A Implementation Details

This section provides implementation details for the algorithms that we evaluate: LP3 [LS], LP3 [MO-MPO], and LP3 [MO-MPO-D]. The results for MetaL, D4PG-Lagrangian, and RewardShaping-0.1 in Table 1 were taken from Calian et al. [5].

We use an asynchronous off-policy actor-learner setup. In this setup, actors fetch policy parameters from the learner and act in the environment, writing transitions to the replay buffer. The learner uses the transitions in the replay buffer to update the preference, policy, and Q-function networks (and, for LP3 [MO-MPO-D], the temperature network). We implement this setup on top of the open-sourced MPO and MO-MPO implementations in Acme [37][3], an open-source framework for distributed RL. To make learning more stable, we maintain a target network for each trained network; we use these target networks for computing gradients, and update them after every 200 gradient steps. We use Adam [41] for optimization, with a learning rate of 10^{-4} unless otherwise specified.

The learned preference is either a single value or a categorical distribution. The policy and Q-functions are implemented as feed-forward neural networks. The policy π_θ outputs a Gaussian distribution with a diagonal covariance matrix. We found that adding layer normalization followed by a hyperbolic tangent ($\tanh$) to the first layer of the policy and Q-function networks improves stability of learning. For the RWRL, Safety Gym, and robot tasks, we train a separate Q-function per objective, with a shared first hidden layer. For the humanoid tasks, the Q-values for action norm cost are computed exactly from the actions, so these "Q-values" have a discount factor of zero.

The hyperparameters we used for our experiments are reported in Table 2.

Gathering data. We gather data for the replay buffer via the actors. We use 32 actors for training. When the policy π_θ is preference-conditioned, at the start of each episode, each actor first samples a preference $\boldsymbol{\epsilon}'$ from the preference distribution p, and then acts according to $\pi_\theta(a|s, \boldsymbol{\epsilon}')$ until the end of the episode. For the next episode, the actor repeats this, sampling a new preference.

Evaluation. For the RWRL and humanoid tasks, we evaluate each trained policy on the same 100 randomly-initialized environments. For the Safety Gym tasks, we evaluate on the same 500 randomly-initialized environments. Table 3a reports how many actor steps policies are trained for, before evaluation. In evaluation, we use a deterministic policy π_θ by using the mean of the Gaussian distribution over actions. When the policy is conditioned on preferences, at the beginning of each episode during evaluation, we first sample a preference $\boldsymbol{\epsilon}$ from the preference distribution, and then execute the mean of $\pi_\theta(a|s, \boldsymbol{\epsilon})$, sticking with the same sampled preference for the entire episode.

B Experimental Domains

B.1 Toy Domain

As a toy domain, we use a multi-arm bandit with a continuous action, $a \in \mathbb{R}$. We use either the Schaffer or Fonseca-Fleming function to represent the Pareto front. Both functions are $f : \mathbb{R} \mapsto \mathbb{R}^2$, where the two-dimensional output can be interpreted as the reward function $r(a) \in \mathbb{R}^2$. In our experiments, the constraint is on the second objective.

The Schaffer function corresponds to a convex Pareto front, and is defined as

$$f_1(a) = a^2 \quad \text{and} \quad f_2(a) = (a-2)^2 \,.$$

The Fonseca-Fleming function corresponds to a concave Pareto front, and is defined as

$$f_1(a) = 1 - \exp(-(a-1)^2) \quad \text{and} \quad f_2(a) = 1 - \exp(-(a+1)^2) \,.$$

These functions are standard test functions in multi-objective optimization, and are originally defined such that the aim is to minimize them [38]. In RL, the aim is to maximize reward, so we define the reward as the negative of these functions.

For each function and constraint threshold, we trained five policies with LP3 [LS] and LP3 [MO-MPO] with different random initializations. The policy is a MLP with hidden layers of size $(20, 20)$, that outputs a (univariate) Gaussian distribution. We train policies for 75000 learner steps, and then plot the average performance over 100 trials.

[3] Available at github.com/deepmind/acme.

Hyperparameter	Default
Q-function(s): $Q_k(s,a)$ or $Q_k(s,a,\boldsymbol{\epsilon})$	
layer sizes	$(512, 512, 256)$
support	$[-150, 150]$
number of atoms	51
n-step returns	5
discount factor γ	0.99
relabel preferences $\boldsymbol{\epsilon}$	yes
policy network: $\pi_\theta(a\|s)$ or $\pi_\theta(a\|s,\boldsymbol{\epsilon})$	
layer sizes	$(256, 256, 256)$
minimum variance	10^{-12}
maximum variance	unbounded
policy and Q-function networks	
layer norm on first layer?	yes
$\tanh$ on output of layer norm?	yes
activation (after each hidden layer)	ELU
single preference: $\boldsymbol{\epsilon}$	
softplus on output?	yes
initial preference $\boldsymbol{\epsilon}$ (before softplus)	0
Adam learning rate	10^{-5}
preference distribution: $p(\boldsymbol{\epsilon})$	
layer sizes for temp net $\eta_\omega(\boldsymbol{\epsilon})$	$(256, 256, 256)$
support for ϵ_{task}	$[0.1, 0.1]$
support for ϵ_{cost}	$[10^{-5}, 0.15]$
number of atoms	100
MPO / MO-MPO for policy network π_θ	
actions sampled per state	20
default ϵ_k	0.1
KL-constraint on mean, β_μ	10^{-3}
KL-constraint on covariance, β_Σ	10^{-7}
initial temperature η	5
Adam learning rate (dual variables)	10^{-2}
MPO / MO-MPO for preference distribution	
actions sampled per state	20
default ϵ_k	0.1
KL-constraint on policy, β	10^{-7}
initial temperature η	5
Adam learning rate (dual variables)	10^{-2}
training	
batch size	512
replay buffer size	10^6
target network update period	200

Humanoid

Q-function(s): $Q_k(s,a)$ or $Q_k(s,a,\boldsymbol{\epsilon})$	
relabel preferences $\boldsymbol{\epsilon}$	no
preference distribution: $p(\boldsymbol{\epsilon})$	
walk and stand: support for ϵ_{cost}	$[10^{-5}, 0.30]$
walk (4 obj): support for ϵ_{task}	$[0.05, 0.05]$
MPO / MO-MPO for preference distribution	
walk: KL-constraint on policy, β	10^{-6}
run and stand: KL-constraint on policy, β	10^{-8}
walk (4 obj): KL-constraint on policy, β	10^{-8}

Safety Gym (Point)

Q-function(s): $Q_k(s,a)$ or $Q_k(s,a,\boldsymbol{\epsilon})$	
support	[-10, 10]
preference distribution: $p(\boldsymbol{\epsilon})$	
support for ϵ_{cost}	$[10^{-5}, 0.20]$

Robot

preference distribution: $p(\boldsymbol{\epsilon})$	
support for ϵ_{cost}	$[10^{-5}, 0.20]$

RWRL

Q-function(s): $Q_k(s,a)$ or $Q_k(s,a,\boldsymbol{\epsilon})$	
humanoid: relabel preferences $\boldsymbol{\epsilon}$	no
MPO / MO-MPO for preference distribution	
KL-constraint on policy, β	10^{-10}

Walker

Q-function(s): $Q_k(s,a)$ or $Q_k(s,a,\boldsymbol{\epsilon})$	
relabel preferences $\boldsymbol{\epsilon}$	no
preference distribution: $p(\boldsymbol{\epsilon})$	
support for ϵ_{task}	$[0.05, 0.05]$
MPO / MO-MPO for preference distribution	
KL-constraint on policy, β	10^{-8}

Table 2: **Left:** Default hyperparameters for all approaches, with decoupled update on mean and covariance of the policy. **Right:** Hyperparameters for experiments in each domain, that differ from the defaults.

Since Lagrangian-based approaches can be sensitive to the learning rate for the Lagrange multipliers, in these experiments for LP3 [LS] we tried out three different policy and Lagrange multiplier learning rate combinations: 10^{-3} for both, 10^{-3} for policy and 10^{-4} for Lagrange multipliers, and vice versa. None of these enabled LP3 [LS] to find optimal solutions on the concave Pareto front of the Fonseca-Fleming function.

B.2 Humanoid

We use the humanoid run and walk tasks from the DeepMind Control Suite [39].[4] The observation space is 67-dimensional and the action space is 21-dimensional. Observations consist of joint angles, joint velocities, center-of-mass velocity, head height, torso orientation, and hand and feet positions. Actions correspond to joint accelerations; the minimum and maximum action limits are -1 and 1, respectively. Each episode is 1000 timesteps.

[4] Available at github.com/deepmind/dm_control.

Task	Actor Steps
RWRL cartpole and quadruped	600 million
RWRL humanoid and walker	1.2 billion
Humanoid run, walk, stand	500 million
Humanoid run, mid-training	200 million
Humanoid walk (4 obj)	200 million
Point goal, button, push	400 million
Point goal with two constraints	200 million

(a)

Task	Property	Value
cartpole swingup	obs dimension	5
	action dimension	1
	constraint on	`balance_velocity`
	safety coefficient	0.1
humanoid walk	obs dimension	67
	action dimension	21
	constraint on	`joint_angle`
	safety coefficient	0.3
quadruped walk	obs dimension	78
	action dimension	12
	constraint on	`joint_angle`
	safety coefficient	0.2
walker walk	obs dimension	18
	action dimension	6
	constraint on	`joint_velocity`
	safety coefficient	0.1

(b)

Task / Algorithm	Constraint Thresholds
Humanoid	
run	$\beta_{\text{cost}} \in \text{linspace}(-4.0, -1.0, 31)$
walk	$\beta_{\text{cost}} \in \text{linspace}(-2.0, -0.5, 31)$
Safety Gym Point	
goal, button, push	$\beta_{\text{cost}} \in \text{linspace}(-15, -2, 14)$
goal (two constraints)	$\beta_{\text{x}} = -2$
	$\beta_{\text{y}} \in \text{linspace}(-10, 0, 11)$
Robot	
push	$\beta_{\text{cost}} \in \text{linspace}(-40, -2, 20)$
RWRL	
LP3 [MO-MPO-D]	$\beta_{\text{cost}} = -200$
LP3 [LS]	$\beta_{\text{cost}} = -100$
Calian et al. [5]:	
cartpole	$\beta_{\text{cost}} \in \{-70, -90, -115\}$
humanoid	$\beta_{\text{cost}} \in \{-278, -378, -478\}$
quadruped	$\beta_{\text{cost}} \in \{-545, -645, -745\}$
walker	$\beta_{\text{cost}} \in \{-57, -77, -97\}$

(c)

Table 3: (a) The number of actor steps that policies are trained for, before evaluation. (b) Details and settings for the RWRL tasks we use. (c) The constraint thresholds that policies are trained on for each task.

The task reward is a shaped reward for maintaining a horizontal speed (in any direction) of 10 m/s for humanoid run and 1 m/s for humanoid walk. The shaped reward is equal to $\min(h/h^*, 1)$, where h is the speed of the agent and h^* is the target speed, both in m/s. The constrained reward is the negative $l2$-norm of the action vector: $r_{\text{cost}}(s, a) = -\|a\|_2$. This can be thought of as limiting the energy usage of the agent.

We also consider a version of humanoid walk with four objectives. In addition to the task reward and energy usage cost, we add a reward for moving forward and a reward for moving left. Both directional rewards are computed with respect to the velocity of the torso in the egocentric frame. Per-timestep, the sum of the move-forward and move-left rewards is equal to 1 if the agent's speed h is greater than 1 m/s and otherwise is equal to the speed. To compute these two rewards, we compute the angle of movement θ (in degrees, with respect to the forward-axis) and use the ratio $\theta/90$ for the move-left reward, and $\min(h, 1) - \theta/90$ for move-forward. For example, if the agent is moving at a 45° angle, then the move-forward and move-left rewards will both be equal to 0.5 for that timestep.

B.3 Safety Gym Point

We use the level-two point mass tasks from OpenAI Safety Gym: goal, button, and push [22].[5] The action space is 2-dimensional, with minimum and maximum action limits of -1 and 1, respectively. The observation space is 60-dimensional for point goal and 76-dimensional for point button and push. There are 1000 timesteps per episode. At the start of each episode, the entities (i.e., goal, agent, obstacles) are randomly initialized.

[5] Available at github.com/openai/safety-gym.

We use the default task configurations from Ray et al. [22], including how many obstacles are spawned, the regions for spawning, the size of entities, and the type of observations. The only difference is that we use sparse instead of shaped task reward, since the former is more realistic.

A sparse task reward of 1 is given in point goal when the agent enters the goal area, in button when the agent touches the target button, and in push when the agent pushes a box into the goal area. The constrained reward is the negative cumulative cost. In the goal task the agent incurs cost by being in hazardous regions or bumping into vases; in button the agent incurs cost by being in hazardous regions, bumping into gremlins, or touching non-target buttons; in push the agent incurs cost by being in hazardous regions or bumping into pillars. All obstacles are static, except for gremlins, which move in a fixed circular pattern. The agent moves vases when it bumps into them.

In our experiments with two constraints (Sec. 5.3), we use a more difficult variant of the point goal task, with two changes. First, the length of the square area in which entities are spawned is three-quarters of the original length, which makes the density of obstacles higher and thus more difficult for the agent to navigate around. In addition, the radius of the goal region is two-thirds the original radius, which allows for it to be placed in more difficult-to-reach locations, because the random initialization of the scene enforces that there are no collisions between the entities.

B.4 Robot

We implemented a simulated robotics task that is similar to the Safety Gym push task. The setup consists of a Sawyer arm with two blocks in its workspace (Fig. 4). The action space is 5-dimensional: one dimension for how open the gripper is, and four dimensions for Cartesian velocity control of the gripper position—the gripper is always perpendicular to the workspace, so the control is along the xyz axes and rotation. The observation space is 247-dimensional. The observations consist of joint angles, joint velocities, joint torques, gripper pose, gripper velocity, wrist forces and torques, the pose and velocity of each cube, the distance between the gripper pose and pose of each cube, the position of the target, and the previous action taken. There are 400 timesteps per episode.

The task reward is the sum of a sparse reach reward and a sparse push reward. The reach reward is given when the center of the robot's gripper is within 2 cm of the block's position. The push reward is given when the green block's position is within 2 cm of the target. The agent incurs a cost of -1 when the gripper is close to the yellow block, or when the green and yellow blocks are close.

At the beginning of each episode, the target position is randomly sampled uniformly from a region with radius 2 cm, in the far left corner of the workspace (from the robot's point of view). The poses of the yellow and green blocks are randomly sampled to be in the near right corner of the workspace.

B.5 Real-World RL

From the Real-World RL (RWRL) suite [40],[6] we use the same tasks and settings as used in Calian et al. [5]. Table 3b provides the observation dimension, action dimension, constraint-specific reward, and safety coefficient for each task. The safety coefficient defines how challenging the constraint is: a safety coefficient of 0 results in a constraint that is impossible to satisfy, and a safety coefficient of 1 means the constraints will always be satisfied.

We evaluate LP3 [MO-MPO-D] and LP3 [LS] on these RWRL tasks. We obtained the results for MetaL and D4PG Lagrangian (in Table 1) directly from the authors of Calian et al. [5].[7] These algorithms are as follows:

- **D4PG Lagrangian:** This algorithm uses D4PG to train policies to optimize the Lagrangian dual objective. In Calian et al. [5], this algorithm is referred to as RC-D4PG.
- **MetaL:** This algorithm combines D4PG Lagrangian with the use of meta-gradients to update the Lagrange multiplier learning rate throughout training.

[6] Available at github.com/google-research/realworldrl_suite.

[7] We needed to obtain the raw results from the authors, in order to compute the numbers in Table 1 and generate the Pareto plots in Fig. 8. This is because the results reported in Calian et al. [5] are averaged across safety coefficients, whereas in this work we consider a single safety coefficient per task—for each task, we pick one of their safety coefficients that is difficult but possible to satisfy. We also needed the per-seed results to generate the Pareto plots.

One thing to note is that in Calian et al. [5], the Lagrange multipliers for MetaL and D4PG Lagrangian are trained in a different way than in our work—the gradient update for the Lagrange multipliers is computed by sampling per-episode returns for the constraint-specific reward from a separate replay buffer. In contrast, in our LP3 [LS] (which uses MPO to optimize the Lagrangian dual objective), the gradient update is computed using a learned Q-function for the constraint-specific reward. This does not seem to deteriorate performance; in fact, our LP3 [LS] baseline outperforms all three algorithms from Calian et al. [5] on three of the four tasks (Table 1).

B.6 Walker

We use the (planar) walker run task from the DeepMind Control Suite, with a few additional observations and modified rewards. This domain has one task reward and two additional objectives with constraints. The observation space is 27-dimensional and the action space is 6-dimensional. Observations consist of planar orientations of all bodies, torso height, velocity, horizontal velocity, and how upright the torso is. Actions correspond to joint accelerations; the minimum and maximum action limits are -1 and 1, respectively. Each episode is 1000 timesteps.

The task reward is a shaped reward based on the height of the torso. It is equal to $h/1.2$: the torso height h of the agent, divided by the target standing height of 1.2 meters from the stand task. The first constrained reward relates to energy usage; it is the negative $\ell 2$-norm of the action vector: $r_{\text{energy}}(s, a) = -\|a\|_2$. The other constrained reward is move-forward, which is a shaped reward for maintaining a forward horizontal speed of 1 m/s. This is equal to $s/8$: the horizontal speed s of the agent, divided by the target speed of 8 m/s from the run task.

B.7 Evaluation

In our experiments, we trained policies with LP3 [MO-MPO(-D)] and LP3 [LS] for a range of different constraint thresholds, which are reported in Table 3c. We consider the "harder" constraint thresholds (in Fig. 3) to be the 50% lowest-magnitude thresholds for humanoid run and the point tasks, and all thresholds for humanoid walk because its ground-truth Pareto front seems concave.

For RWRL, we train policies with the same constraints as used in Calian et al. [5], and compare the average per-episode task reward and cost with that of the algorithms evaluated in that work.

For the humanoid and point tasks, we train policies on the same set of constraint thresholds. Thus, each algorithm produces a policy trained separately for each different threshold. At the end of training, all policies are evaluated on the same set of randomly-initialized environments. For LP3 [MO-MPO-D] policies, we sample a new preference at the start of each episode.

We plot the results in the same way that Pareto fronts are typically plotted—in terms of each policy's average per-episode task reward and cost (e.g., in Fig. 3).[8] For both axes, higher values are better; a policy is non-dominated if no other point lies both above and to the right of it.

For the robot task, we train policies for two different constraint thresholds, and evaluate performance throughout the training process, rather than only evaluating fully-trained policies, because we are interested in comparing sample efficiency.

For walker run, we train LP3 [MO-MPO-D] policies to learn a portion of the Pareto front, by learning preferences with respect to inequality fitness functions (9) for two out of the three objectives—move forward and energy usage.

C Additional Experiments and Analysis

C.1 Comparison to Multi-Objective RL Algorithms

We first evaluate whether our full algorithm, LP3 [MO-MPO-D], can find good solutions. Since the ground-truth Pareto front is not available in general, we first find an approximate Pareto front for humanoid run by training policies with MO-MPO for a range of preference settings: $\epsilon_{\text{task}} = 0.1$ and

[8] The action norm constraint for the humanoid tasks is per-timestep and there are 1000 timesteps per episode, so a threshold of -2 corresponds to an average per-episode cost of -2000.

ϵ_{norm} at linearly spaced intervals between 10^{-5} and 0.15. The result is shown in the left-most plot in Fig. 6, where policies are evaluated after 500 million actor steps.

Recall that LP3 [MO-MPO-D] trains a preference-conditioned policy in Module 1, and learns a distribution over preferences in Module 2. An ablation of this is to train a preference-conditioned policy for a *fixed* distribution over preferences. One way to do this is to extend MO-MPO to train preference-conditioned policies, as described in Sec. 4.3.1; we call this approach controllable MO-MPO. We used controllable MO-MPO to train preference-conditioned policies with five random initializations, for $\epsilon_{\text{task}} = 0.1$ and ϵ_{norm} sampled uniformly between 10^{-5} and 0.15, analagous to the settings for MO-MPO. This produces a Pareto front that is comparable to MO-MPO's, both in terms of coverage and solution quality (Fig. 6, second from the right). Policies are evaluated after 500M actor steps, like for MO-MPO. Note that controllable MO-MPO is much more computationally efficient than MO-MPO: whereas we must train a separate policy per preference with MO-MPO, we are able to train just one policy with controllable MO-MPO, and condition it on difference preferences in order to obtain different tradeoffs.

It is possible to use linear scalarization, instead of MO-MPO, as the multi-objective RL algorithm for training preference-conditioned policies. So as a baseline, we use MPO [30] to train policies conditioned on linear scalarization weights; we call this baseline controllable MPO. These policies were conditioned on a w_{task} of 1.0 and w_{norm} sampled uniformly between 0 and 0.15. Compared to controllable MO-MPO, we had to train controllable MPO for longer (900M actor steps instead of 500M) in order to reach reasonable task performance, and controllable MPO was still unable to find solutions with low action norm (Fig. 6, second from the left). This is not particularly surprising: Abdolmaleki et al. [6] showed that MO-MPO outperforms MPO with linear scalarization on humanoid run, so one would expect controllable MO-MPO to also outperform controllable MPO. We tried running controllable MPO with higher weights for the action norm, to try and obtain solutions with lower action norm, but these did not show any learning progress within 500M actor steps.

Policies trained with LP3 [MO-MPO-D] for equality constraint thresholds from -4 to -1 also lie on the MO-MPO Pareto front (Fig. 6, right-most). We trained a separate preference-conditioned policy and preference distribution for each unique constraint threshold. This supports that learning the preference distribution simultaneously while training the policy and Q-functions does not negatively impact the final solutions that are found.

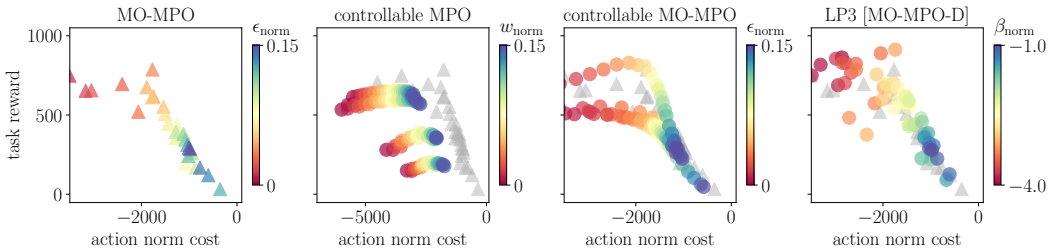

Figure 6: A comparison of the Pareto fronts for humanoid run found by different approaches. For MO-MPO and LP3 [MO-MPO-D], each dot corresponds to a separately-trained policy. For controllable MPO and controllable MO-MPO, five trained policies (with different random initializations) are evaluated by conditioning on a range of linearly-spaced preferences, which correspond to weights or KL-divergence bounds, respectively. The gray triangles denote the MO-MPO Pareto front, for easier comparison. Controllable MO-MPO finds better solutions than controllable MPO, especially for smaller action norm cost. The Pareto front found by LP3 [MO-MPO-D] is on-par with that found by MO-MPO, indicating that LP3 [MO-MPO-D] is able to find solutions with good tradeoffs between the task reward and cost.

C.2 LP3 [LS]: MPO Lagrangian Baseline

The baseline we use in our empirical evaluation is LP3 [LS], which corresponds to using MPO to optimize for the Lagrangian dual objective. We ran this baseline on the level 1 point tasks, using the default settings in OpenAI Safety Gym (including dense reward) so that we can directly compare against the constraint-satisfying policies obtained in Ray et al. [22].[9] Our LP3 [LS] baseline

[9] Since we noticed that our Q-functions underestimate cost (Appendix C.8), we use a cost threshold of 15 for training, rather than the actual cost threshold of 25.

compares favorably with those in Ray et al. [22], which instead combine Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) with Lagrangian relaxation (Fig. 7). Policies were trained with TRPO-Lagrangian and PPO-Lagrangian for 10 million actor steps; after the same number of actor steps, LP3 [LS] obtains on-par or slightly better task reward while meeting the constraints.[10]

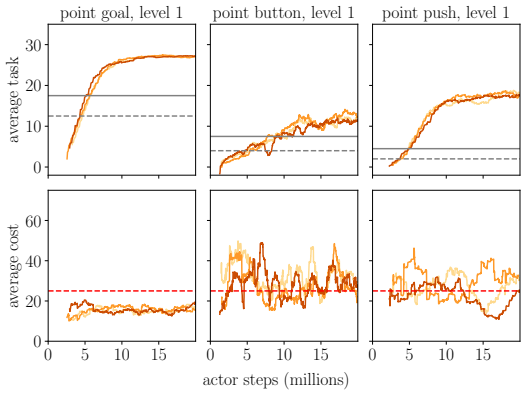

Figure 7: **Top:** The task reward (averaged over episodes) achieved by policies trained with our LP3 [LS] (i.e., MPO-Lagrangian) baseline, over the course of training. The solid and dashed grey lines denote the task reward obtained by PPO- and TRPO-Lagrangian, respectively, after 10 million actor steps in Ray et al. [22]. Our LP3 [LS] baseline achieves similar or better task reward, in the same number of actor steps. Each color is a different random seed. **Bottom:** Our LP3 [LS] baseline meets the constraint of incurring less than 25 expected cumulative cost per episode, indicated by the dotted red line. Each colored line is a different random seed.

C.3 Quality of Solutions: RWRL

For all four RWRL tasks, LP3 [MO-MPO-D] finds policies that are better than those found by the approaches in Calian et al. [5], in terms of *both* task reward and cost. These results are summarized in Table 1, and the performance per trained policy is plotted in Fig. 8. We evaluated LP3 [MO-MPO-D] policies conditioned on the median preference from the learned preference distributions.

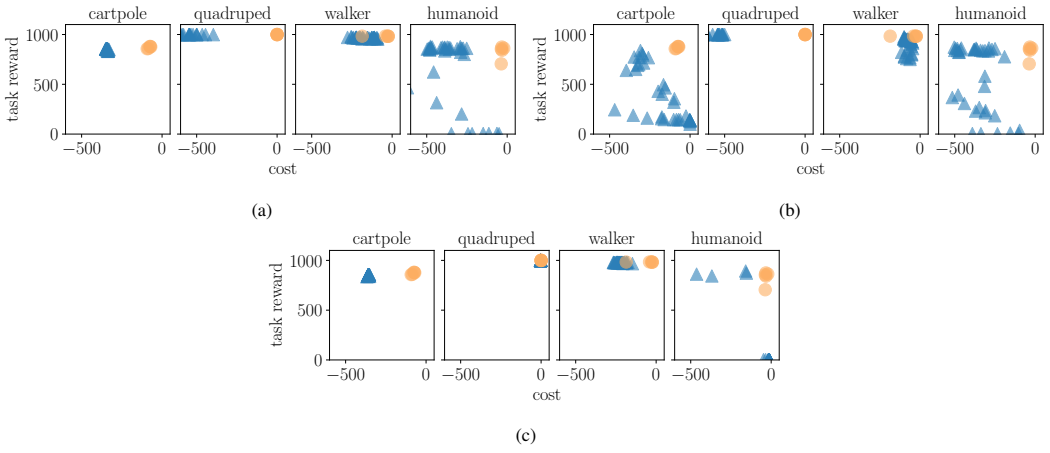

Figure 8: Performance of policies trained on the RWRL tasks, with LP3 [MO-MPO-D] (orange circles) versus (a) MetaL, (b) D4PG Lagrangian, and (c) RewardShaping-0.1. Across all tasks, LP3 [MO-MPO-D] finds policies that are better than those found by the alternate approaches, in terms of *both* task reward and cost. Each dot corresponds to a separately-trained policy. Above and to the right indicates better performance.

[10]This required making LP3 [LS] more sample-efficient, by increasing the Adam learning rate to 10^{-4} and enforcing exactly 16 actor steps per learner step, which slows down the rate at which actors gather data.

C.4 Quality of Solutions: Humanoid and Point

On humanoid run, point goal, and point button, LP3 [MO-MPO-D] and LP3 [LS] find similar solutions for easier constraint thresholds, but as the constraint threshold becomes more difficult to satisfy (i.e., smaller in magnitude), LP3 [MO-MPO-D] does better in four out of five tasks (Fig. 3, 9, 15).

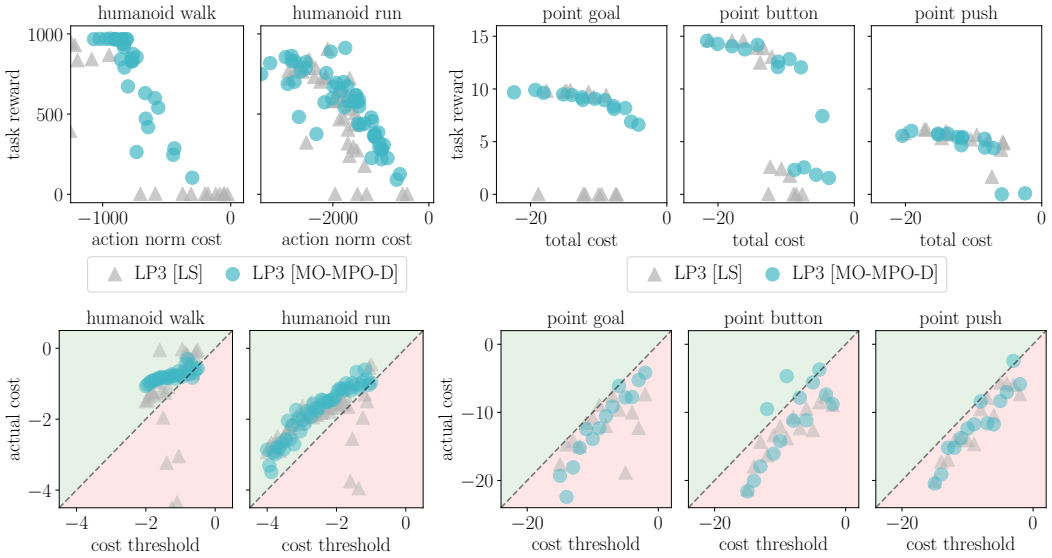

Figure 9: Top row: Across all constraint thresholds, LP3 [MO-MPO-D] performs at least as well as the LP3 [LS] baseline. In these plots, for both axes, higher values are better. Each dot corresponds to a separate policy trained for a particular constraint threshold. Bottom row: LP3 [MO-MPO-D] and LP3 [LS] perform comparably in learning policies that satisfy the constraints. Policies that lie in the red region (below the dotted line) violate the constraint.

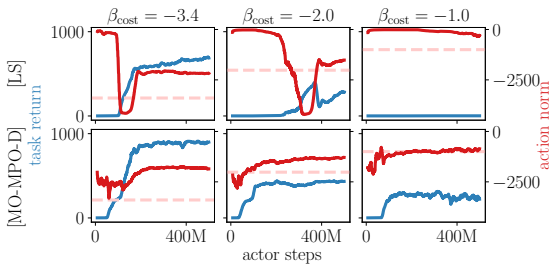

Figure 10: For humanoid run, these learning curves show how LP3 [LS] (top) can only focus on improving one objective at a time: when LP3 [LS] starts learning the task, the cost increases significantly and violates the constraint. In contrast, LP3 [MO-MPO-D] (bottom) is able to improve both objectives at once, which leads to better sample efficiency. The dashed lines indicate the constraint threshold.

The reason LP3 [MO-MPO-D] outperforms LP3 [LS] may be because when two objectives are conflicting, such as the task reward and action norm cost in humanoid run, LP3 [MO-MPO-D] is able to optimize the policy for both at once, whereas LP3 [LS] cannot (Fig. 10).

Learning a Set of Constraint-Satisfying Policies. For humanoid stand and run, we run LP3 [MO-MPO-D] with the inequality fitness function, (9). This successfully learns a portion of the Pareto front that satisfies the constraint (Fig. 11). For these plots, we evaluated the policy conditioned on preferences at evenly-spaced percentiles of the learned distribution.

Note that there is some variation in the learned Pareto front across the different seeds for humanoid run. This is because in humanoid run, task reward is given for running in *any* direction. So, there are many possible running gaits that an agent could adopt to achieve high task reward. Since LP3 [MO-MPO-D] trains a single policy to represent a portion of the Pareto front, this is limited to selecting one of the possible strategies. The policy's initialization impacts which strategy it ends up finding,

and the resulting Pareto front. In contrast, in stand there is less variation across strategies, and thus also less variation in the Pareto front across seeds.

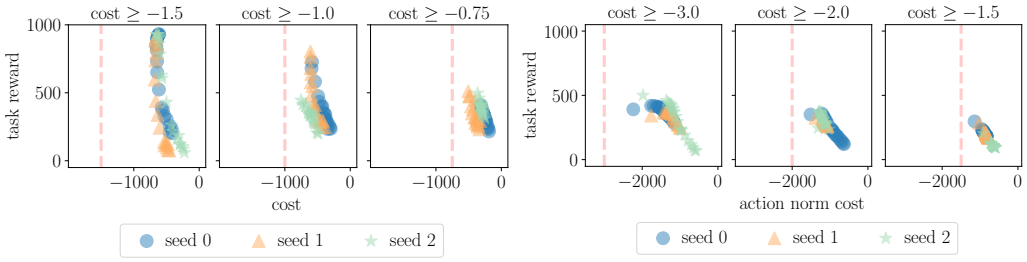

Figure 11: For humanoid stand (left) and run (right), we LP3 [MO-MPO-D] to train preference-conditioned policies, each of which captures a *set* of constraint-satisfying nondominated policies, i.e., a portion of the Pareto front. This is useful when the constraint is soft or not known exactly a priori. These plots are created by evaluating each fully-trained policy conditioned on preferences at evenly-spaced percentiles of the learned preference distribution. The dashed lines indicate the constraint threshold.

C.5 Constraint Satisfaction

LP3 [MO-MPO-D] always finds policies that satisfy the constraints for the humanoid tasks, whereas the Lagrangian baseline occasionally violates the constraint. With longer training, these violations might disappear, but we do not expect the task reward to improve. These violations are due to the relatively unstable behavior of the Lagrangian baseline: when the constraint starts to be violated, this leads to a gradual increase in the Lagrange multiplier, and it takes some time for this to propagate into changes in the policy. This can be seen empirically in Fig. 10 (left plot, top row) where the average cost dips below (i.e., violates) the constraint threshold, indicated by the dotted line.

For the point tasks, the policies found by both LP3 [MO-MPO-D] and the LP3 [LS] baseline slightly violate the constraints (Fig. 9, bottom row); this is due to Q-function underestimates for the cost, rather than a limitation of the policy optimization (Appendix C.8).

C.6 Scaling to Multiple Constraints

As mentioned in the main paper, for the walker run task with one task reward and two constraint-specific rewards, we use LP3 [MO-MPO-D] to train policies to learn a portion of the Pareto front. We do the same for humanoid walk, that has two rewards without constraints and two constraint-specific rewards. For each of the two settings of constraints per task, we trained three policies with LP3 [MO-MPO-D], each initialized using a different random seed. Fig. 12 and Fig. 13 show the performance of all fully-trained policies from these experiments. All policies learn to satisfy both constraints, and overall the learned Pareto fronts are relatively consistent across seeds.

These plots are created by evaluating each fully-trained policy conditioned on preferences at evenly-spaced percentiles of the learned preference distribution for either the move or energy usage objectives, combined with the median preference for the other(s).

We also investigated how well LP3 [MO-MPO-D] scales to multiple constraints for a more difficult version of point goal (see Sec. B.3). There are two constraints: one on the cost of crossing over hazards, and one on bumping into vases. For each type of cost, we fix its threshold to -2 and vary the other between -10 and 0. The baseline fails to obtain task reward when the constraint for hazards is fixed at the challenging threshold of -2, whereas LP3 [MO-MPO-D] finds a range of solutions (Fig. 14, right).

C.7 Multiple Random Seeds

Humanoid and Point To verify that the empirical results we observed in Sec. 5.3 are statistically significant, we used LP3 [MO-MPO-D] and LP3 [LS] to train policies for the same constraint threshold, starting from five random initializations. We evaluated this for four constraint thresholds per task for humanoid and point, selected from the ranges in Table 3c. Results are shown in Fig. 15.

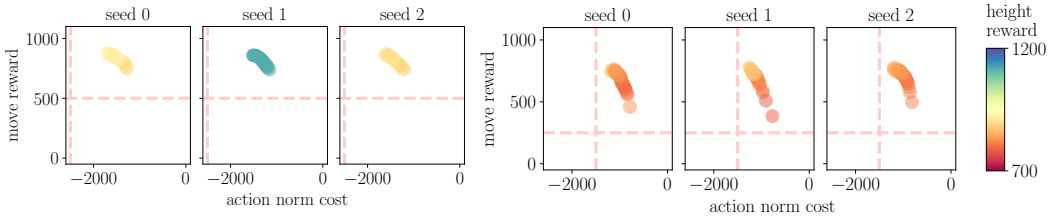

Figure 12: For the walker run task, with three objectives (and constraints on two of them), we use LP3 [MO-MPO-D] to train policies that learn a portion of the Pareto front. We train policies for two settings of constraints, a lower bound of 500 on move-forward return per episode and -2.5 on per-timestep energy usage (left), and a lower bound of 250 on move-forward return per episode and -1.5 on per-timestep energy usage (right). These plots show the resulting Pareto front for each random seed. All policies learn to meet the constraints, and there is relatively little variation across seeds. The dashed lines indicate the constraint threshold.

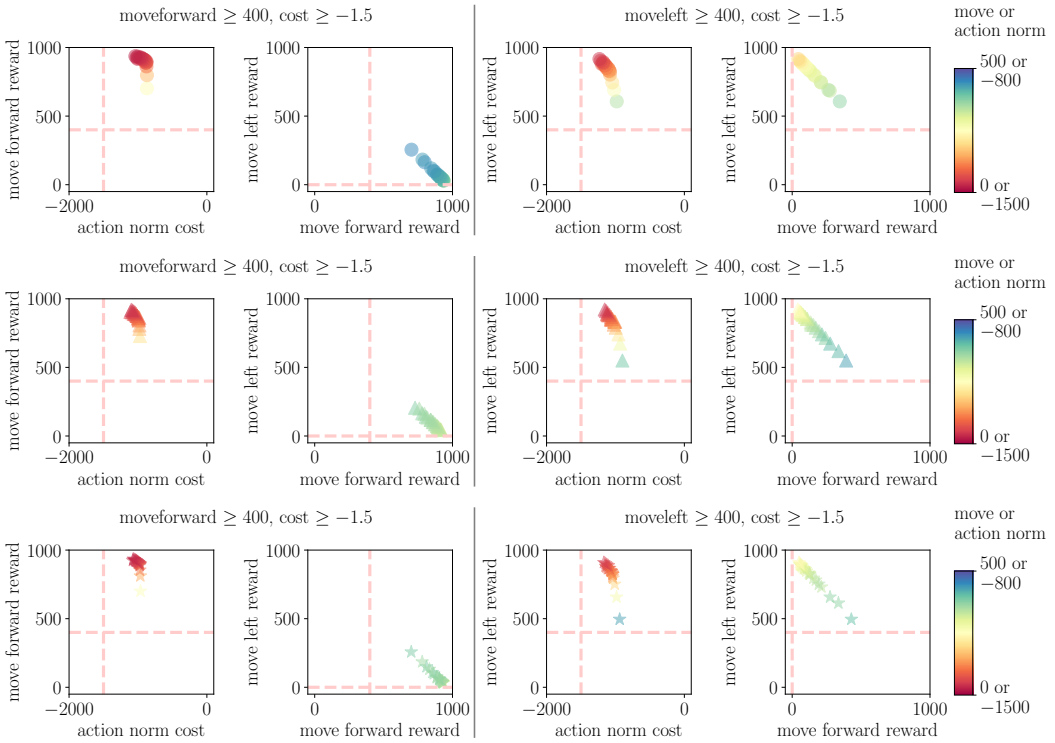

Figure 13: For the humanoid walk task, with four objectives (and constraints on two of them), we use LP3 [MO-MPO-D] to train policies that learn a portion of the Pareto front. We train policies for two settings of constraints: a bound of 400 on either move-forward or move-left return per episode and -1.5 on per-timestep energy usage. These plots show the resulting Pareto front for each random seed. All policies learn to meet the constraints, and there is relatively little variation across seeds. The dashed lines indicate the threshold.

LP3 [MO-MPO-D] achieves significantly higher task reward than the baseline for the harder-to-satisfy constraint thresholds in four out of the five tasks. For the other constraint thresholds, both approaches achieve similar task reward. In terms of satisfying constraint thresholds, our approach performs on-par with or better than the baseline.

As mentioned, the slight violation of constraint thresholds for the Safety Gym point tasks is due to underestimation of Q-values, rather than a limitation of the policy improvement method; this is described in the following subsection.

Robot We ran LP3 [MO-MPO-D] and LP3 [LS] for the robot push task, for five random seeds per threshold. Consistent with the results in the main paper, we see that LP3 [MO-MPO-D] and LP3 [LS] converge to similar task performance, but LP3 [LS] takes about twice as long to reach similar

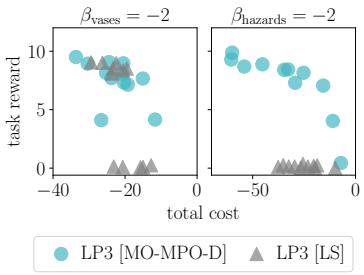

Figure 14: For point goal with two constraints, LP3 [MO-MPO-D] finds better policies than the LP3 [LS] baseline, especially when the constraint on the hazards cost is hard to satisfy (right plot).

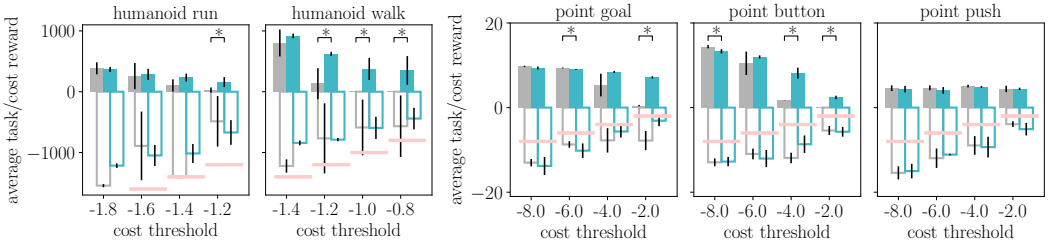

Figure 15: LP3 [MO-MPO-D] (in aqua) achieves significantly higher task reward than the LP3 [LS] baseline (in gray) for the harder-to-satisfy constraint thresholds in four out of five tasks for humanoid and point. In terms of satisfying constraint thresholds, LP3 [MO-MPO-D] performs on-par with or better than the baseline. Each bar shows either the average task reward (solid bars) or cost (clear bars) per episode, averaged over five seeds. For both, higher is better. The error bars refer to standard deviation. The red lines denote the constraint thresholds: clear bars above the constraint threshold indicate that the constraint is met. Asterisks denote $p \leq 0.05$, computed via Welch's t-test.

performance (Fig. 16). LP3 [MO-MPO-D] also trains policies that satisfy the constraint earlier in learning, compared to those trained by LP3 [LS].

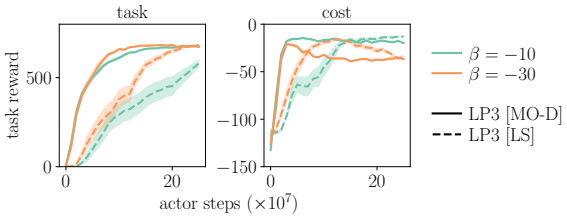

Figure 16: Learning curves for the robot push task, where β denotes the constraint threshold. The LP3 [LS] baseline takes about twice as long to converge to similar task performance as LP3 [MO-MPO-D]. LP3 [MO-MPO-D] also trains policies that more quickly learn to satisfy the constraints. The shaded area denotes standard error across five seeds.

C.8 Q-function Estimates

We noticed that for the point tasks, both LP3 [MO-MPO-D] and the Lagrangian baseline train policies that slightly exceed the constraints (Fig. 9, bottom row). After digging deeper, we realized that this results from suboptimal Q-function learning, rather than suboptimal policy optimization. The learned Q-values for the cost consistently underestimate the actual cost incurred (Fig. 17a). The policy optimization finds a policy that meets the constraints, assuming that the learned Q-values for the cost are accurate (Fig. 17b).

Thus, better constraint satisfaction can be obtained by improving policy evaluation, so that estimated Q-values are more accurate—this is orthogonal to our proposed approach, which is focused on policy improvement.

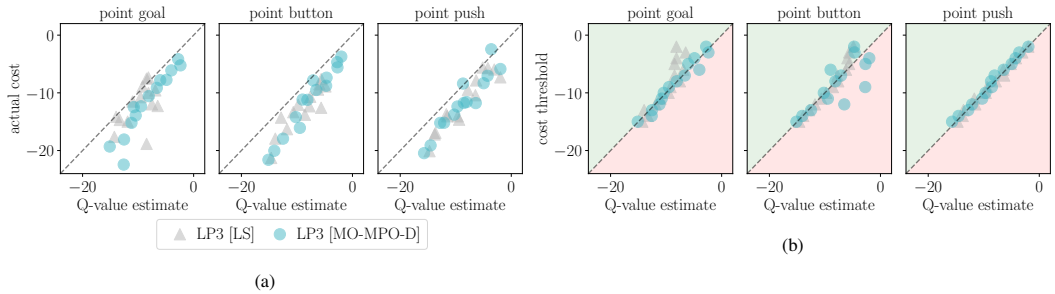

Figure 17: In the plots above, the Q-value estimate is obtained by averaging the Q-values for 100 batches of (s, a) pairs from the replay buffer, after 200M actor steps of training. Each point corresponds to a separately-trained policy, for a different constraint threshold. (a) The learned Q-function for cost underestimates the cost per episode. The actual cost is obtained by evaluating the deterministic policy after 200M actor steps, on the same 100 randomly-initialized environments. (b) For both approaches, almost all trained policies satisfy the constraint (i.e., are on or above the dotted line), according to the learned Q-values.

C.9 Ablation Study on Preference Relabeling

Recall that when we train preference-conditioned policies, as in LP3 [MO-MPO-D], we relabel preferences. This means that for each learning step, we augment the states of transitions sampled from the replay buffer with preferences sampled from the current preference distribution (Sec. 4.3.1). This is motivated by prior work, that shows relabeling goals improves sample-efficiency of learning [33]. In this context, relabeling preferences allows our algorithm to learn from off-policy data that is collected with a much different preference distribution than the current one.

To investigate the utility of preference relabeling, we run an ablation study with LP3 [MO-MPO-D] on the RWRL tasks. Across the four RWRL tasks, we find that relabeling preferences leads to improved performance on cartpole, slightly improved performance on walker, on par performance on quadruped, and slightly worse performance on humanoid. Although in this ablation study, preference relabeling did not significantly help performance across all tasks, our hypothesis is that in a regime that requires more sample-efficient learning (e.g., learning from scratch on a real robot), relabeling preferences will have more benefit, because it makes it more feasible to learn from highly off-policy data.

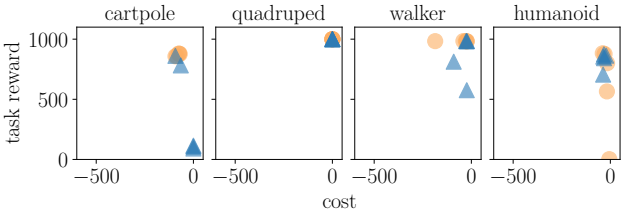

Figure 18: In the RWRL tasks, for most tasks relabeling preferences (orange circles) results in better or on-par performance compared to not relabeling (blue triangles). These plots show five random seeds per task, separately plotted.

D Connection to Existing Work

A main contribution of this work is our proposed framework called LP3, that takes a multi-objective perspective to tackling constrained RL problems (Sec. 4.2). We showed that Lagrangian-relaxation-based algorithms are a special case of this framework. LP3 [MO-MPO-D] is a particular instantiation of this framework, that solves a CMO-MDP by finding a *set* of constraint-satisfying policies. LP3 [MO-MPO-D] consists of training a preference-conditioned policy in parallel with learning a distribution over preferences that leads to constraint satisfaction. There are also other possibilities for finding a set of constraint-satisfying policies, that fit within our proposed framework and relate to existing work.

For example, one could keep a discrete set of preferences and train a separate policy per preference in this set, rather than training a single preference-conditioned policy as LP3 [MO-MPO-D] does. The set of preferences could be either static or learned. If it is static, then at the end of training, the individually-trained policies can be either accepted or rejected depending on constraint satisfaction. This approach is computationally wasteful though, because it fully trains policies that violate the constraints. This downside could be addressed by adjusting the set of preferences during training, to prefer constraint-satisfying policies.

Using a learned discrete set of preferences is similar to the approach taken in Xu et al. [29]. That work optimizes for the hypervolume of the Pareto front, but could be extended to additionally optimize for meeting a specified constraint. A downside of this approach, compared to LP3 [MO-MPO-D], is that training a separate policy per preference quickly becomes computationally burdensome for larger numbers of preferences. In addition, in some situations, training a single preference-conditioned policy may improve learning, because training over multiple preferences can enable more diverse data collection and lead to increased exploration.

E Algorithmic Details

In this section, we first give more detail on how we extend MO-MPO to train preference-conditioned policies—we call this algorithm controllable MO-MPO (as mentioned in Sec. C.1). LP3 [MO-MPO-D] combines controllable MO-MPO with a learned preference distribution.

Then, we describe how to learn the preference policy and provide an algorithm box for the policy improvement step. We also describe the overall procedure we use to collect data and learn from the data.

E.1 Controllable MO-MPO

We introduce controllable MO-MPO, which extends MO-MPO to learn preference-conditioned policies. We use this for Module 1 of LP3 [MO-MPO-D] (within our framework, Fig. 1), in order to learn a single policy for a distribution of preferences. Note that any other multi-objective RL algorithm capable of training preference-conditioned policies could be used for this step.

An overview of MO-MPO is given in Sec. 3.2. Recall that MO-MPO can only train policies $\pi(a|s)$ for a single preference $\boldsymbol{\epsilon}$. In contrast, controllable MO-MPO trains a single policy $\pi(a|s, \boldsymbol{\epsilon})$ for a range of preferences. Extending MO-MPO to controllable MO-MPO requires making both the policy and Q-functions preference-conditioned, replacing the scalar temperature with a preference-conditioned temperature network, as well as modifying the underlying MO-MPO optimization principles. We also introduced relabeling of preferences, as described in Sec. 4.3.1.

Controllable MO-MPO is a policy iteration algorithm with two steps:

- Policy evaluation: learn preference-conditioned Q-functions for all objectives.
- Policy improvement: improve the preference-conditioned policy according to Q-functions.

The policy evaluation step is described in Sec. 4.3.1. Here we provide more detail on the policy improvement step. Some of the information provided in Sec. 4.3.1 is repeated, so that the description here is self-contained.

Preference-conditioned policy improvement. The policy improvement step assumes a state distribution $\mu(s)$, per-objective Q functions $Q_k^{\text{old}}(s, a, \boldsymbol{\epsilon})$, the current policy $\pi_{\text{old}}(a|s, \boldsymbol{\epsilon})$, and the current preference distribution $p_{\text{old}}(\boldsymbol{\epsilon})$. It consists of two sub-steps: 1) finding per-objective improved action distributions, and 2) distilling these distributions into a new preference-conditioned policy via supervised learning.

Finding per-objective distributions: To find per-objective improved action distributions $q_k(a|s, \boldsymbol{\epsilon})$, we optimize the following RL optimization problem for each objective:

$$\begin{aligned} \max_{q_k} \quad & \mathbb{E}_{p_{\text{old}}(\boldsymbol{\epsilon})\mu(s)}\Big[\int_a q_k(a|s, \boldsymbol{\epsilon})\, Q_k^{\text{old}}(s, a, \boldsymbol{\epsilon})\, \mathrm{d}a\Big] \\ \text{s.t.} \quad & \mathbb{E}_{\mu(s)}\Big[\mathrm{KL}(q_k \| \pi_{\text{old}}(a|s, \boldsymbol{\epsilon}))\Big] < \epsilon_k \quad \forall\, \boldsymbol{\epsilon} \sim p_{\text{old}}(\boldsymbol{\epsilon})\,. \end{aligned} \tag{10}$$

We can solve this problem in closed form to obtain

$$q_k(a|s,\boldsymbol{\epsilon}) \propto \pi_{\text{old}}(a|s,\boldsymbol{\epsilon}) \exp\left(\frac{Q_k^{\text{old}}(s,a,\boldsymbol{\epsilon})}{\eta_{\omega_k}(\epsilon_k)}\right), \tag{11}$$

where $\eta_{\omega_k}(\boldsymbol{\epsilon}_k)$ is a preference-dependent temperature function for objective r_k, and is parameterized by ω_k. This temperature function is obtained by minimizing the following dual function:

$$g(\omega_k) = \mathbb{E}_{p_{\text{old}}(\boldsymbol{\epsilon})\mu(s)}\Big[\eta_{\omega_k}(\epsilon_k)\Big(\epsilon_k + \log \int_a \pi_{\text{old}}(a|s,\boldsymbol{\epsilon}) \exp\left(\frac{Q_k^{\text{old}}(s,a,\boldsymbol{\epsilon})}{\eta_{\omega_k}(\epsilon_k)}\right) \mathrm{d}a\Big)\Big]. \tag{12}$$

In practice, we maintain a single function $\eta_\omega(\boldsymbol{\epsilon})$ with shared parameters ω for all objectives.

The optimization problem in (10) generalizes the one in MO-MPO, which assumes fixed KL-constraints $\boldsymbol{\epsilon}$, to a distribution $\pi_{\text{old}}(\boldsymbol{\epsilon})$ over KL-constraints. Thus (11) and (12) can be derived by following steps analogous to those given in Abdolmaleki et al. [6].

To approximate the expectations over the preference distribution and state distribution, we draw L states from the replay buffer and L preferences $\boldsymbol{\epsilon}$ from the preference policy $\pi_{\text{old}}(\epsilon)$. To approximate the integrals over a, for each $(s,\boldsymbol{\epsilon})$ pair we sample M actions from the current policy $\pi_{\text{old}}(a|s,\boldsymbol{\epsilon})$.

Learning a new parameterized action policy: After obtaining per-objective improved policies, we use supervised learning to distill these distributions into a new parameterized policy:

$$\begin{aligned} \max_\theta \quad & \sum_{k=0}^{K} \mathbb{E}_{p_{\text{old}}(\boldsymbol{\epsilon})\mu(s)}\big[\mathrm{KL}(q_k(a|s,\boldsymbol{\epsilon}) \| \pi_\theta(a|s,\boldsymbol{\epsilon}))\big] \\ \text{s.t.} \quad & \mathbb{E}_{p_{\text{old}}(\boldsymbol{\epsilon})\mu(s)}\big[\mathrm{KL}(\pi_{\text{old}}(a|s,\boldsymbol{\epsilon}) \| \pi_\theta(a|s,\boldsymbol{\epsilon}))\big] < \beta, \end{aligned} \tag{13}$$

subject to a trust region with bound $\beta > 0$ for more stable learning. To solve this optimization, we use Lagrangian relaxation as described in Abdolmaleki et al. [30, 6].

E.2 Learning Preference Distributions

As discussed in the main paper, one option for Module 2 of our framework is to use a preference distribution. Here we give more detail on how to learn a new preference distribution $p(\boldsymbol{\epsilon})$, given the current preference distribution $p_{\text{old}}(\boldsymbol{\epsilon})$ and fitness functions $f_k(\boldsymbol{\epsilon})$ (that evaluate $\boldsymbol{\epsilon}$ in terms of constraint satisfaction). One could add the fitness functions together to obtain $\sum_k f_k(\boldsymbol{\epsilon})$, and use any off-the-shelf RL algorithm to optimize for a new preference policy. On the other hand, this problem can be been as a multi-objective problem where the fitness for each constraint k is an objective. To this end, we use MO-MPO for learning $p(\boldsymbol{\epsilon})$. Note that in this paper we have at most two constraints. In the case of a problem with one constraint (i.e., one "objective"), MO-MPO [6] reduces to MPO [30].

More formally, following the MO-MPO algorithm, we optimize the following constrained optimization problem, that can be solved via Lagrangian relaxation [6]:

$$\begin{aligned} \max_\psi \quad & \sum_{k=1}^{K} \mathrm{KL}(z_k(\boldsymbol{\epsilon}) \,\|\, p_\psi(\boldsymbol{\epsilon})) \\ \text{s.t.} \quad & \mathrm{KL}(p_{\text{old}}(\boldsymbol{\epsilon}) \,\|\, p_\psi(\boldsymbol{\epsilon})) < \delta\,, \end{aligned} \tag{14}$$

where δ defines a trust region for more stable learning and $z_k(\boldsymbol{\epsilon})$ is a non-parametric improved policy for each objective, i.e,

$$z_k(\boldsymbol{\epsilon}) \propto p_{\text{old}}(\boldsymbol{\epsilon}) \exp\left(\frac{f_k(\boldsymbol{\epsilon})}{\varphi_k}\right).$$

φ_k is a temperature variable that we maintain for each objective (or fitness function) and is obtained by optimizing the convex dual function

$$g(\varphi_k) = \varphi_k \alpha_k + \varphi_k \log \int_{\boldsymbol{\epsilon}} p_{\text{old}}(\boldsymbol{\epsilon}) \exp\left(\frac{f_k(\boldsymbol{\epsilon})}{\varphi_k}\right) \mathrm{d}\boldsymbol{\epsilon}\,, \tag{15}$$

where α_k for each objective k defines a desired KL-divergence bound between the new improved policy $q_k(\boldsymbol{\epsilon})$ and current policy $p_{\text{old}}(\boldsymbol{\epsilon})$. We use the same value of $\alpha_k = 0.1$ for all fitness functions k throughout the paper. For more details on MO-MPO, please refer to [6].

E.3 General Algorithm

We maintain one online network and one target network for each Q-function, policy, and preference distribution. We also maintain one online network for the temperature function. Target networks are updated every fixed number of steps by copying parameters from the online network. Online networks are updated using gradient descent in each learning iteration. We refer to the target networks by using the subscript/superscript "old" throughout the paper.

We use an asynchronous actor-learner setup. In this setup actors fetch policy parameters from the learner and act in the environment while writing transitions to the replay buffer. At the beginning of each episode, we first sample one preference parameter $\boldsymbol{\epsilon}$ from the preference policy, which remains fixed until the end of the episode. The learner uses the transitions in the replay buffer to update the Q-functions, policies and temperature functions. Algorithm 1 describes one step of policy improvement for LP3 [MO-MPO-D].

Algorithm 1 LP3 [MO-MPO-D]: One policy improvement step

given batch size (L), number of actions and $\boldsymbol{\epsilon}$ to sample (M), current policy $p_{\text{old}}(\boldsymbol{\epsilon})\pi_{\text{old}}(a|s,\boldsymbol{\epsilon})$, current $(K+1)$ Q-functions $\{Q_k^{\text{old}}(s,a,\boldsymbol{\epsilon})\}_{k=0}^{K}$, temperature network $\eta_\omega(\boldsymbol{\epsilon})$, (K) constraint fitness functions $\{f_k(\boldsymbol{\epsilon})\}_{k=1}^{K}$, (K) temperature variables $\{\varphi_k\}_{k=1}^{K}$, (K) KL bounds $\{\alpha_k\}_{k=1}^{K}$, replay buffer $\mathcal{D}$, first-order gradient-based optimizer $\mathcal{O}$

initialize $\pi_\theta(a|s,\boldsymbol{\epsilon})$ from the parameters of $\pi_{\text{old}}(a|s,\boldsymbol{\epsilon})$
initialize $p_\psi(\boldsymbol{\epsilon})$ from the parameters of $p_{\text{old}}(\boldsymbol{\epsilon})$
repeat
  // Collect dataset $\{s^i, \boldsymbol{\epsilon}^i, a^{ij}, Q_k^{ij}\}_{i,j,k}^{L,M,K}$**, where**
  // L states $s^i \sim \mathcal{D}$
  // L preferences $\boldsymbol{\epsilon}^i \sim p_{\text{old}}(\boldsymbol{\epsilon})$
  // M actions $a^{ij} \sim \pi_{\text{old}}(a|s^i,\boldsymbol{\epsilon}^i)$ **and** $Q_k^{ij} = Q_k^{\text{old}}(s^i,\boldsymbol{\epsilon}^i,a^{ij})$

  // Compute (non-parametric) action distribution for each objective
  $\delta_\omega \leftarrow \nabla_\omega \sum_k \frac{1}{L}\sum_i^L \eta_\omega(\boldsymbol{\epsilon}^i)[k]\left[\epsilon_k^i + \log \sum_j^M \frac{1}{M}\exp\left(\frac{Q_k^{ij}}{\eta_\omega(\boldsymbol{\epsilon}^i)[k]}\right)\right]$, where [k] is the index of the vector
  Update ω based on δ_ω, using optimizer $\mathcal{O}$
  for k = 0, ..., K **do**
    $q_k^{ij} \propto \exp(\frac{Q_k^{ij}}{\eta_\omega(\boldsymbol{\epsilon}^i)[k]})$
  end for

  // Update action policy
  $\delta_\theta \leftarrow -\nabla_\theta \sum_i^L \sum_j^M \sum_{k=0}^K q_k^{ij} \log \pi_\theta(a^{ij}|s^i,\boldsymbol{\epsilon}^i)$
    (subject to additional KL regularization)
  Update π_θ based on δ_π, using optimizer $\mathcal{O}$

  // Collect dataset $\{\boldsymbol{\epsilon}^i, f_k^i\}_{i,k}^{M,K}$**, where**
  // M preferences $\boldsymbol{\epsilon}^i \sim p_{\text{old}}(\boldsymbol{\epsilon})$ **and** $f_k^i = f_k(\boldsymbol{\epsilon}^i)$

  // Compute epsilon distribution for each constrained objective
  for k = 1, ..., K **do**
    $\delta_{\varphi_k} \leftarrow \nabla_{\varphi_k}\varphi_k\alpha_k + \varphi_k \log\left(\sum_i \frac{1}{M}\exp\left(\frac{f_k^i}{\varphi_k}\right)\right)$
    Update φ_k based on δ_{φ_k}, using optimizer $\mathcal{O}$
    $z_k^i \propto \exp(\frac{f_k^i}{\varphi_k})$
  end for

  // Update preference policy
  $\delta_\psi \leftarrow -\nabla_\psi \sum_i^M \sum_{k=1}^K z_k^i \log p_\psi(\boldsymbol{\epsilon}^i)$
    (subject to additional KL regularization)
  Update p_ψ based on δ_ψ, using optimizer $\mathcal{O}$

until fixed number of steps
return $\pi_{\text{old}}(a|s,\boldsymbol{\epsilon}) = \pi_\theta(a|s,\boldsymbol{\epsilon})$ and $p_{\text{old}}(\boldsymbol{\epsilon}) = p_\psi(\boldsymbol{\epsilon})$