determines what the agent *actually does*, using a given physical interface. The process of jointly improving the embodiment and the policy toward performance goals is referred to as *co-design*.

Embodiment and policy are coupled: the quality of an embodiment is the performance of the best policy it can achieve for downstream tasks. Hence, fully assessing each candidate embodiment requires finding an optimal policy for it on a variety of tasks. However, running expensive policy optimization to completion before knowing the potential of an embodiment is wasteful. Tighter co-optimization is essential, especially when the embodiment and policy subspace structures yield challenging high-dimensional optimization landscapes.

The above calls for extending optimization paradigms such as Reinforcement Learning (RL), which are usually framed as singly non-stationary problems. The broader class of *doubly non-stationary* problems is more suitable, since both the policy and the embodiment can change rapidly. Finding global or even high-quality local optima for this extended problem is expensive, and gauging real progress is nontrivial. To date, no unified benchmark tracks the performance of co-design methods across a variety of domains, or disentangles progress in embodiment search and policy optimization from progress specific to co-design.

Co-design today stands in a similar place to where RL stood before the introduction of unified benchmarks, when agents still struggled to solve basic tasks (Mnih et al., 2015). Benchmarks and unifying frameworks, *e.g.* OpenAI Gym and Gymnasium (Brockman et al., 2016; Towers et al., 2024), enabled compound progress that accelerated the field. RL agents have since reached superhuman performance in a wide variety of areas, such as strategic game playing in Go (Silver et al., 2016) & Dota 2 (Berner et al., 2019), drone racing (Geles et al., 2026), control of tokamak plasmas for nuclear fusion (Degrave et al., 2022), and robot locomotion (Peng et al., 2017; 2018; Margolis et al., 2021), to name a few. A unified benchmark for co-design should spark comparable progress.

In pursuit of that, we introduce CO-DESIGN GYM – a unified benchmark for co-design algorithms. Our contribution is threefold. First, we extend the canonical Gymnasium API with an explicit design space, so that any policy-learning task can be lifted into a co-design task. Second, we leverage this generalized API to convert prominent existing policy optimization environments into co-design tasks. Finally, we present novel co-design environments in which embodiment optimization is decisive for what the policy can achieve: from reshaping fingertips for precise manipulation of small objects & reshaping a hand for fast in-hand reorientation, to dynamic ball catching, multi-robot truck unloading, and evolving soft robots for locomotion. Overall, we offer 20 environment families and over 85 presets, with accelerated GPU-based resets that make it practical to evaluate thousands of embodiments in parallel. Benchmarking representative evolutionary, reinforcement-learning, hybrid and transformer-based co-design methods under matched compute budgets, we find that no single method dominates, and that many tasks remain far from solved. This leaves substantial headroom for new methods, and we call upon the community to take advantage of this opportunity.

2 BACKGROUND AND RELATED WORK

Co-design problems appear naturally across disciplines in design tasks that feature both a static embodiment and a time-varying policy component, and in which design choices made in each domain can affect design choices in the other. Over the last several decades, methods have been developed for the co-design of computer systems hardware and software (De Michell & Gupta, 1997; Milder et al., 2012), optics and their processing algorithms (Dowski Jr & Cathey, 1995), aeronautical materials and stabilizers (Salama et al., 1989), and chemical plants and processes (Luyben & Floudas, 1994), just to name a few. More recently, co-design of the embodiments and policies of virtual and physical robots has seen growing interest. This problem is exceptionally relevant now that investment in robotic hardware is at all-time high: advanced hardware prototyping is finally affordable, thus co-design solutions don't have to be confined to simulation. Robots typically must solve tasks that involve a complex combination of dynamical moves, medium-to-long-term-planning, a mix of discrete and continuous optimization in both morphological structure and actions all at once, while

still enabling fast (real-time) operation of the resulting system. A further challenge is to transfer the success from simulation to reality. All these aspects make co-optimization of robot systems particularly challenging, yielding an ideal ground for testing novel co-design methods.

Early research built upon the seminal works of Ventrella (1994) and Sims (1994) and employed evolutionary algorithms (Hiller & Lipson, 2011; Cheney et al., 2014b;a; Corucci et al., 2016), a trend that still sees application in modern learning-based approaches (Wang et al., 2019; Yuan et al., 2021) and are enjoying a renaissance in combination with off-the-shelf large models (Chen et al., 2026; Gao et al., 2026). While these approaches produce robust specimens (Bongard, 2011; Hejna III et al., 2021) their sampling-based search can be inefficient when simulation time becomes the computational bottleneck. In such cases, complementary approaches that use gradient-based search tend to more efficient (Matthews et al., 2023). Given continuous and smoothly differentiable parameterizations of morphology and controllers, these approaches employ unconstrained and constrained optimizers to search over optimal parameterizations of geometry (Wampler & Popović, 2009; Coros et al., 2013; Spielberg et al., 2017; Ha et al., 2017; Ma et al., 2021), materials (Spielberg et al., 2023), and controllers and observers (Spielberg et al., 2019) for solving specified tasks. As the field has begun to mature in the last decade, holistic frameworks have emerged that have driven toward a unifying theory of co-design (Censi, 2015; Zardini, 2023). Sampling- and gradient-based approaches have also begun to merge under the umbrella of generative systems (Wang et al., 2023b; Zhao et al., 2020; Xu et al., 2021; Dai et al., 2026; Ha et al., 2026).

Though much of the above work focuses on the algorithmic nature of co-design algorithms, co-designed creatures are not confined to the virtual world. Successful physical demonstrations of co-designed robots reach back as far as Park & Asada (1994) and have been applied to form factors such as quadrupeds (Wang et al., 2026a), grippers (Xu et al., 2024; Fay et al., 2025), full humanoids (Wang et al., 2026b), and more at large (Yu et al., 2026) and small scales (Kriegman et al., 2020; Balciunaite et al., 2026).

Despite algorithmic progress, evaluation progress has lagged. Two precedents have made some progress in this regard. SoftZoo (Wang et al., 2023a) provides benchmarking for soft robot co-design algorithms across terrains, materiality, and morphological representations, but lacks physical realism and is limited to 3D elastic dynamics. EvoGym (Bhatia et al., 2022) provides an extensive array of tasks and environments and emphasizes complex neural control tasks, but it is also limited to soft robots, and in this case, 2D, low-fidelity pixel representations. Further evaluation setups relevant to co-design exist, but either focus on a narrow set of tasks, or lack realistic environment modeling. Our work is most inspired by the OpenAI Gym (Brockman et al., 2016) and its successor, Gymnasium (Towers et al., 2024), which have been seminal for RL research under a shared framework, but lack the extensibility needed to accommodate search over morphologies and other embodiment aspects.

3 A Formal Treatment of Co-Design

Co-design views the choice of embodiment and policy as co-dependent decision problems. Environment rules, as well as task requirements and objectives/utilities/rewards can also be included in the joint co-design optimization space, but here we focus on core embodiment-policy interactions.

In general, choosing an embodiment $\mathbf{e}$ from a set $\mathbf{E}$ refers to setting agent's capabilities, subject to overarching rules of an environment. Those rules can be tied to physics, but can also come from completely different aspects, e.g. the logic of virtual characters or software (not physical laws).

We formally define the co-design problem via Definitions 1 & 2 below, obtained from the standard Reinforcement Learning (RL) formulation by removing the assumption of a fixed embodiment. The intent is to draw on the richness and strengths of RL formalism and theory, but formulate an optimization problem that is well-suited for co-design.

Definition 1 (Environment)**.** Formally, an *environment* can be expressed as a tuple $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \rho_0, T, \gamma)$ with state space $\mathcal{S}$, action space $\mathcal{A}$, initial distribution $\rho_0 \in \Delta(\mathcal{S})$, transition function $T : \mathcal{S} \times \mathcal{A} \times \mathbf{E} \to \Delta(\mathcal{S})$, and discount $\gamma \in [0, 1)$.

Definition 2 (Co-design Problem)**.** A *co-design problem* is a 4-tuple $\mathcal{I} = (\mathbf{E}, \mathbf{\Pi}, \mathcal{E}, R) \in \mathbf{I}$, where $\mathbf{E}$ is a set of admissible embodiments, $\mathbf{\Pi}$ is a policy class, $\mathcal{E}$ is an environment over $\mathbf{E}$, and $R : \mathcal{S} \times \mathcal{A} \times \mathbf{E} \to \mathbb{R}$ is a reward function. Each $\mathbf{e} \in \mathbf{E}$ admits the decision problem $M_\mathbf{e} \in \mathbf{M}$, with control policies $\Pi_\mathbf{e} \subseteq \mathbf{\Pi}$ and return $J(\mathbf{e}, \pi)$. A *solution* is a pair $(\mathbf{e}, \pi)$ with $\mathbf{e} \in \mathbf{E}$ and $\pi \in \Pi_\mathbf{e}$,

with a value $J(\mathbf{e}, \pi)$. Two problems I_α and I_β are equivalent if and only if $\mathbf{E}_\alpha = \mathbf{E}_\beta$, $\mathbf{\Pi}_\alpha = \mathbf{\Pi}_\beta$, $\mathcal{E}_\alpha = \mathcal{E}_\beta$, and $R_\alpha = R_\beta$.

We call the set of all solutions of $\mathcal{I}$ its *solution set*: $\mathrm{Sol}(\mathcal{I}) = \{(\mathbf{e}, \pi) : \mathbf{e} \in \mathbf{E},\ \pi \in \Pi_\mathbf{e}\} \subseteq \mathbf{E} \times \mathbf{\Pi}$, and write:

$$J^\star(\mathbf{e}) = \sup_{\pi \in \Pi_\mathbf{e}} J(\mathbf{e}, \pi), \qquad J^{\star\star}(\mathcal{I}) = \sup_{\mathbf{e} \in \mathbf{E}} J^\star(\mathbf{e}) = \sup_{(\mathbf{e}, \pi) \in \mathrm{Sol}(\mathcal{I})} J(\mathbf{e}, \pi).$$

Throughout, $M_\mathbf{e}$ is the decision problem obtained by fixing the embodiment argument of T and R to $\mathbf{e}$, $J(\mathbf{e}, \pi)$ is the expected discounted return of π in $M_\mathbf{e}$, and $\Pi_\mathbf{e} = \mathbf{\Pi} \cap \Pi(\mathbf{e})$, where $\Pi(\mathbf{e})$ is the set of policies that embodiment $\mathbf{e}$ can execute.

Any problem typically encompasses many subproblems, which restricts a part of the problem to an eligible subset. In the case of co-design, this can be a restriction on the design space or policies to an eligible subset. We formally define subproblems below. Since they exist within larger problems, solving any subproblem incurs lower or equal return than solving its corresponding problem, which we show in Proposition 4.

Definition 3 (Subproblem)**.** For $\mathbf{E}' \subseteq \mathbf{E}$, let the restriction $R|_{\mathbf{E}'}$ be R with its embodiment argument limited to $\mathbf{E}'$. Let $I_\alpha = (\mathbf{E}_\alpha, \mathbf{\Pi}_\alpha, \mathcal{E}_\alpha, R_\alpha)$ and $I_\beta = (\mathbf{E}_\beta, \mathbf{\Pi}_\beta, \mathcal{E}_\beta, R_\beta)$ be problems. I_α is a subproblem of I_β ($I_\alpha \subseteq I_\beta$) if and only if $\mathbf{E}_\alpha \subseteq \mathbf{E}_\beta$, $\mathbf{\Pi}_\alpha \subseteq \mathbf{\Pi}_\beta$, $\mathcal{E}_\alpha = \mathcal{E}_\beta|_{\mathbf{E}_\alpha}$, and $R_\alpha = R_\beta|_{\mathbf{E}_\alpha}$.

Proposition 4. *Let* $I_\alpha = (\mathbf{E}_\alpha, \mathbf{\Pi}_\alpha, \mathcal{E}_\alpha, R_\alpha)$ *and* $I_\beta = (\mathbf{E}_\beta, \mathbf{\Pi}_\beta, \mathcal{E}_\beta, R_\beta)$ *be problems. If* $I_\alpha \subseteq I_\beta$ *then* $J^{\star\star}(I_\alpha) \leq J^{\star\star}(I_\beta)$.

Proof. By definition, $\mathcal{E}_\alpha$ and R_α agree with $\mathcal{E}_\beta$ and R_β on $\mathbf{E}_\alpha$. Then, each $\mathbf{e} \in \mathbf{E}_\alpha$ has the same decision problem $M_\mathbf{e}$, and hence the same return J, in both I_α and I_β. Since $\mathbf{\Pi}_\alpha \subseteq \mathbf{\Pi}_\beta$, each such $\mathbf{e}$ has $\Pi_{\alpha,\mathbf{e}} = \mathbf{\Pi}_\alpha \cap \Pi(\mathbf{e}) \subseteq \mathbf{\Pi}_\beta \cap \Pi(\mathbf{e}) = \Pi_{\beta,\mathbf{e}}$; together with $\mathbf{E}_\alpha \subseteq \mathbf{E}_\beta$, this gives $\mathrm{Sol}(I_\alpha) \subseteq \mathrm{Sol}(I_\beta)$. A supremum over a subset cannot exceed that over its superset, so $J^{\star\star}(I_\alpha) \leq J^{\star\star}(I_\beta)$. □

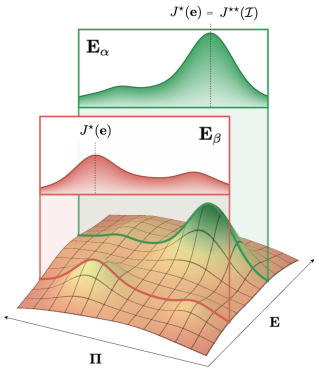

Figure 2: A concept of a joint co-design landscape with policy space Π and embodiment space E. Each is shown as a 1D slice of a 2D space, but in practice they are high-dimensional subspaces. The joint can include other subspaces, e.g. task & interface.

Co-design as a superset of policy optimization. An algorithm that tackles policy optimization usually settles the embodiment design decision a priori. It chooses a single embodiment $\mathbf{e}$, forming a Subproblem as we discussed above. Figure 2 visualizes this concept, demonstrating that policy optimization problems explore one slice of a fundamentally higher-dimensional space. This posits that they live as subproblems in the same space as the broader class of co-design problems.

What follows from this, is that policy optimization is strictly contained within co-design, and in many cases it forms the inner loop of the problem. This containment raises fundamentally interesting formalisms and containment, which we discuss further in Proposition 5 and Appendix F.

While fundamentally a joint problem, embodiment and policy are typically modelled as distinct subspaces. This is useful for creating optimizers with different properties for these subspaces, incorporating structural priors, or enabling hierarchy. For instance, sampling from the policy subspace (e.g. realtime action) may be a lot faster than from the embodiment subspace (e.g. making prototypes of novel hardware). The former can benefit from a scalable optimizer, while the latter requires data-efficient optimization. Furthermore, any hierarchical considerations can be accommodated within the above co-design formulation as well. For example, temporal hierarchy can be obeyed by selecting embodiment first, then rolling out policy actions at each optimization trial, although this hierarchical view may only be necessary in some cases.

CO-DESIGN GYM reflects this premise: providing a unified interface to pair policy problems with explicit design spaces enables the broader community to study the doubly non-stationary problem.

4 CO-DESIGN GYM

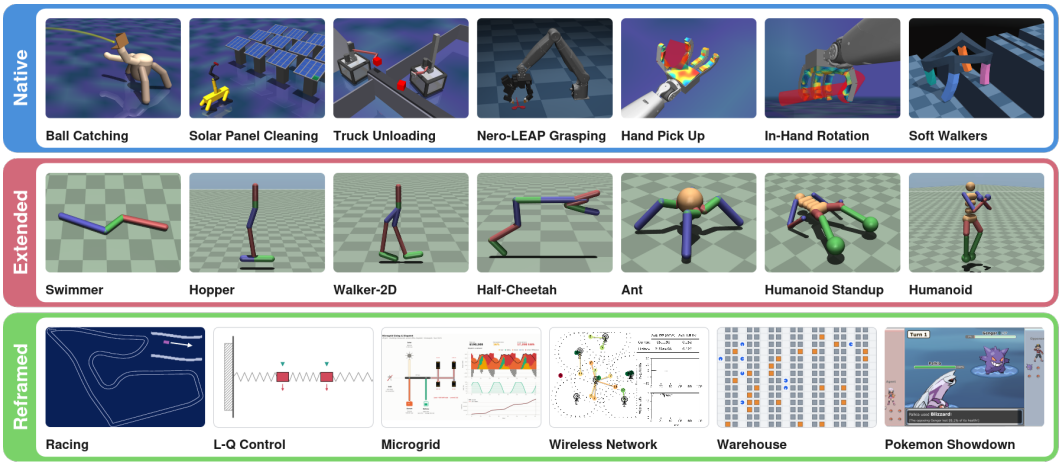

Figure 3: Overview of the available environments. For additional illustrations of the environment presets, refer to Appendix 4: Environments.

Co-design gym combines environments that span a wide range of domains, including but not limited to robotics, video games, and grid management. Our environments include completely from-scratch setups, co-design ports of the popular and classic OpenAI MuJoCo Control tasks in MuJoCo-Warp, and lifts of other RL problems and environments into co-design problems. Each environment pairs an inner policy optimization loop with an outer embodiment problem, and ships with several presets of varying complexity. All rigid-body robotics tasks are built in MuJoCo (Todorov et al., 2012) and MuJoCo Warp (DeepMind, 2025; Macklin, 2022). For full descriptions, reward definitions, and additional presets, please refer to Appendix A. Most of our environments are also natively GPU-accelerated.

Ball Catching Robot. A robot must catch a thrown ball in a glove while remaining upright. The policy controls the robot, and the embodiment varies its limbs and torso. Three presets modify the throw: fast and flat, high arc, and varied 3D fielding: testing co-design under dynamic conditions inspired by Morgan et al. (2026).

Solar Panel Cleaning. A quadruped robot with a roller-tipped arm must clean solar panels without any other body part touching them. The policy controls body and arm, and the embodiment modifies all limbs.

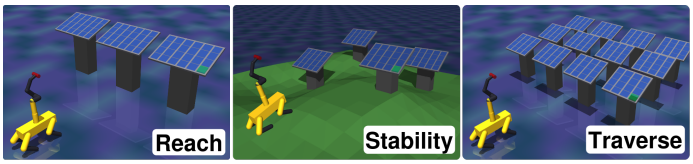

Figure 4: Solar panel cleaning

Presets cover a variety of solar panel layouts and terrains: targeting practical deployments of co-designed locomoting robots for intricate manipulation scenarios.

Multi-Robot Truck Unloading. A mobile manipulator unloads boxes from a truck onto a conveyor, where another sorts and shelves or stacks them by colour. A single policy controls all robots, and the embodiment modifies the arms of each robot. Presets vary the number of unloaders and different box colours: presenting a coordination challenge at the embodiment and control levels.

Shape-Shifting Hand. A four-fingered hand must either reorient an object in-hand or pick up and pose a cube, ball, or hammer. Supporting transition-level embodiment changes, these environments enable the use of a new class of design optimization algorithms for complex single-hand manipulation.

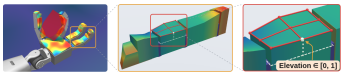

Figure 5: The Shape-shifting Hand for co-design

Nero-LEAP Grasping. A LEAP hand on a Nero arm must stably lift small tools and hardware from a table. The policy controls the hand, and the embodiment replaces the thumb and index

fingertip meshes. Presets cover different groups of objects: testing high-dimensional mesh-based shape design coupled with fine, contact-rich interactions.

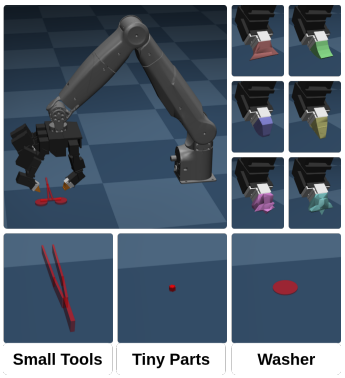

Figure 6: Nero-LEAP grasping

Soft Robot Walkers. Following Bhatia et al. (2022) and Cheney et al. (2014b), a soft robot must traverse flat, stepped, or gapped terrain. Each terrain is paired with five 2D/3D spring, MPM, beam, and FEM solvers (15 presets). Co-designing soft robots presents a unique challenge due to their deformable bodies and the large and varied design space spanning shape, mass, material, and actuator properties.

Car Configuration and Racing. A scaled F1Tenth car (O'Kelly et al., 2020) must minimise lap time on the Levine or Spielberg track. The policy drives the car. The embodiment sets the vehicle setup, such as tyre grip, cornering stiffness, and drive power, and each parameter trades off against the others.

Linear-Quadratic Structure Control. A mass-spring-damper chain must be regulated against disturbances. The embodiment includes the masses, stiffnesses, damping, and actuator and sensor placement. Three presets cover continuous design, combinatorial placement under noise, and budget-constrained design with saturated inputs, so the environment serves as a lightweight sanity check for co-design algorithms.

Microgrid Sizing and Dispatch. A microgrid must meet the randomised demand of one or more buildings over 24–72 hours, either grid-connected or off-grid (Henri et al., 2020). The policy dispatches energy, and the embodiment sizes solar, battery, and diesel capacity. Design costs are charged in the reward, making this a budgeted problem. A variety of presets test the idiosyncratic problem of grid control with careful resource allocation.

Wireless Network Placement. Base stations must give strong, consistent coverage to moving mobile users (Schneider et al., 2022). The policy assigns users to stations, and the embodiment includes the number, placement, and transmit power of the stations. Presets range from slow urban pedestrians to sparse, high-speed motorway traffic: combining the challenges of control in networking with geographic layouts and resource allocation.

Robotic Warehouse. A robot fleet fetches requested shelves and delivers them to goals, with the aim of maximising throughput (Bonnet et al., 2024; Christianos, 2020). The policy controls the fleet, and the embodiment lays out shelves on a fixed floor, trading route length against congestion. Presets scale from 2 to 8 robots: representing the challenges of finding optimal layout patterns for fleets of uniform agents.

Pokemon Showdown. The agent must win single battles, or a gauntlet of five League opponents, in Pokemon Showdown (University, 2011; Sahovic, 2019). The policy chooses moves, and the embodiment is the six-Pokemon team. Opponents range from random players to the League teams of all nine game generations, with or without healing between battles. With a combinatorially explosive discrete design space, and intricate environment dynamics, this tests sparse design space navigation in addition to complex environment rule-learning.

4.1 API

To facilitate ease of use we keep API changes minimal relative to the widely used Gymnasium (Towers et al., 2024) API. That way, existing control implementations remain usable and co-design methods differ only where necessary.

Any Gymnasium environment becomes a co-design environment by declaring an `embodiment_space`, a map from $\mathbf{e} \in \mathbf{E}$ to a realised body, and by forwarding the new argument through `reset`; the dynamics, reward, and termination logic are untouched. The original environment is recovered exactly as the base embodiment $\mathbf{e}_0$, so the conversion adds designs without altering the original task.

The `embodiment_space` is a Gymnasium `Space` representing $\mathbf{E}$ (see Section 3), alongside the usual `observation_space` and `action_space`. It supports `sample()` and `contains()`

with the standard semantics, so a design space can be enumerated, sampled from, or checked for membership by the same machinery used for actions.

Each environment declares a *base embodiment* $\mathbf{e}_0 \in \mathbf{E}$: the conventionally chosen body, with which it is realised until some other design is requested. Passing `None` retains the embodiment currently in force, which for an environment never given an explicit one is $\mathbf{e}_0$ throughout; passing $\mathbf{e} \in \mathbf{E}$ requests that the agent be realised as $\mathbf{e}$. An environment called with `embodiment=None` everywhere is therefore a standard Gymnasium environment with embodiment $\mathbf{e}_0$.

Within-task modifications Our API admits the embodiment argument in `step` as well as `reset`. This choice allows the embodiment to change during an episode. This is not physically meaningful for many tasks—bodies do not deform on command mid-trajectory—but it is potentially useful during training, where it permits a single rollout to experiment with multiple designs. Because within-episode modification is physically infeasible for many tasks, results obtained with it are not directly interpretable as task performance. A method that exploits it during evaluation would report returns no physical system could achieve. Each environment therefore carries an `is_eval_env` flag, fixed at construction, `env = CoDesignEnv(..., is_eval_env=True)`. If true, changes that alter the agent's embodiment take effect only when supplied to `reset`.

Researcher code

```python
import codesign_gym as cgym

# one entry point returns (env, base_embodiment)
env, base_embodiment = cgym.make_vec(environment_name, num_envs)

# an outer optimizer proposes a design ...
embodiment = design_optimizer.propose_embodiment()
obs, info = env.reset(embodiment=embodiment)

# ... an inner policy acts using it
for t in range(horizon):
    action = policy(obs, embodiment)
    if update_embodiment_each_step:
        embodiment = design_optimizer.propose_embodiment()
    obs, reward, term, trunc, info = env.step(action, embodiment)
```

Figure 7: Minimal usage of `codesign_gym`. A single entry point, `make_vec`, returns a natively parallelized environment together with its base embodiment supporting GPU-acceleration. An outer design optimizer proposes an embodiment that is passed to `reset`, and an inner policy then acts against it. The embodiment can optionally be re-proposed and passed to `step` at every timestep.

5 Experiments with Representative Optimization Methods

To demonstrate our environments, we benchmark four co-design algorithms on them. Existing research in co-design has not had a unified interface/setup and therefore makes state-of-the-art algorithms difficult to compare. Our selection of baselines represents different frontiers of co-design algorithms from evolutionary approaches and Reinforcement Learning (RL) to transformer-based search and hybrid approaches.

We compare against four baselines covering evolutionary, reinforcement-learning, alternating, and generative approaches to co-design. For all baselines, every 5% of training the current best embodiment and controller are evaluated in parallel environments on held-out seeds.

CMA-ES. CMA-ES searches over controller and embodiment parameters jointly as a single vector. The controller parameters are the weights of a neural-network policy.

FastTD3 (Seo et al., 2025). We add the embodiment parameters to the action space of FastTD3, an off-policy RL algorithm for continuous control. The embodiment actions are applied at the first step of each rollout and the control actions at every later step. Unused action components are zeroed in the replay buffer, so the policy learns only from actions that were actually taken.

PPO+NGOpt (Schulman et al., 2017; Rapin & Teytaud, 2018) PPO trains the controller. After each PPO update, NGOpt proposes one embodiment per training environment and scores it by its mean return in the rollouts where it was used.

LOKI. We generalise Jeon et al. (2026) to arbitrary embodiment parametrisations by replacing its robot-centric tokenization with a naive one. We pre-train its Transformer VAE on 500k randomly sampled valid embodiments. We then run the original online loop with PPO-trained Transformer controllers, using 6 clusters instead of 40 for comparability with the other baselines. At evaluation, the controller of the best cluster is paired with that cluster's top embodiments.

Preset	CMA-ES	FastTD3	PPO+NGOpt	LOKI
Native				
BallCatcher-Pitch	0.296 ± 0.121	**3.42 ± 1.62**	0.435 ± 0.325	0.0633 ± 0.0021
BallCatcher-Arc	0.316 ± 0.107	**4.15 ± 0.65**	1.59 ± 0.23	0.0875 ± 0.0140
BallCatcher-Fielding	0.143 ± 0.045	**3.15 ± 2.48**	0.461 ± 0.400	0.0739 ± 0.0037
NeroGraspAll	**2.34 ± 0.43**	−250 ± 0	0.147 ± 0.181	0.0899 ± 0.0195
SolarCleanerReach	0.0249 ± 0.0015	0.00639 ± 0.00182	**0.186 ± 0.270**	0.00613 ± 0.00207
TruckUnloadSingle	0.840 ± 0.391	**2.26 ± 2.36**	0.790 ± 0.097	0.506 ± 0.008
Reframed				
RacingSpielberg	6.77 ± 2.04	**76.9 ± 38.9**	64.1 ± 8.8	38.4 ± 1.8
WarehouseSmall	0.00313 ± 0.00383	0 ± 0	**12.6 ± 2.4**	0.738 ± 0.439
NetworkUrban	−87.2 ± 2.0	−100.0 ± 0.0	−81.4 ± 5.6	**−76.2 ± 1.1**
MicrogridRandom	−2,578 ± 549	**−1,286 ± 259**	−19,698 ± 1,023	−25,107 ± 130
PokenvRandom	−0.57 ± 4.44	3.81 ± 3.99	7.05 ± 3.95	**8.18 ± 3.10**
Extended				
HalfCheetah-v4	**8,033 ± 442**	3,317 ± 53	4,782 ± 145	3,400 ± 203
Hopper-v4	1,261 ± 72	897 ± 184	1,588 ± 891	**1,601 ± 213**
Walker2d-v4	**3,428 ± 1,564**	3,087 ± 736	3,257 ± 272	1,429 ± 177
Swimmer-v4	**405 ± 1**	83.2 ± 1.3	70.4 ± 26.7	53.1 ± 12.4
Ant-v4	3,726 ± 469	3,526 ± 170	**4,708 ± 284**	36.9 ± 5.7
Humanoid-v4	730 ± 15	**3,610 ± 1,654**	684 ± 57	379 ± 14
HumanoidStandup-v4	**226,124 ± 23,094**	73,522 ± 27,979	127,672 ± 15,381	79,271 ± 2,493

Table 1: The Co-Design Gym's first scoreboard: Final evaluation return (mean ± std over 5 seeds; higher is better) of each baseline under a matched 4-hour wall-clock budget (LOKI: 3 hours online after 1 hour of preparation). FastTD3 on NeroGraspAll: every proposed design failed structural validation, so every episode received the rejection reward (−250). Bold: best mean.

For a detailed description of our baselines, we refer to Appendix B.

We evaluate each of these methods on our co-design gym tasks, reporting performance across evaluation environments, embodiment rejection rates, and training behaviour. Each baseline and environment are trained under matched hardware conditions with 4-hour wall-clock time budgets other than LOKI. For comparability, we perform online training for LOKI for 3-hours, as an average of 1-hour was consumed in data generation, VAE pre-training, and cluster generation. Results are reported across five seeds.

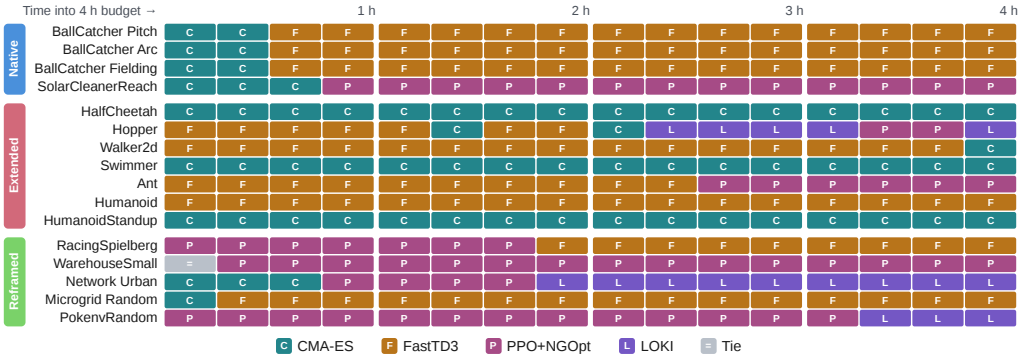

Figure 8: **Leading baseline over training** on the 16 presets run with all four baselines. Each cell names the method with the highest mean eval return (5 seeds) at that point, using the latest evaluation so far. CMA-ES leads early on 9 presets but at the end on only 4; FastTD3 ends ahead on 6, PPO+NGOpt and LOKI on 3 each. The leader changes at least once on 12 of 16 presets, and no method leads on every domain group, so the ranking of co-design baselines depends on both the task and the budget.

Contained Study: LLM Co-Design. Large language models (LLMs) have shown strong capabilities across a wide range of domains (Minaee et al., 2024), which raises the question of whether they can act as co-design optimizers. We examine this in a contained study in which an LLM (Gemma 4 31B) proposes robot embodiments in the outer loop while PPO learns the controller in the inner loop. Because this setup mirrors PPO+NGOpt with the LLM in place of NGOpt, it serves as an ablation of the outer-loop design optimizer. Interestingly, PPO+LLM stays near zero return throughout training, even though its prompt provides additional task-relevant information to the model. We refer to Appendix D for a detailed description.

6 Discussion and Conclusion

No single dominating co-design method. Across the 18 presets on which all four baselines were run, no method leads, and the emerged ranking depends on both the task and the budget (Figure 8). The final winners also cluster by domain. CMA-ES performs best on most of the extended locomotion tasks, FastTD3 on the ball-catching, racing and most microgrid presets, and PPO+NGOpt on most of the warehouse, wireless and Pokémon presets. Across the variety of domains represented in our benchmark, none of the evaluated co-design methods can claim general capability. This is made visible by the presentation of our unified benchmark; and fragmented evaluation settings of prior work would not reveal this.

Substantial headroom remains. Many presets are far from solved on which no co-design algorithm approached the return ceiling. For instance, on BallCatcher-Pitch the best method completes only 16.6% of catches. Additionally, all methods score near zero on more challenging benchmarks such as SolarCleaner, WarehouseMedium and WarehouseCongested. The benchmark therefore leaves ample room for methods that explore the co-optimization of embodiment and policy more efficiently than current approaches across continuous and discrete tasks. We hope that this encourages exciting new development of co-design algorithms to push the frontier on many fun, practical, and challenging domains.

Limitations and future work. For many proposed environments, the design space does not encompass all conceivable degrees of freedom; further extensions are possible, leaving room for richer parametrizations. Moreover, due to the breadth of the proposed benchmark and the generality of our underlying simulators, our baselines cover only a subset of the available presets. Furthermore, the fixed wall-clock budget in our experiments favors faster implementations, so part of the performance gap between baselines may reflect engineering rather than algorithmic differences. Finally, achieving high throughput with parallelized, GPU-accelerated physics simulation represents a substantial amount of work in structuring the compilation, caching, and workload scheduling. Incorporating more complex phenomena, such as fluids, wind, or smoke, may require integrating new simulators, with comparably large engineering effort.

Conclusion. We presented Co-Design Gym, a unified benchmark for the joint optimization of embodiment and policy. In our work we formalized co-design problems and solutions focusing on pairs of design spaces and policy classes, and situated standard RL as a special case of co-design with a single admissible design. We realized this formalism as an extension of the Gymnasium API, and used it to build on a large variety of widely used benchmark environments as novel co-design problems. In addition, we built a number of new environments from scratch specifically to stress-test the interplay between the embodiment and policy optimization, native and targeted to co-design. Finally, our experiments tested representative co-design methods and analyzed their performance. The appendix complements these results with full environment specifications and additional presets (Appendix A), baseline implementation details (Appendix B), per-preset learning curves and final designs (Appendix C), a contained study in which an LLM given privileged task knowledge fails as an outer-loop designer (Appendix D), an analysis of the design gap that fixed-design methods cannot measure (Appendix E), and a detailed illustration of the API (Appendix F). Unified benchmarks helped carry RL from struggling on basic tasks to superhuman performance. We hope Co-Design Gym can play the same role for co-design, advancing towards its broader promise: optimizing every dimension of a problem jointly to solve it as effectively as possible.

AI USE STATEMENT

(This section is **required** and does not count toward the page limit.)

In this work, we used generative AI tools for providing feedback on research experiments and implementing methods, and for feedback and suggestions on writing. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, formulating mathematical claims, proposing or refining hypotheses, assisting with translation, supporting qualitative and thematic data analysis, or interpreting results, and providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, generating synthetic data sets, cleaning and reformatting datasets are not applicable to this work. Additionally, we used generative AI tools for creating/modifying scientific figures or images, identifying relevant literature. We have reviewed all AI-assisted work. LLM-proposed literature was independently vetted and verified, and generative AI outputs were verified before being incorporated into scientific figures. Further, LLM-generated code was tested and verified for correctness by 2 authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Aiste Balciunaite, Mike Y Michelis, Miriam Filippi, Pablo Paniagua, Oncay Yasa, and Robert K Katzschmann. Co-optimized volumetric muscle design for large dynamic deformations in versatile biohybrid robots. *Nature Communications*, 2026.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub W. Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *ArXiv*, abs/1912.06680, 2019. URL https://api.semanticscholar.org/CorpusID:209376771.

Jagdeep Bhatia, Holly Jackson, Yunsheng Tian, Jie Xu, and Wojciech Matusik. Evolution gym: A large-scale benchmark for evolving soft robots. In *Neural Information Processing Systems*, 2022. URL https://api.semanticscholar.org/CorpusID:245011343.

Josh Bongard. Morphological change in machines accelerates the evolution of robust behavior. *Proceedings of the National Academy of Sciences*, 108(4):1234–1239, 2011.

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I. Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in jax, 2024. URL https://arxiv.org/abs/2306.09884.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Andrea Censi. A mathematical theory of co-design. *arXiv preprint arXiv:1512.08055*, 2015.

Changhe Chen, Xiaohao Xu, Xiangdong Wang, Jiaqi Wang, and Xiaonan Huang. Natural selection via foundation models for soft robot evolution. In *2026 IEEE 9th International Conference on Soft Robotics (RoboSoft)*, pp. 422–429. IEEE, 2026.

Nicholas Cheney, Jeff Clune, and Hod Lipson. Evolved electrophysiological soft robots. In *Artificial life conference proceedings*, pp. 222–229. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , 2014a.

Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding. *ACM SIGEVOlution*, 7(1): 11–23, 2014b.

Filippos Christianos. Multi-robot warehouse (rware), 2020. URL https://github.com/semitable/robotic-warehouse.

Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Transactions on Graphics (TOG)*, 32(4):1–12, 2013.

Francesco Corucci, Nick Cheney, Hod Lipson, Cecilia Laschi, and Josh Bongard. Evolving swimming soft-bodied creatures. In *ALIFE XV, the fifteenth international conference on the synthesis and simulation of living systems, late breaking proceedings*, volume 6, 2016.

Yanning Dai, Yuhui Wang, Dylan R. Ashley, and Jürgen Schmidhuber. Efficient morphology–control co-design via stackelberg proximal policy optimization. In *International Conference on Learning Representations (ICLR)*, 2026.

Giovanni De Michell and Rajesh K Gupta. Hardware/software co-design. *Proceedings of the IEEE*, 85(3):349–365, 1997.

Google DeepMind. mujoco$_w arp, 2025. URL$. GitHub repository. GPU-optimized version of the MuJoCo physics simulator, designed for NVIDIA hardware.. Accessed 2026-09-12.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan D. Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, J-M. Moret, Seb Noury, Federico Pesamosca, D. Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, B. P. Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin A. Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602:414 – 419, 2022. URL https://api.semanticscholar.org/CorpusID:246904229.

Edward R Dowski Jr and W Thomas Cathey. Extended depth of field through wave-front coding. *Applied optics*, 34(11):1859–1866, 1995.

Kehlani Fay, Darin Anthony Djapri, Anya Zorin, James Clinton, Ali El Lahib, Hao Su, Michael Thomas Tolley, Sha Yi, and Xiaolong Wang. House of dextra: Cross-embodied co-design for dexterous hands. 2025. URL https://api.semanticscholar.org/CorpusID:283466942.

George Gao, Tianyu Li, Junyao Shi, Yihan Li, Zizhe Zhang, Nadia Figueroa, and Dinesh Jayaraman. Vlmgineer: Vision-language models as robotic toolsmiths. In *International Conference on Learning Representations*, volume 2026, pp. 91732–91763, 2026.

Ismail Geles, Leonard Bauersfeld, Markus Wulfmeier, and Davide Scaramuzza. Superhuman safe and agile racing through multi-agent reinforcement learning, 2026. URL https://arxiv.org/abs/2605.22748.

Huy Ha, C Karen Liu, and Shuran Song. Transformer transformer: A unified model for motion-conditioned robot co-design. *arXiv preprint arXiv:2607.25798*, 2026.

Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Joint optimization of robot design and motion parameters using the implicit function theorem. In *Robotics: Science and systems*, volume 13, pp. 10–15607, 2017.

Donald J Hejna III, Pieter Abbeel, and Lerrel Pinto. Task-agnostic morphology evolution. *arXiv preprint arXiv:2102.13100*, 2021.

Gonzague Henri, Tanguy Levent, Avishai Halev, Réda Alami, and Philippe Cordier. pymgrid: An open-source python microgrid simulator for applied artificial intelligence research. *ArXiv*, abs/2011.08004, 2020. URL https://api.semanticscholar.org/CorpusID:226964357.

Jonathan Hiller and Hod Lipson. Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics*, 28(2):457–466, 2011.

Hyeonseong Jeon, Ainaz Eftekhar, Aaron Walsman, Kuo-Hao Zeng, Ali Farhadi, and Ranjay Krishna. Convergent functions, divergent forms. *Advances in Neural Information Processing Systems*, 38:46049–46076, 2026.

Sam Kriegman, Douglas Blackiston, Michael Levin, and Josh Bongard. A scalable pipeline for designing reconfigurable organisms. *Proceedings of the National Academy of Sciences*, 117(4): 1853–1859, 2020.

Phil Lord and Christopher Miller. Project hail mary. Film, March 2026. Screenplay by Drew Goddard, based on the novel by Andy Weir. Starring Ryan Gosling.

ML Luyben and CA Floudas. Analyzing the interaction of design and control—1. a multiobjective framework and application to binary distillation synthesis. *Computers & chemical engineering*, 18 (10):933–969, 1994.

Pingchuan Ma, Tao Du, John Z Zhang, Kui Wu, Andrew Spielberg, Robert K Katzschmann, and Wojciech Matusik. Diffaqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation. *ACM Transactions on Graphics (TOG)*, 40(4):1–14, 2021.

Miles Macklin. Warp: A High-performance Python Framework for GPU Simulation and Graphics, March 2022. URL https://github.com/nvidia/warp.

Gabriel B Margolis, Tao Chen, Kartik Paigwar, Xiang Fu, Donghyun Kim, Sangbae Kim, and Pulkit Agrawal. Learning to jump from pixels. *arXiv preprint arXiv:2110.15344*, 2021.

David Matthews, Andrew Spielberg, Daniela Rus, Sam Kriegman, and Josh Bongard. Efficient automatic design of robots. *Proceedings of the National Academy of Sciences*, 120(41):e2305180120, 2023.

Peter Milder, Franz Franchetti, James C Hoe, and Markus Püschel. Computer generation of hardware for linear digital signal processing transforms. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 17(2):1–33, 2012.

Shervin Minaee, Tomás Mikolov, Narjes Nikzad, Meysam Asgari Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. *ArXiv*, abs/2402.06196, 2024. URL https://api.semanticscholar.org/CorpusID:267617032.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Andrew S. Morgan, Gregory Xie, Capprin Bass, Rachel Thomasson, Chunpeng Wang, Erfan Shahriari, Harrison Busa, Oluwaseun Araromi, Joseph Aronov, Michael Burgess, Velin D. Dimitrov, Matthew A. Estrada, Faris Hamdi, Samuel Kendig, Taeyoon Lee, Jose Oscar Mur-Miranda, Emma Sommers, Paul Titchener, Margaret Wang, Achu Wilson, Mark Yeatman, Osman Dogan Yirmibesoglu, Alfred A. Rizzi, Annan Mozeika, Nicolas Rojas, and Lael Odhner. Athenazero: A low-inertia, bimanual robot for dynamic manipulation. *Science Robotics*, 11(118), 2026. ISSN 2470-9476. 10.1126/scirobotics.aee1868. URL http://dx.doi.org/10.1126/scirobotics.aee1868.

Matthew O'Kelly, Hongrui Zheng, Dhruv Karthik, and Rahul Mangharam. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. In *NeurIPS 2019 Competition and Demonstration Track*, pp. 77–89. PMLR, 2020.

Jahng-Hyon Park and Haruhiko Asada. Concurrent design optimization of mechanical structure and control for high speed robots. 1994.

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *Acm transactions on graphics (tog)*, 36(4):1–13, 2017.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, 37(4):1–14, 2018.

J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad, 2018.

Haris Sahovic. poke-env, 2019. URL https://github.com/hsahovic/poke-env. GitHub repository. Poke-env: Python Interface for Pokemon Showdown Bots. Accessed 2026-09-23.

M Salama, R Ramaker, and M Milman. Recent experience in simultaneous control-structure optimization. *NASA, Langley Research Center, NASA (DOD Controls-Structures Interaction Technology 1989*, 1989.

Stefan Schneider, Stefan Werner, Ramin Khalili, Artur Hecker, and Holger Karl. mobile-env: An open platform for reinforcement learning in wireless mobile networks. In *Network Operations and Management Symposium (NOMS)*. IEEE/IFIP, 2022.

John D. Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

Younggyo Seo, Carmelo Sferrazza, Haoran Geng, Michal Nauman, Zhao-Heng Yin, and Pieter Abbeel. FastTD3: Simple, Fast, and Capable Reinforcement Learning for Humanoid Control. 2025. 10.48550/ARXIV.2505.22642. URL https://arxiv.org/abs/2505.22642. Version Number: 3.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. URL https://api.semanticscholar.org/CorpusID:515925.

Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 15–22. 1994.

Andrew Spielberg, Brandon Araki, Cynthia Sung, Russ Tedrake, and Daniela Rus. Functional co-optimization of articulated robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5035–5042. IEEE, 2017.

Andrew Spielberg, Allan Zhao, Yuanming Hu, Tao Du, Wojciech Matusik, and Daniela Rus. Learning-in-the-loop optimization: End-to-end control and co-design of soft robots through learned deep latent representations. *Advances in Neural Information Processing Systems*, 32, 2019.

Andrew Spielberg, Tao Du, Yuanming Hu, Daniela Rus, and Wojciech Matusik. Advanced soft robot modeling in chainqueen. *Robotica*, 41(1):74–104, 2023.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012. 10.1109/IROS.2012.6386109.

Mark Towers, Ariel Kwiatkowski, Jordan K. Terry, John U. Balis, Gian Marco De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, KG Arjun, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. 2024. URL https://api.semanticscholar.org/CorpusID:290088643.

Smogon University. pokemon-showdown, 2011. URL https://github.com/smogon/pokemon-showdown. GitHub repository. Pokémon battle simulator.. Accessed 2026-09-12.

Jeffrey Ventrella. Explorations in the emergence of morphology and locomotion behavior in animated characters. 1994.

Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. *ACM Transactions on Graphics (TOG)*, 28(3):1–8, 2009.

Hanwen Wang, Zhenlong Fang, Josiah Hanna, and Xiaobin Xiong. Efficient and versatile quadrupedal skating: Optimal co-design via reinforcement learning and bayesian optimization. *arXiv preprint arXiv:2603.18408*, 2026a.

Jianren Wang, Letian Qian, Zikai Wang, Weiwei Wu, Junjie Zong, Abhinav Gupta, and Deepak Pathak. Bridge: An open-source humanoid platform via morphology-control co-design for physical ai. *arXiv preprint arXiv:2609.03497*, 2026b.

Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. Neural graph evolution: Towards efficient automatic robot design. *arXiv preprint arXiv:1906.05370*, 2019.

Tsun-Hsuan Wang, Pingchuan Ma, Andrew Everett Spielberg, Zhou Xian, Hao Zhang, Joshua B. Tenenbaum, Daniela Rus, and Chuang Gan. Softzoo: A soft robot co-design benchmark for locomotion in diverse environments. In *The Eleventh International Conference on Learning Representations*, 2023a. URL `https://openreview.net/forum?id=Xyme9p1rpZw`.

Tsun-Hsuan Johnson Wang, Juntian Zheng, Pingchuan Ma, Yilun Du, Byungchul Kim, Andrew Spielberg, Josh Tenenbaum, Chuang Gan, and Daniela Rus. Diffusebot: Breeding soft robots with physics-augmented generative diffusion models. *Advances in Neural Information Processing Systems*, 36:44398–44423, 2023b.

Andy Weir. *Project Hail Mary*. Ballantine Books, New York, 2021. ISBN 9780593135204.

Jie Xu, Andrew Spielberg, Allan Zhao, Daniela Rus, and Wojciech Matusik. Multi-objective graph heuristic search for terrestrial robot design. In *2021 IEEE international conference on robotics and automation (ICRA)*, pp. 9863–9869. IEEE, 2021.

Xiaomeng Xu, Huy Ha, and Shuran Song. Dynamics-guided diffusion model for sensor-less robot manipulator design. *arXiv preprint arXiv:2402.15038*, 2024.

Chen Yu, David Matthews, Jingxian Wang, Jing Gu, Douglas Blackiston, Michael Rubenstein, and Sam Kriegman. Agile legged locomotion in reconfigurable modular robots. *Proceedings of the National Academy of Sciences*, 123(10):e2519129123, 2026.

Ye Yuan, Yuda Song, Zhengyi Luo, Wen Sun, and Kris Kitani. Transform2act: Learning a transform-and-control policy for efficient agent design. *arXiv preprint arXiv:2110.03659*, 2021.

Gioele Zardini. *Co-design of complex systems: From autonomy to future mobility systems*. PhD thesis, ETH Zurich, 2023.

Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

APPENDIX A: ENVIRONMENTS

Unless stated otherwise, the tasks described below use MuJoCo (Todorov et al., 2012) and MuJoCo Warp (DeepMind, 2025; Macklin, 2022) as the physics engine.

Ball Catching Robot. In this environment a robot inspired by Rocky from Project Hail Mary (Weir, 2021; Lord & Miller, 2026) is tasked with catching a ball that is thrown towards it. At the end of the arm a "glove" is attached, in which the ball must be caught. The source of the ball is fixed but the target and speed are randomised. We provide three presets. 1) BallCatcherPitch in which the ball is thrown fast, flat, and low. 2) BallCatcherArc in which the ball is thrown in a slower, high arc descending towards the quadruped. 3) BallCatcherFielding in which the ball is thrown with varied speed and direction toward a broad 3D catch region requiring versatility. In these tasks, the control optimization loop is training a policy for the arm-equipped quadruped, and the design problem is the configuration of the arm. Partial rewards are obtained for catching and dropping the ball, but full reward (10) is accomplished when catching, retaining the ball, and with all four legs touching the ground. This final condition is required as the robot likes to learn to jump.

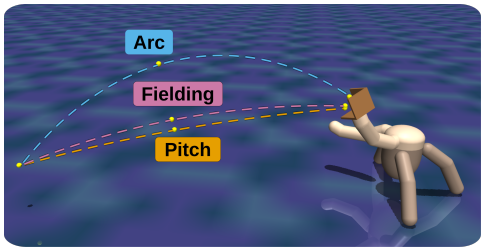

Figure 9: Ball Catching

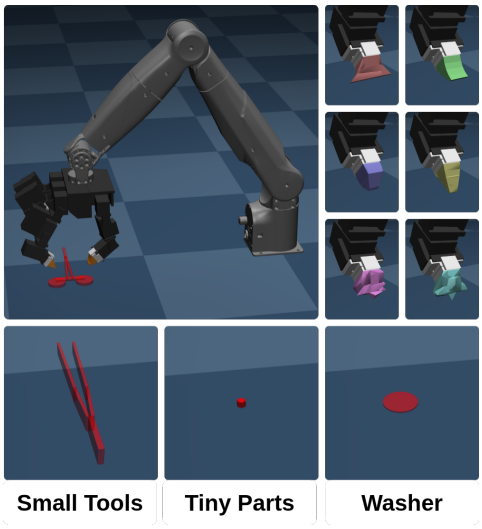

Figure 10: Handling small objects

Nero-LEAP Grasping Robot. A Nero robot arm with a LEAP hand is tasked with picking up small objects from a flat surface. The LEAP hand has fingertips affixed. The objects are randomized from a set of ten divided into three categories. A success in this task is considered a sustained, controlled, and stable lift of the object. We provide four main presets. 1) NeroGraspSmall in which the object is randomly selected from the 'small' category. 2) NeroGraspTiny in which the object is randomly selected from the 'tiny' category. 3) NeroGraspThin in which the object is a thin washer whose radius is randomly selected. 4) NeroGraspAll in which the object is randomly selected from all ten available objects. There are additional presets for each object, which are further described in Appendix A.1. The trajectory of the Nero arm is fixed, and the control optimization loop is training a policy for the LEAP hand. The design problem in this task is to modify the meshes of the LEAP fingertips on the thumb and index, to enable a stable and sustained lift. The base vertices of the mesh are fixed to the robot, but the remainder of the mesh can be fully replaced by the user.

Multi-Robot Truck Unloading. In this task mobile manipulator robots are tasked with unloading a truck packed with boxes. One mobile manipulator collects boxes from the truck and places them on a conveyor belt, around which other robots are placed. These must collect different boxes from the conveyor (keyed by color) and place them in a specified area. We provide two presets. 1) TruckUnloadSingle in which there is a single mobile manipulator collecting the red boxes from the conveyor belt, and placing them on a shelf. 2) TruckUnloadMulti in which there is an additional mobile manipulator collecting the blue boxes from the conveyor, and stack these boxes into an area behind them. In these tasks, the control optimization loop

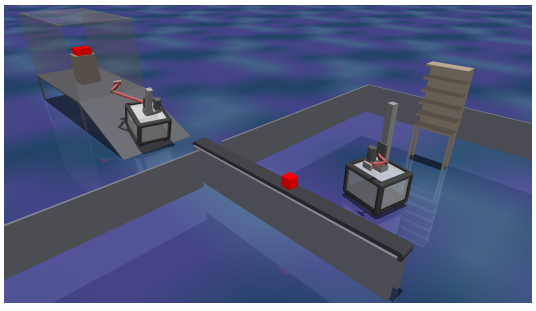

Figure 11: Multi-robot truck unloading.

is training a single policy to control all of the robots. The design problem is to modify the robot arms on the mobile manipulators to enable dexterous and quick unloading and packing of boxes.

Solar Panel Cleaning Robot. In this environment a quadrupe is equipped with an arm and tasked with carefully cleaning an assortment of solar panels. At the end of the arm is a small roller which cleans a solar panel cell when in contact. Since panels are delicate, no other part of the robot may come in contact with the panel cells. We provide three presets. 1) SolarCleanerReach in which three panels are mounted higher up than the base configuration of the robot and laid out in a line. 2) SolarCleanerStability in which four panels are distributed across a rocky and uneven terrain. 3) SolarCleanerTraverse in which a 3x4 grid of solar panels is tightly aligned forming narrow aisles for robot traversal. In these tasks, the control optimization loop is training a policy to control the Ant robot and arm. The design problem is to modify the legs and the arms of the robot to enable stable and careful locomotion alongside precise arm control.

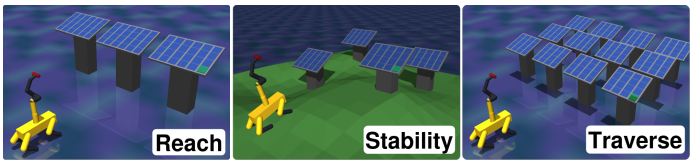

Figure 12: Solar panel cleaning

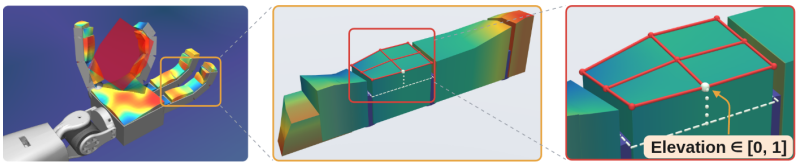

Figure 13: The Shape-shifting Hand co-design environment.

Shape-shifting Hand. In this environment a four-fingered anthropomorphic hand with a 3-DoF wrist is tasked with reorienting an object in its palm to a target orientation. Targets are expressed relative to the palm so that turning the wrist alone does not reward the agent. When the object is within a set tolerance of the target orientation, a new one is sampled a set angular distance away from its current orientation. Reward is given for lowering the relative angle between the current orientation and the target and for holding the object at the target. This reward is only given when the object is in contact with the hand to prevent reward hacking via throwing it. The return for meeting a single goal is 10. The episode ends if the cube falls off the hand, or 7.5 s pass without a target being met. We provide four presets. 1) RotateInHand, in which an 8 cm cube is used, the goal tolerance is 0.1 rad and new targets are generated 1 rad away from a met target; 2) HandPickUp-Cube, in which the hand is picking up an 8 cm cube; 3) HandPickUp-SquashBall, in which the hand is picking up a ball; 4) HandPickUp-Hammer, in which the hand is picking up a hammer. In this task, the control optimization loop is training a policy for the 19 velocity-controlled joints of the fingers and wrist. The design problem is to modify the contact surfaces and proportions of the hand. The palm and every finger segment are covered by height field pads, each a 3×3 grid of points, giving 216 surface parameters. Three more embodiment parameters are added - finger spacing, finger width, and thumb width.

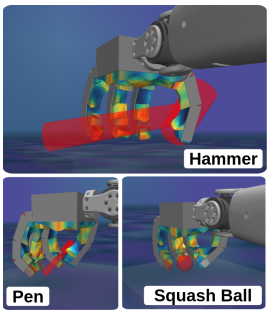

Figure 14: The Shape-shifting Hand Pick Up co-design environment.

Shape-shifting Hand Pick Up. This environment reuses the hand from "Shape-shifting Hand" in a more complex task that requires establishing a stable, dexterous grasp. The task is to pick up an object lying on a table beneath the hand and reposition it in a target position and orientation. Unlike in-hand rotation, the object must first be lifted - a target only counts if the object is touching the hand without touching the table, and reward is halved while the object touches both. A small additional reward term is given for reducing the distance from the fingertips to the object. We provide three presets. 1) HandPickUp-Cube in which an 8 cm cube must fulfil target orientations. 2) HandPickUp-SquashBall in which the object is instead a 4 cm ball. 3) HandPickUp-Hammer in which a hammer object must fulfil a target position orientation, corresponding to a cylindrical grip. The control optimization loop is training a policy for the fingers and wrist, and the design problem is the same as in the In-Hand Rotation task: the elevation of the 216 heightfield points, and the three finger configuration parameters (finger spacing, finger width, thumb width).

Pokemon Showdown. In this task a Pokemon battle is conducted using a team of six Pokemon tasked with winning the battle or sequence of battles against an opponent trainer. We provide four main presets. 1) PokenvRandom in which the opponent's Pokemon team is randomly sampled and the opponent agent takes random moves in a Pokemon Generation 4 Single Battle. 2) PokenvMaxPower in which the opponent's Pokemon team is randomly sampled and the opponent agent chooses moves with the highest base power in a Pokemon Generation 6 Single Battle. 3) PokenvHeuristic in which the opponent's Pokemon team is randomly sampled and the opponent agent plays according to the SimpleHeuristicPlayer (Sahovic, 2019) in a Pokemon Generation 9 Single Battle. 4) PokenvLeague in which the opponent is a sequence of 5-opponents representing the Pokemon League, an elite set of trainers from the video games. This preset has a number of variants, which we discuss in more detail in Appendix A.1. In this task, the control optimization loop is tasked with finding the best policy to play with a given Pokemon team, and the design problem is to build the optimal Pokemon team. This task is built on Pokemon Showdown (University, 2011) and Poke-env (Sahovic, 2019).



Figure 15: The Pokemon Showdown co-design environment.

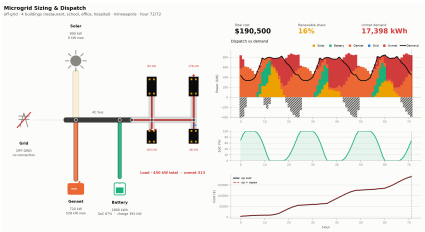

Figure 16: The Microgrid Sizing and Dispatch co-design environment.

Microgrid Sizing and Dispatch. In this budgeted optimization task an energy Microgrid is supplying power to a small collection of buildings, each with their own power demands and schedules which are randomised. A success is considered when there is no unmet demand and there was little unnecessary spending over a given time frame. We provide three presets where the Microgrid is connected to a larger grid. 1) MicrogridRandom (24 hours) in which there is a single random building connected to the grid. 2) MicrogridForecast (24 hours) in which there is a single random building connected to the grid and observations are augmented with noisy 12 hour forecasts. 3) MicrogridCampus (24 hours) in which there are several buildings. We provide a further two presets where the Microgrid is not connected to a larger grid. 1) MicrogridOffGridSingle (72 hours) in which there is a single random building connected to the grid. 2) MicrogridOffGridCampus (72 hours) in which there are several buildings connected the grid. In these tasks, the control optimization loop is training a policy for energy flow, and the design problem specifies solar capacity, battery size, and diesel generator capacities. Each design choice incurs a cost which is paid in the task reward at the first step, forming a budgeted co-optimization problem. This task is built on `python-microgrid` (Henri et al., 2020).

Wireless Network Placement and Connection. In this budgeted optimization task a set of wireless stations are covering a high-traffic area providing service to mobile-phone users. Success in this task is directly correlated with the strength and regularity of the signal provided across all users. We provide three presets. 1) NetworkUrban in which pedestrian low-velocity users are moving in random movement patterns. 2) NetworkSuburban in which there is a mix of velocities with some vehicular users in random movement patterns. 3) NetworkMotorway in which users are travelling at high velocities, and terrain requires sparse coverage. In these tasks, the control optimization loop chooses which base station each mobile user is provided connection to as they move around the map. The design problem specifies how many base stations to deploy (under a maximum), where they are placed, and what transmission power capacity each station is given. This task is built on Schneider et al. (2022).

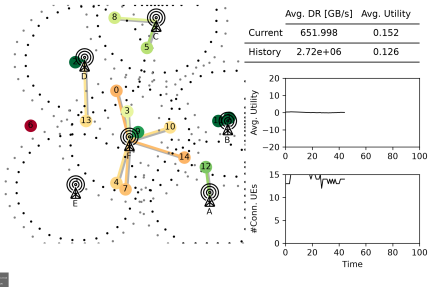

Figure 17: The Wireless Network Placement and Connection co-design environment.

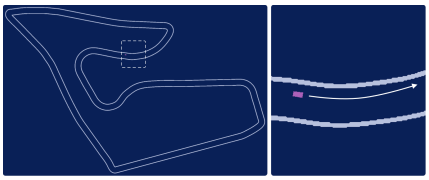

Figure 18: WIP: F1Tenth environment.

Car Configuration and Racing. In this task a scaled racing car is driving around a track, trying to complete laps in the lowest possible time. Each car is modelled with load sensitivity, yaw-inertia, and friction trade-offs. We provide two presets. 1) RacingLevine in which the car is racing around the Levine track preset. 2) RacingSpielberg in which the car is racing around the Spielberg track preset. In these tasks, the control optimization loop is training a policy for the car and the design problem specifies the vehicle setup. This configuration includes tyre grip, cornering stiffness, drive power, and more which all possess trade-offs. This task is built on F1Tenth (O'Kelly et al., 2020).

Robotic Warehouse Control and Layout. In this task a warehouse is equipped with multiple robotic agents with the goal of providing the highest throughput of request completion. These requests are deliveries to goals, and they are randomized for each environment within a certain max load. We provide three presets. 1) WarehouseSmall in which there is a 10x10 warehouse floor, two robots, and $\geq$25 shelves. 2) WarehouseMedium in which there is a 16x16 warehouse floor, four robots, and $\geq$115 shelves. 3) WarehouseCongested in which there is a 20x22 warehouse floor, eight robots, and $\geq$264 shelves. In these tasks, the control optimization loop is training a policy for the fleet of robots to jointly navigate them to retrieve requested shelves and deliver them to goals. The design problem specifies the layout and placement of the shelves on a fixed floor trading off route length and congestion. This task is built on Jumanji (Bonnet et al., 2024) and RWARE (Christianos, 2020).

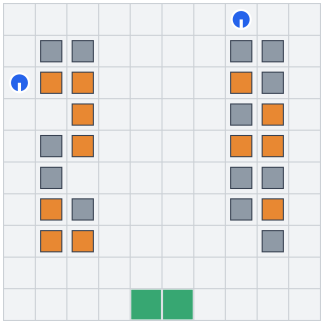

Figure 19: The Robotic Warehouse Control and Layout co-design environment.

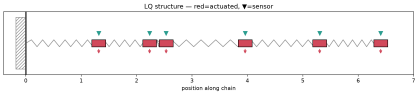

Figure 20: The Linear-Quadratic Structure Control co-design environment.

Linear-Quadratic Structure Control. In this task a mass-spring-damper chain is regulated against environmental disturbances. We provide three presets: 1) continuous design with unconstrained, full-state control; 2) actuator and sensor placement with control under process and measurement noise; and 3) actuator placement under a mass and actuator budget with saturated control inputs. In these tasks, the control optimization loop trains a policy to regulate the structure against the disturbances, while the design problem specifies the masses, stiffnesses, damping, and the actuator/sensor placements. Because the three presets span continuous, combinatorial, and budget-constrained design, this environment is designed to serve as a lightweight sanity-checker exercising the distinct facets of a co-design algorithm.

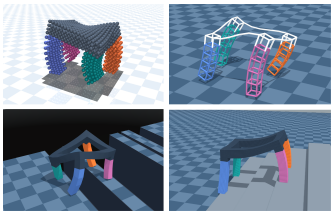

Figure 21: The Soft Robot co-design environment.

Soft Robot Walkers. In this task a soft robot is tasked with efficiently traversing a given domain. We provide three terrain presets: 1) a flat terrain; 2) a terrain with equidistant stairs; 3) a terrain with gaps and holes. For each of these three terrains, we provide five versions of the problem each supported by GPU-accelerated implicit Euler integration: 1) in 2D using a Spring-solver (Bhatia et al., 2022); 2) in 2D using an Material Point Method (MPM)-solver (Spielberg et al., 2023); 3) in 3D using an MPM-solver (Spielberg et al., 2023); 4) in 3D using a Beam-solver (Cheney et al., 2014b); 5) in 3D using an Finite Element Method (FEM)-solver (Ma et al., 2021). This leads to a total of 15 unique presets. In these tasks, the control optimization loop trains a policy to control the actuation of the soft robot walker. The design problem specifies the shape, mass, actuation, and material properties of the robot. This task was inspired by Bhatia et al. (2022) and Cheney et al. (2014b), and is built on NVIDIA Warp Macklin (2022).

A.1 Additional Environment Presets

Alongside the main presets described and presented in Section 4. There are additional environment configurations exposed by the gym for the Nero-LEAP Grasping and Pokemon Showdown environments.

A.1.1 Nero-LEAP Grasping

The NeroGrasp class of environment presets contains ten objects. These are scissors, tweezers, spatula, stick, screw, long screw, headless screw, t-nut, lock nut, and washer. Each object is categorised in the main presets, but can be launched on its own as well. This exposes nine additional presets.

NeroGrasp Additional Presets	
Object Name	**Preset Name**
Scissors	NeroGraspScissors
Tweezers	NeroGraspTweezers
Spatula	NeroGraspSpatula
Stick	NeroGraspStick
Screw	NeroGraspScrew
Long Screw	NeroGraspScrewLong
Headless Screw	NeroGraspSetScrew
T-Nut	NeroGraspTNut
Lock Nut	NeroGraspLockNut

A.1.2 Pokemon Showdown

The Pokenv class of environment presets exposes the PokenvLeague preset. This environment is the most challenging of the Pokemon presets, and involves fighting 5 designed opponents in sequence, mimicking the Pokemon League from the video games. Over the nine generations of Pokemon video games, there are nine iterations of the Pokemon League, who are typically challenged by the player towards the end of the main storyline. In this environment preset, we therefore instantiate all nine of these leagues with their teams taken directly from the video games, each controlled by the SimpleHeuristicPlayer (Sahovic, 2019). Each league exposes two presets. One where the agent's team is "healed" and reset to full strength between battles, and one where it is not. The second class of presets is substantially harder, and requires the co-design learner to design a team and play such that it can survive 5 successive opponents. In addition to these 18 presets, we also expose two additional presets which randomly select the opponent league from all nine generations.

PokenvLeague Presets			
		Team status reset?	
Opponent league	**Base name**	**Yes**	**No (+H)**
Generation X, $X \in \{1, \ldots, 9\}$	`PokenvLeague-G`X	`-G`X	`-G`X`H`
Random generation	`PokenvLeague-Rand`	`-Rand`	`-RandH`

B BASELINE OPTIMIZATION METHODS

Baseline 1: CMA-ES. The CMA-ES evolutionary global optimizer is deployed for co-design by sampling embodiments and controller parameters from the algorithm. This puts both the controller and the embodiment into the same joint search space from which the algorithm samples. The controller parameters sampled are used in a neural-network controller and the embodiment values are passed directly to the environment. At evaluation boundaries (every 5% of training), the current best controller-embodiment pair is evaluated in multiple parallel evaluation environments on held-out seeds.

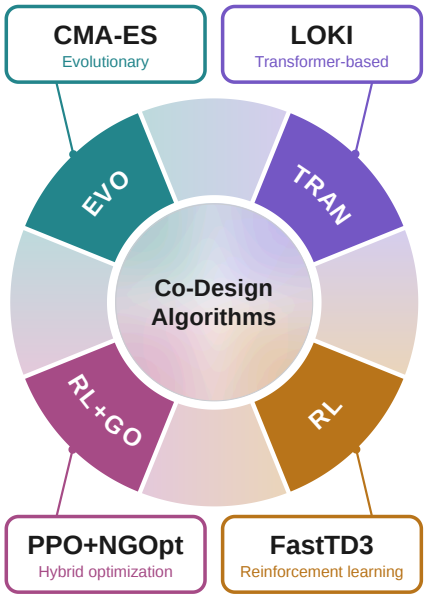

Figure 22: WIP: An overview of the space of co-design algorithms highlighting our chosen baselines.

Baseline 2: FastTD3. FastTD3 is an off-policy state-of-the-art RL algorithm predominantly designed for continuous control tasks, such as robot control. To instantiate FastTD3 for co-design, we expand its action space to include both the embodiment parameters and the control actions. At the first step of a FastTD3 rollout, the chosen embodiment values are used to instantiate/reset the environment and the control actions are discarded. In subsequent steps, the embodiment values are discarded and the control actions are used to step through the environment. The replay buffer is populated with the corrected samples (after setting either the embodiment values or control actions to 0) so that the policy learns from which actions were actually taken. At evaluation boundaries (every 5% of training), the current policy generates a new embodiment and control actions for multiple parallel evaluation environments on held-out seeds.

Baseline 3: PPO and NGOpt. In this baseline PPO and NGOpt are used in alternation to perform co-design. PPO is used as normal to train the controller. After each PPO update iteration, NGOpt samples one embodiment per training environment. Samples from NGOpt are scored using the mean return obtained by that embodiment in the rollouts in which it was used. At evaluation boundaries (every 5% of training), the current best embodiment is used to instantiate multiple parallel evaluation environments on held-out seeds and is controlled with the current PPO policy.

Baseline 4: LOKI (Jeon et al., 2026). In this baseline algorithm, we generalise the approach of Jeon et al. (2026) to arbitrary embodiment parametrisations. For each set of embodiment parameters $\mathbf{e}$, we construct a naive tokenization scheme, in place of the robot-centric tokenization scheme from the original work. Following this, we randomly sample $500,000$ valid embodiments and pre-train the Transformer VAE on these embodiments following the original training recipe. Afterwards, we run the online training loop of Jeon et al. (2026) with 6 clusters (as opposed to the original 40) to ensure comparability with other baselines, using PPO to train Transformer controllers. At evaluation boundaries (every 5% of online training), the controller from the best cluster is selected and the top performing embodiments from that cluster are used to instantiate multiple parallel evaluation environments on held-out seeds.

C DETAILED RESULTS

Preset	CMA-ES	FastTD3	PPO+NGOpt
Native			
SolarCleanerTraverse	**0.0185 ± 0.0159**	0.00431 ± 0.00335	0.00043 ± 0.00133$^{(6)}$
Reframed			
RacingLevine	6.07 ± 1.03	**48.3 ± 11.6**	39.6 ± 10.3
WarehouseMedium	0 ± 0	0 ± 0	**0.0172 ± 0.0125**
WarehouseCongested	0 ± 0	−50 ± 100	**0.00313 ± 0.00625**
NetworkSuburban	−15.8 ± 2.4	−92.0 ± 1.7	**−6.99 ± 3.23**
NetworkMotorway	39.5 ± 2.5	−83.0 ± 28.2	**57.4 ± 2.6**
MicrogridForecast	−4,958 ± 285	**−1,124 ± 114**	−22,178 ± 195
MicrogridCampus	−14,121 ± 1,442	**−5,793 ± 449**	−92,551 ± 11,476
MicrogridOffGridSingle	−16,576 ± 5,792	**−13,917 ± 8,719**	−81,464 ± 1,249
MicrogridOffGridCampus	**−38,235 ± 2,479**	−81,419 ± 50,542	−260,851 ± 4,386
PokenvMaxPower	−4.82 ± 5.32	−3.83 ± 4.22	**−0.31 ± 3.34**
PokenvHeuristic	−15.0 ± 3.9	−8.49 ± 3.92	**−3.64 ± 4.98**

Table 2: Final evaluation return (mean ± std over 5 seeds; higher is better) on the remaining presets, run with CMA-ES, FastTD3 and PPO+NGOpt under a matched 4-hour wall-clock budget. Bold: best mean. †: runs still in progress. $^{(n)}$: $n \neq 5$ seeds.

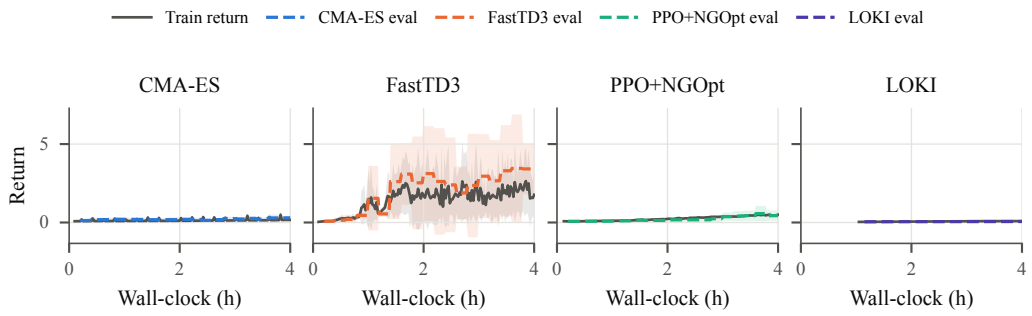

Figure 23: **BallCatcher-Pitch.** Only FastTD3, which emits the design as the first action of each episode, learns to catch (16.6% full catches vs. ≤1.1% for the others; final return 3.42 vs. ≤0.44). Its final return varies widely across seeds (1.35–5.98), so the joint design-and-policy agent is effective here but unreliable. None of the outer-loop methods reaches catching behaviour within the budget.

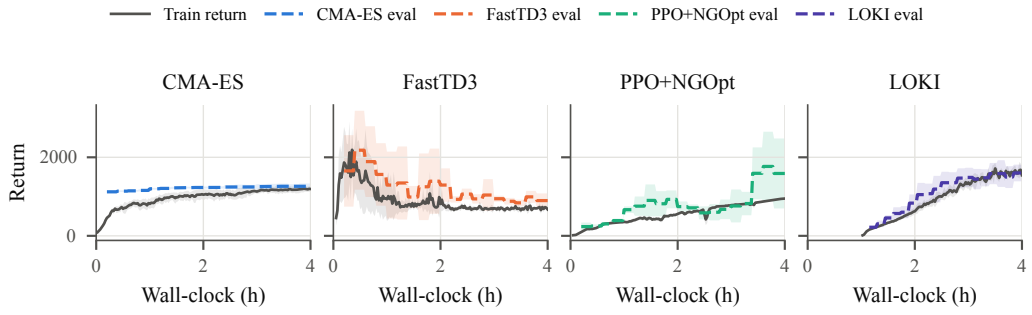

Figure 24: **Hopper.** The four families trace four different learning curves on the same task, and the leader changes six times in four hours. FastTD3 peaks early (2,800 at 0.6 h) and decays to 897; PPO+NGOpt jumps late (761→1,588 in the final hour, seeds 462–2,885); LOKI improves steadily throughout; CMA-ES plateaus at ≈1,260 almost immediately. A single budget checkpoint would rank the methods differently at almost every time.

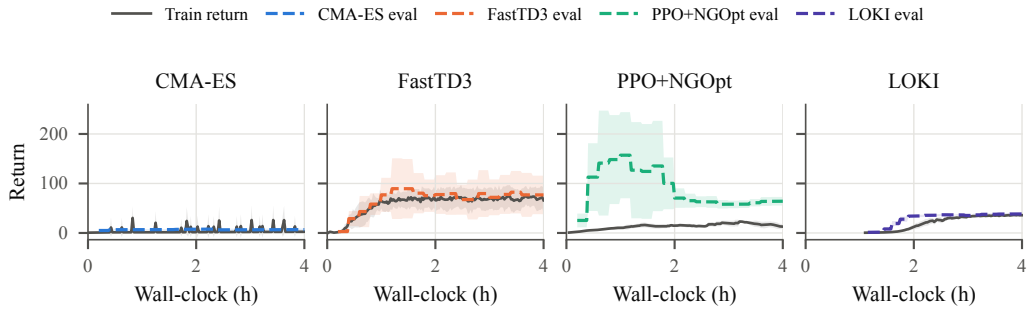

Figure 25: **RacingSpielberg.** PPO+NGOpt finds a strong vehicle setup within the first hour (eval 157 at 1 h) and then loses it, ending at 64 after ≈245 design rounds: more outer-loop design updates are not monotonically better, and a good design found early is not retained. Its train return (≈13) stays far below eval because most rounds train on exploratory setups. FastTD3 is stable at ≈77; CMA-ES never leaves ≈7.

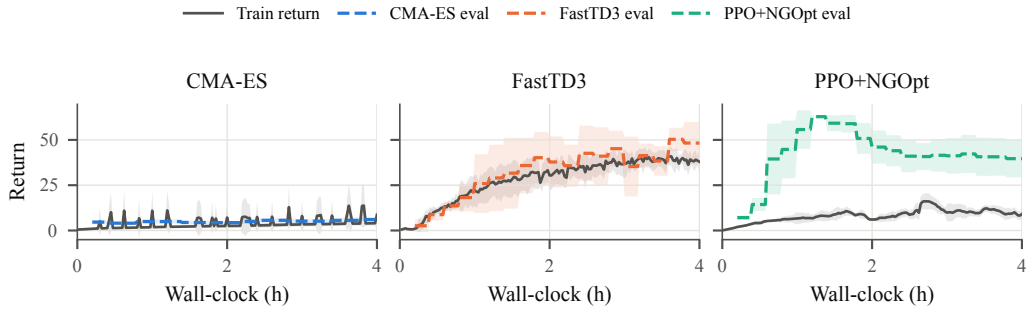

Figure 26: **RacingLevine.** The same pattern as on RacingSpielberg on a second track: PPO+NGOpt peaks within the first hour (63) and declines to 40 over ≈274 design rounds, while FastTD3 improves steadily, overtakes it and ends with the most completed laps (37% vs. 20%). Early design-search gains that are not retained recur across tracks, so they reflect the algorithm rather than a single environment.

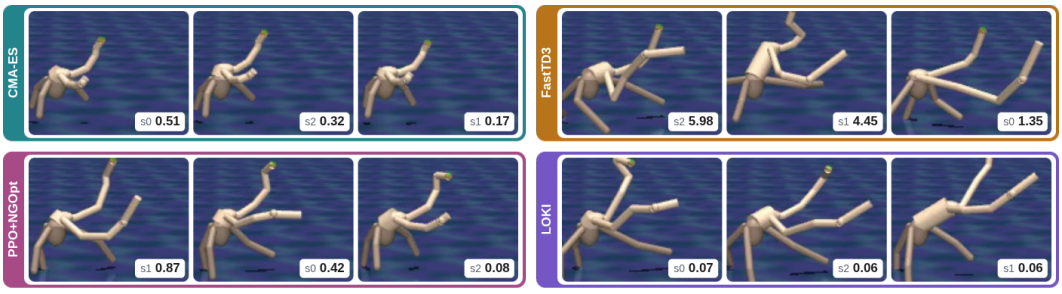

Figure 27: **Final designs on BallCatcher-Pitch**, one per run, taken from the first frame of the run's final evaluation rollout and shown at a common scale; seeds are sorted by final eval return (above each image) and the best is framed thick. The methods converge to visibly different bodies, and FastTD3 and LOKI also differ from seed to seed, whereas CMA-ES's designs look alike: similar returns can come from different morphologies.

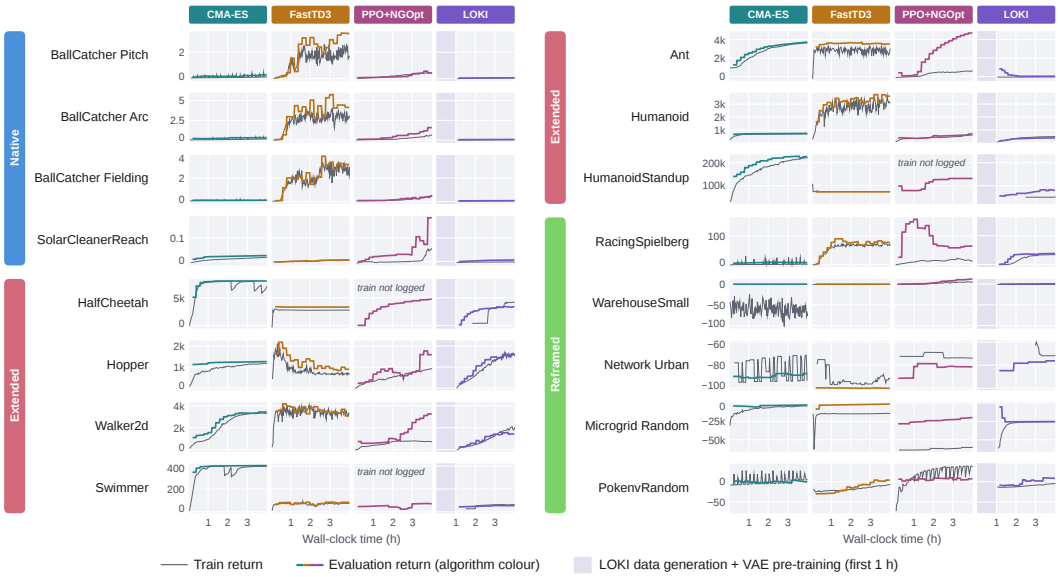

Figure 28: **Train vs. eval return on every preset run with all four baselines** (mean over 5 seeds; one block of panels per preset, sharing a y-axis). Train return averages over the designs being trained or explored and eval return uses the selected design, so the gap measures each method's cost of design exploration; for CMA-ES on the Warehouse presets the train return also includes the penalty for rejected designs. PPO+NGOpt does not log a train return on HalfCheetah, Swimmer and HumanoidStandup, and its HumanoidStandup runs were still in progress when this figure was generated.

D LLM CO-DESIGN

In this experiment, we use Gemma 4 31B as the design optimizer in a PPO+LLM co-design loop on the BallCatcher-Pitch preset. The algorithm mirrors the PPO+NGOpt baseline (Section 5), with the LLM taking the place of NGOpt as the outer-loop design optimizer. The PPO controller is conditioned on the parameters of the embodiment it controls.

We query the LLM for embodiment proposals using the prompt below, with a temperature of X. Placeholders of the form {VAR} are replaced with the corresponding values at query time.

This prompt conditions the LLM on far more information than any other baseline receives. It conveys details of the environment that the other baselines can only learn through interaction, and it provides context on each embodiment parameter, such that a model could anticipate the relationship between provided embodiment parameters and design.

To ensure a fair comparison, PPO+LLM must train on as many embodiments per design round as the other baselines. Otherwise, any difference in performance could simply reflect a narrower exploration of the design space. PPO+NGOpt, its direct counterpart, samples one embodiment for each of the 16,384 parallel environments. Having the LLM generate that many designs directly, however, is infeasible under API rate limits and token costs. We therefore split design generation into two stages. In each design round, the LLM proposes 16 *template* embodiments, and each template is perturbed with Gaussian noise (standard deviation 0.05) to yield 1,024 variants, giving 16,384 embodiments in total. The prompt informs the LLM of this perturbation scheme. For evaluation runs, both methods evaluate the single best-scoring design found so far.

Results. Despite the informational advantage granted to the agentic designer, PPO+LLM performs poorly in all runs, reaching low final return and showing little learning progress. The embodiments proposed by the LLM were often impractical and over-engineered, such that, even when the controller did learn meaningful behaviour, the given design stalled learning progress.

SYSTEM INSTRUCTION: Propose exactly {NUM_TEMPLATES} embodiment templates in one response. Return only a JSON object containing an `embodiments` array with exactly {NUM_TEMPLATES} lists. Each list must contain exactly {EMBODIMENT_DIM} numeric values in parameter-index order. Maximize the mean episodic return across random perturbations of each proposed template. Respect all parameter bounds and integer constraints. Copies of each template receive independent Gaussian perturbations scaled by the corresponding parameter ranges. Perturbed values are clipped to their bounds and integer-valued parameters are rounded. Use the supplied template-statistics history to explore the design space and improve robust designs. The reported statistics describe perturbed copies rather than the exact templates. Rejected copies receive the rejection penalty and are included when computing mean reward. The PPO controller changes after every round, so account for the round labels when comparing results. Return no code, explanation, or text outside the required JSON object.
INPUT: The user message is a JSON object with the following structure:

```
{
  "task": "BallCatcher-Pitch",
  "description": {TASK_DESCRIPTION},
  "task_context": {
    "parameter_defaults": {PARAMETER_DEFAULTS},
    "parameter_columns": {PARAMETER_COLUMNS},
    "parameter_rows": {PARAMETER_ROWS},
    "parameter_encoding": {PARAMETER_ENCODING}
  },
  "round": {ROUND},
  "search_state": {
    "completed_rounds": {COMPLETED_ROUNDS},
    "completed_ppo_iterations": {COMPLETED_PPO_ITERATIONS},
    "population_size": {POPULATION_SIZE},
    "template_count": {NUM_TEMPLATES},
    "copies_per_template": {COPIES_PER_TEMPLATE},
    "remaining_seconds": {REMAINING_SECONDS},
    "best_observed_design": {BEST_EMBODIMENT},
    "best_observed_reward": {BEST_REWARD}
  },
  "base_embodiment": {BASE_EMBODIMENT},
  "perturbation_std_fraction": {PERTURBATION_STD},
  "template_statistics_columns": {STATISTICS_COLUMNS},
  "template_statistics_rows": {STATISTICS_ROWS},
  "history_embodiment_encoding": "Each historical embodiment cell contains
     [parameter_index, value] pairs; omitted indices equal base_embodiment.",
  "diagnostic_definitions": "Rates are fractions of evaluated copies with the
     event at least once during their first episode; rejected copies contribute
     zero events and zero length. catch_rate requires accumulated dwell and
     retention; goal_entry_rate includes brief contact with the reward region.
     Missing signals are omitted."
}
```

TASK: Propose {NUM_TEMPLATES} new embodiment templates that are likely to maximize mean episodic return after independent perturbation. Use the task description, embodiment parameter semantics, current PPO training state, best result observed so far, historical template statistics, rejection statistics, and available episode diagnostics.

OUTPUT REQUIREMENTS: Return JSON only: no extra text, code fences, comments, or reasoning. Return one JSON object matching this schema:

```
{
  "embodiments": [
    [<exactly {EMBODIMENT_DIM} numeric values>],
    ...
    [<exactly {EMBODIMENT_DIM} numeric values>]
  ]
}
```

IMPORTANT INFORMATION:

- The output must contain exactly {NUM_TEMPLATES} embodiment templates.
- Every template must contain exactly {EMBODIMENT_DIM} values in parameter-index order.
- Every value must satisfy its supplied bounds and integer constraint.
- The optimization target is mean episodic return across the perturbed copies of each template.
- Rejected embodiments receive the rejection penalty and reduce the template's mean reward.
- Historical rewards come from different PPO training rounds; their round labels must therefore be considered.
- Only valid JSON matching the specified schema is accepted.

Figure 29: Prompt used by the PPO+LLM outer loop on BallCatcher-Pitch. Placeholders of the form {VAR} are replaced with the corresponding values at query time.

Figure 30 compares training and evaluation return against the number of PPO iterations. Over a comparable number of iterations (68 and 59 for PPO+LLM, 68–71 for PPO+NGOpt), PPO+NGOpt improves from ≈0.08 to 0.43 eval return, while both PPO+LLM runs stay at or near zero throughout training and at every evaluation. Between 27% and 36% of LLM-designs generated were unusable. The best design was updated 3 and 4 times in the two PPO+LLM runs, and between 1 and 15 times across the PPO+NGOpt seeds. Figure 31 shows every distinct evaluated design for seed 0 of each method.

Run	Stopped at	PPO iters	Design rounds	Final eval	Peak eval
PPO+NGOpt (5 seeds)	4 h (budget)	70	69	0.43 ± 0.33	0.59 ± 0.46

Table 3: PPO+LLM (Gemma 4 31B, 16 templates per round, perturbation std 0.05, 16,384 environments) on BallCatcher-Pitch, per seed, against the PPO+NGOpt baseline.

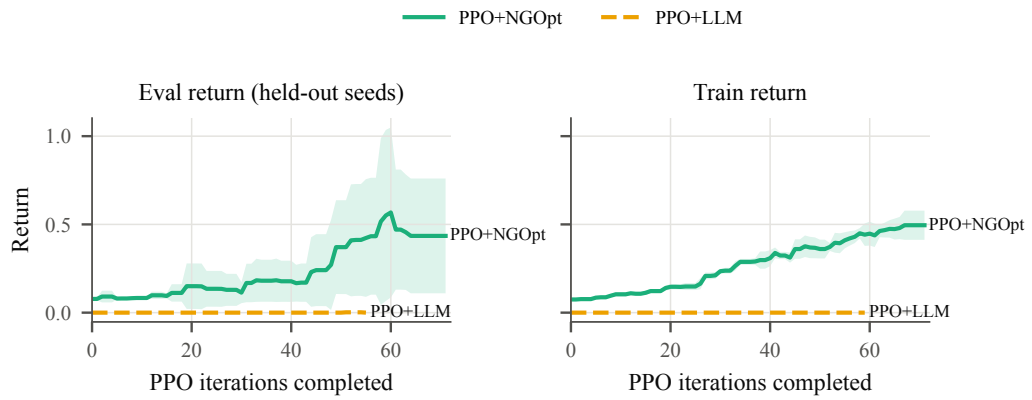

Figure 30: **PPO+LLM vs. PPO+NGOpt on BallCatcher-Pitch**, plotted against PPO iterations because both algorithms run the same PPO inner loop and the PPO+LLM runs stopped early (×). PPO+NGOpt's return (5-seed mean ± std) rises from ≈0.08 to 0.43 at evaluation, whereas both PPO+LLM runs stay at or near zero in training (≤0.005) and at every evaluation (≤0.02), over a comparable number of PPO iterations (68 and 59 vs. 68–71).

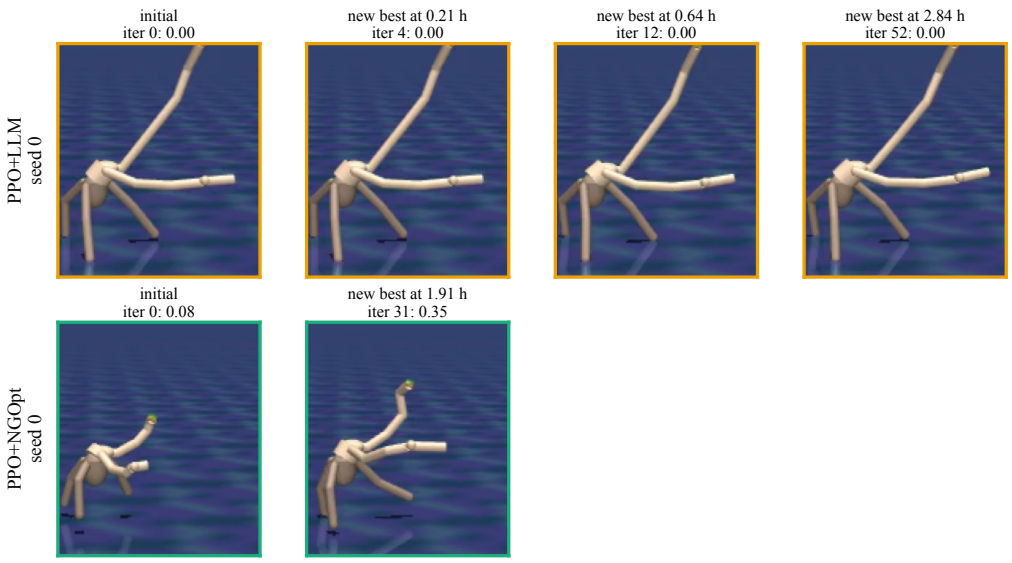

Figure 31: **Every distinct evaluated design** (seed 0, illustrative) for PPO+LLM (amber frames) and PPO+NGOpt (aqua frames): the initial design and the design after each time a better-scoring candidate replaced it (PPO+LLM at 0.21, 0.64 and 2.84 h; PPO+NGOpt once, at 1.91 h), each taken from the first frame of the next evaluation video and shown at a common scale (title: PPO iteration and eval return of that evaluation).

E API ILLUSTRATION

Figure 32 illustrates the inner workings of our API: It functions as an interface between co-design algorithm and the environments while minimizing deviations to the established Gymnasium structure, thereby ensuring broad compatibility across established frameworks.

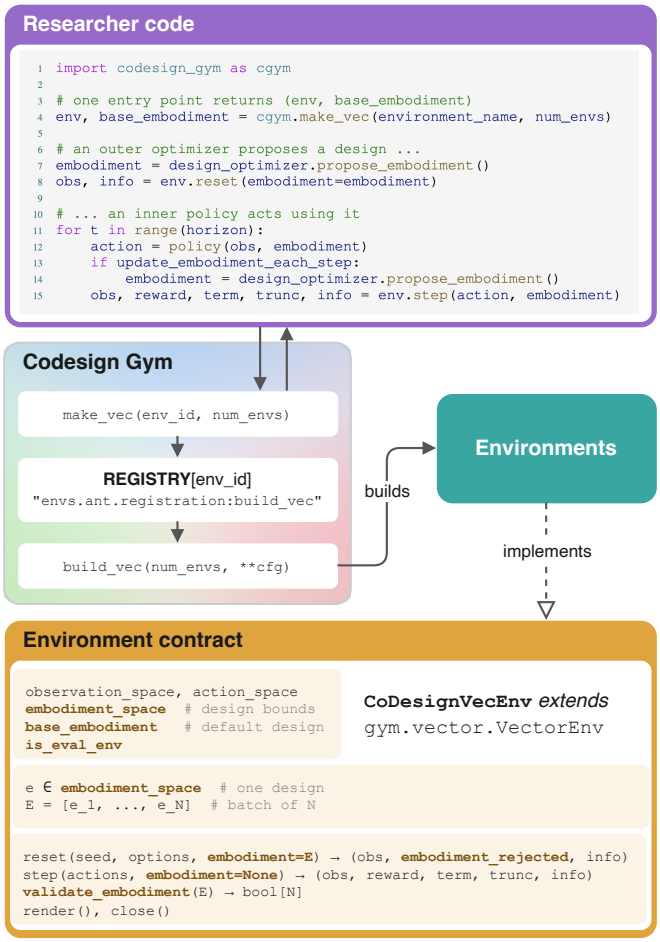

Figure 32: Co-design Gym API flowchart

F CO-DESIGN PROBLEMS CONTAIN REINFORCEMENT LEARNING PROBLEMS

By Proposition 5, we formalize the final discussions in Section 3 and what Figure 3 visualizes: policy optimization problems—that is, policy optimization problems—are a subset of co-design problems.

Proposition 5. *Let $\mathcal{I} = (\mathbf{E}, \mathbf{\Pi}, \mathcal{E}, R)$ be a problem, admitting a set of decision problems $\mathbf{M}_{\mathcal{I}}$. The map $\phi : \mathbf{M}_{\mathcal{I}} \to \mathbf{I}$ s.t. $M_{\mathbf{e}} \mapsto (\{\mathbf{e}\}, \Pi_{\mathbf{e}}, \mathcal{E}|_{\{\mathbf{e}\}}, R|_{\{\mathbf{e}\}})$ is an injective, value-preserving map from decision problems into problems, whose image is exactly the subproblems of $\mathcal{I}$ with a single design. Solving the decision problem $M_{\mathbf{e}_0} \in \mathbf{M}_{\mathcal{I}}$ is therefore solving the problem $\phi(M_{\mathbf{e}_0}) = (\{\mathbf{e}_0\}, \Pi_{\mathbf{e}_0}, \mathcal{E}|_{\{\mathbf{e}_0\}}, R|_{\{\mathbf{e}_0\}}) \subseteq \mathcal{I}$, and yields value $J^\star(\mathbf{e}_0) \leq J^{\star\star}(\mathcal{I})$.*

Proof. ϕ is injective and admits an inverse ϕ^{-1} on the set of single-design subproblems of $\mathcal{I}$ ($\mathbf{I}_S$), $\phi^{-1}|_{\mathbf{I}_S} : (\{\mathbf{e}\}, \Pi_{\mathbf{e}}, \mathcal{E}|_{\{\mathbf{e}\}}, R|_{\{\mathbf{e}\}}) \mapsto M_{\mathbf{e}}$. Let $M_{\mathbf{e}} \in \mathbf{M}_{\mathcal{I}}$. The value of any decision problem is a supremum over a single design, so $J^{\star\star}(\phi(M_{\mathbf{e}})) = J^{\star}(\mathbf{e})$; and a subproblem of $\mathcal{I}$ lies in the image iff its design set is a singleton. Since $\{\mathbf{e}\} \subseteq \mathbf{E}$, $\Pi_{\mathbf{e}} \subseteq \mathbf{\Pi}$, and $\phi(M_{\mathbf{e}})$ carries the restrictions of $\mathcal{E}$ and R, we have $\phi(M_{\mathbf{e}}) \subseteq \mathcal{I}$, so $J^{\star\star}(\phi(M_{\mathbf{e}})) \leq J^{\star\star}(\mathcal{I})$ by Proposition 4. □

Containment would carry no weight if the choice of embodiment made no difference to what is achievable. Two conditions rule this out, and both are weak: the degenerate case they exclude is one in which every admissible embodiment is exactly as good as every other. Write $\mathrm{gap}(\mathbf{e}_0; \mathcal{I}) = J^{\star\star}(\mathcal{I}) - J^{\star}(\mathbf{e}_0)$ for the *design gap* of a fixed embodiment $\mathbf{e}_0$.

Proposition 6. *, Let $\mathcal{I}$ be a task and $\mathbf{e}_0 \in \mathbf{E}$.*

(i) If $J^{\star}$ is constant on $\mathbf{E}$, then $\mathrm{gap}(\mathbf{e}, \mathcal{I}) = 0$ for every $\mathbf{e}$: every embodiment is optimal and the design decision carries no information.

(ii) $\mathrm{gap}(\mathbf{e}_0; \mathcal{I}) > 0$ if and only if $J^{\star}(\mathbf{e}_0) < \sup_{\mathbf{e}} J^{\star}(\mathbf{e})$, and this requires $J^{\star}$ to be non-constant.

Proof. Both are immediate from $J^{\star\star}(\mathcal{I}) = \sup_{\mathbf{e}} J^{\star}(\mathbf{e})$. □

The containment therefore has content whenever embodiment matters at all – when even one design differs from the rest in $J^{\star}$ – and adopting the larger problem pays whenever the conventional embodiment is not already an optimal one. Each fails only in a narrow circumstance. The first requires the admissible bodies to be interchangeable for the task, so that nothing about the design space is worth choosing. The second requires the embodiment fixed in advance to have been exactly the right one, chosen without any means of confirming it. Where the second holds, the gain from co-design is the gap itself.

Two things follow. First, the gap is fixed before learning begins: no amount of data, compute, or optimizer quality reduces it, because the restriction is on which solutions are available rather than on how well they are found. Second, it is invisible from inside. A method restricted to $\mathbf{e}_0$ never evaluates $M_{\mathbf{e}}$ for any $\mathbf{e} \neq \mathbf{e}_0$, so it cannot estimate, bound, or report its own design gap – not even from a converged learning curve. Whether either condition holds is thus a property of the task that no fixed-design method can measure, and the size of the gap is an empirical quantity rather than one to be assumed. Measuring it requires a benchmark whose design space is part of the problem specification.