LEARNING ON-POLICY LEARNING TO SPECIALIZE ROBOT POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce *Generative On-Policy Learners* (GOLs), generative models that quickly adapt policy parameters to an unknown target domain. Conditioned on an initial (potentially random) policy and a few interaction episodes in the target domain, GOLs learn to generate improved policy parameters, by mimicking the policy evolution in traditional on-policy reinforcement learning. Unlike prior approaches, GOLs do not require (1) knowledge about domain parameters and rewards, (2) a large number of target domain interactions, or (3) expensive online domain identification. Generation is performed in a behavior-aligned latent space. We evaluate GOLs on continuous control problems, where they outperform state-of-the-art baselines in few shot domain adaptation on the most dynamic tasks. In addition, we demonstrate that our weight generation approach can iteratively improve policies in a sim-to-real transfer task. Finally, we characterize the key properties of the training dataset required to effectively train the weight generator, offering insights for scaling generative policy synthesis.

1 INTRODUCTION

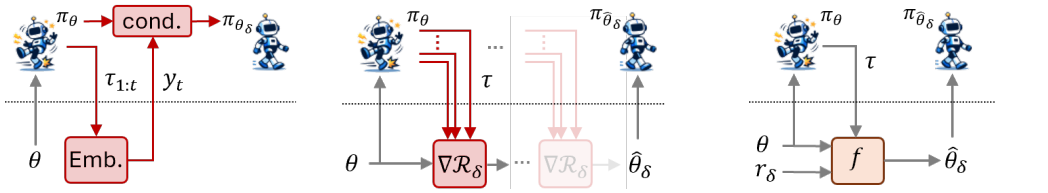
In weight generation, a hypernetwork is trained to output the parameters of a neural network (Schmidhuber, 1992; Bertinetto et al., 2016; Ha et al., 2016; Metz et al., 2022). Weight generation holds profound implications for few-shot domain adaptation (Rezaei-Shoshtari et al., 2023; Ren et al., 2025). With limited interactions from a target domain, a hypernetwork could quickly generate the parameters of a controller *specialized* to the robot's context, *e.g.*, its payload or hardware configuration. This approach represents an understudied alternative to popular methods for few-shot adaptation (Fig. 1), including direct gradient-based optimization in the target domain and conditional adaptation, where a network conditioned on the state-action history jointly estimates the domain and outputs commands.

However, adaptation via weight generation faces significant challenges: due to their compositional nature, multi-layer neural networks amplify small deviations in weights and inputs, making their behavior sensitive to small errors in generation. For example, a slight error in a parameter might cause a large deviation in the output if it flips the activation state of a ReLU, whereas the same error might be harmless if the unit remains inactive. Recent work has successfully addressed this challenge for *static* computer vision tasks, such as image classification (Schürholt et al., 2022; Navon et al., 2023; Wang et al., 2025; Schürholt et al., 2024; Zhmoginov et al., 2022).

Extending this success to *dynamic* tasks such as robot control has proven far more difficult: due to its sequential nature, errors in the generation of policy parameters propagate across time, and can shift the state-action distribution outside of the policy's training range. Prior works have made significant strides in this problem (Beck et al., 2023; Ren et al., 2025; Rezaei-Shoshtari et al., 2023; Liang et al., 2024). However, they often require auxiliary inputs during adaptation, such as expert demonstrations (Liang et al., 2024) or privileged information about the target domain (Beck et al., 2023; Rezaei-Shoshtari et al., 2023; Ren et al., 2025).

In this work, we introduce generative on-policy learners (GOL), a class of latent-space generative models that mimic *on-policy* RL algorithms: they generate parameters solely based on an initial policy's parameters and a single state-action trajectory in the target domain (Fig. 1). GOL learns from checkpoints trained with classical RL algorithms (Schulman et al., 2017; Haarnoja et al., 2018). Specifically, given a policy from an earlier stage of training and a single rollout from it, they predict a target policy at a more advanced training stage. During adaptation, GOL generates trained policies with just a few interactions in the target domain, without associated rewards.

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.



(a) Conditional adaptation (b) Gradient-based adaptation (c) Adaptation by Policy Generation

Figure 1: Overview of domain adaptation approaches. Domain adaptation seeks to find $\hat{\theta}_\delta$ of policy $\pi_{\hat{\theta}_\delta}$ in domain δ by adapting initial parameters θ of policy π_θ with as few interactions in the target domain as possible. (a) Conditional adaptation converts trajectory history $\tau_{1:t}$ into domain-predictive latents y_t (RNN or feed-forward-based) to modulate a meta-policy $\pi_\theta(a|s_t, y_t)$ mimicking the target policy π_{θ_δ}. (b) Gradient-based adaptation instead updates the network weights directly via estimated policy gradients $\nabla\hat{\mathcal{R}}_\delta$. However, computing gradients relies on expert demonstrations or reward signals and requires a large number of trajectories to be effective. (c) In this work, we adopt a policy generation approach for adaptation, in which weight updates are entirely generated by a neural network f. This model only requires a single trajectory, desired performance relative to an expert, and initial checkpoint, mitigating excessive data collection or measuring rewards in the target domain.

We empirically demonstrate that GOL adapts policies to out-of-distribution domains better than state-of-the-art methods in highly dynamic locomotion and flight tasks, outperforming prior work in policy generation, conditional adaptation, and gradient-based meta-learning. Additionally, to the best of our knowledge, this is the first work to establish that weight generation is viable and effective for sim-to-real few-shot adaptation. On a real-world drone racing task, GOL generates policies capable of fast and safe flight in as few as two adaptation steps (12s of real-world data), even when initialized from policies that can barely fly. Finally, we analyze the properties of the training checkpoints required to make GOL effective. This analysis reveals a correlation between the effective rank and gradient-projection magnitude of the policies in the training dataset and the quality of the resulting generative model.

Contributions. We present: (1) a novel design that frames few-shot adaptation as the prediction of network parameters, without any privileged knowledge or reward from the target domain; (2) an architecture and training paradigm that tackles the challenges of weight generation in continuous control; and (3) a novel insight into which properties of the training dataset correlate with effective weight generation for continuous control.

2 Related Work

We review prior work in few-shot adaptation for continuous control, which falls into three categories: (*i*) Conditional Adaptation, (*ii*) Gradient-based Adaptation, (*iii*) Adaptation by Policy Generation.

Conditional Adaptation has its roots in adaptive control (Mareels et al., 1987) and follows the core idea of conditioning control policies on state-action history-based latents that are predictive of the *target domain*. Different methods are distinguished by how they derive these latents: One set of methods assumes access to explicit domain parameters from a simulator, which are used to ground history embeddings via supervised learning (Lee et al., 2020; Kumar et al., 2021; Qi et al., 2023; Zhang et al., 2025). Other methods instead relax this assumption: they learn to generate embeddings purely from offline data alone while discouraging the encoder from generating behavior-policy-specific features (Gao et al., 2023), perform online probabilistic filtering on domain variables (Rakelly et al., 2019), leveraging pseudo-transitions to reduce sensitivity to different behavior policies (Zhou et al., 2024), or estimating context uncertainty to collect informative trajectories (Zhou et al., 2025a; Zintgraf et al., 2021). A third set of methods, instead, does not inject any inductive bias into the architecture or loss function. Instead they rely exclusively on domain randomization (Tobin et al., 2017) and long-context history encoders (Liu et al., 2025; Duan et al., 2016; Andrychowicz et al., 2020) to jointly estimate the domain and predict commands. However, having a sufficiently long context comes with large test-time computation cost and is therefore incompatible with low-latency control on compute-constrained platforms.

Gradient-Based Adaptation updates policy weights directly using gradients computed from interactions in the target domain. Because these interactions are usually expensive to collect, these

methods mix large amounts of source domain data with limited amounts of target domain data and run tailored learning algorithms. (Eysenbach et al., 2020) estimates the dynamics mismatch from transitions and then modifies source-domain rewards to discourage behaviors that are unlikely to transfer. Other methods, instead, reduce the influence of misleading source domain data by penalizing value estimation for transitions with large dynamics gap (Niu et al., 2022), or filtering them using target domain value estimation (Xu et al., 2023). Another approach learns to predict the next state from target-domain transitions and penalizes source-domain transitions whose next states deviate from the target-domain prediction (Lyu et al., 2024). More recent methods model offline and online dynamics with coupled flows, using their Wasserstein distance to estimate the dynamics gap and directly exploring online toward regions where that gap is large (Kong et al., 2026). These methods adapt to new domains with limited interactions, but require a large number of expensive gradient computations and policy updates.

Adaptation by Policy Generation is most closely related to our work, and works by predicting the weights of the adapted policy directly using a weight generator (Beck et al., 2023; Rezaei-Shoshtari et al., 2023; Liang et al., 2024; Zhou et al., 2025b; Ren et al., 2025; Hegde et al., 2024; Hegde & Sukhatme, 2023). Prior works, however usually assume access to some privileged knowledge: For instance, one set of works conditions weight generation on the trajectories (Liang et al., 2024) or intermediate goals (Zhou et al., 2025b) of an optimal policy, which forces the weight generator to learn to optimize an imitation learning loss. However, when adapting to a novel domain we cannot rely on expert demonstrations. Relying on a similar assumption, other work generates control policies from the ground-truth transition parameters domain (Beck et al., 2023; Ren et al., 2025; Rezaei-Shoshtari et al., 2023) which are similarly challenging to access in practice and often unavailable for the target domain. Our work is also related to (Peebles et al., 2022), which, similarly to ours, can recursively predict policy parameters but was exclusively demonstrated on a simulated cartpole task and requires rewards from the target domain. Our approach overcomes these limitations and does not require any privileged knowledge or reward from the target domain.

3 BACKGROUND

Markov Decision Process. Let $\mathcal{M}_\delta = (\mathcal{S}, \mathcal{A}, \mathcal{T}_\delta, \mathcal{R}, \gamma, \iota)$ be a Markov Decision Process (MDP) in domain δ, with states $s_t \in \mathcal{S}$, actions $a_t \in \mathcal{A}$, domain-dependent transition probabilities $\mathcal{T}_\delta : \mathcal{S} \times \mathcal{A} \to \text{Dist}(\mathcal{S})$, rewards $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, discount factor $\gamma \in (0, 1]$, and initial state distribution $\iota \in \text{Dist}(\mathcal{S})$. Our goal is to find policy $\pi : \mathcal{S} \to \text{Dist}(\mathcal{A})$ with maximal expected cumulative reward

$$\pi^*_\delta = \arg\max_\pi \mathcal{R}_\delta(\pi) \quad \text{with} \quad \mathcal{R}_\delta(\pi) := \mathbb{E}_{\tau \sim p_{\pi,\delta}} \left[\sum_{t=1}^{T} \mathcal{R}(s_t, a_t) \gamma^t_\delta \right]. \tag{1}$$

State-action trajectories $\tau = \{(s_t, a_t)\}_{t=1}^T$ of length T are sampled from policy induced distribution $p_{\pi,\delta}(\tau)$. As is standard in continuous control problems, we will optimize equation 1 over the family of parametric policies $\pi_\theta(a|s) := \mathcal{N}(a|\mu_\theta(s), \sigma^2 I)$, with fixed covariance $I\sigma^2$, and learnable mean $\mu_\theta(s)$ parametrized by multi-layer perceptron (MLP) weights θ. Thus, we will abbreviate $\mathcal{R}_\delta(\theta) := \mathcal{R}_\delta(\pi_\theta)$ and $p_{\theta,\delta} := p_{\pi_\theta,\delta}$, and instead focus on finding $\theta^*_\delta = \arg\max_\theta \mathcal{R}_\delta(\theta)$.

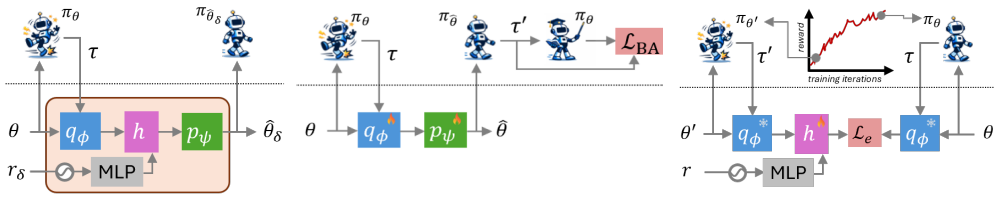
Domain Adaptation. In real world domains $\delta \sim p_\Delta(\delta)$ *(i)* trajectory collection is expensive, *(ii)* rewards and dynamics are usually unknown, and if known *(iii)* rewards yield unstable gradients. Thus we leverage data collected in artificial *source domains* $\delta' \sim p_{\Delta'}(\delta')$, with known dynamics $\mathcal{T}_{\delta'}$ and rewards $\mathcal{R}$, but $p_{\Delta'} \neq p_\Delta$. Maximizing rewards over source domains yields generalist policy (Tobin et al., 2017):

$$\theta^* = \arg\max_\theta \mathbb{E}_{\delta' \sim p_\Delta}[\mathcal{R}_{\delta'}(\theta)]. \tag{2}$$

In the target domain $\delta \sim p_\Delta$, however, this policy, exhibits a performance gap $\mathcal{R}_\delta(\theta^*) \leq \mathcal{R}_\delta(\theta^*_\delta)$ compared to specialist policy θ^*_δ. Domain adaptation (see Fig. 1) seek to close this gap in one of two ways: Gradient-based adaptation techniques perform iterative refinement of weights θ^* (Fig. 1 (b)):

$$\theta^*_\delta \approx \theta^* + \eta \sum_{i=1}^{N} \nabla \hat{\mathcal{R}}_\delta(\theta_{i,\delta}) \quad \text{with} \quad \theta_{i+1,\delta} = \theta_{i,\delta} + \eta \nabla \hat{\mathcal{R}}_\delta(\theta_{i,\delta}) \quad \text{and} \quad \theta_{1,\delta} = \theta^* \tag{3}$$

Here, $\nabla \hat{\mathcal{R}}_\delta(\theta_{i,\delta})$ is an estimate of the policy gradient, η is a learning rate and N counts the number of iterations. Estimating policy gradients in the target domain, however, requires a reward function



(a) Method overview (b) AE pre-training (c) Evolution module training

Figure 2: Method overview and training recipe. (a) Our method learns to generate the parameters $\hat{\theta}_\delta$ of an MLP-based policy $\pi_{\hat{\theta}_\delta}$ with desired return r_δ by leveraging the initial parameters θ of a policy π_θ, and its state-action trajectory τ in the target domain. It features a trajectory-conditioned auto-encoder (AE, q_ϕ, p_ψ) and a reward-conditioned latent propagator h, each trained sequentially. (b) Behavior-aligned AE pre-training minimizes a DAgger-inspired behavior cloning loss $\mathcal{L}_{\text{BA}}$. (c) Evolution module training minimizes the mismatch $\mathcal{L}_{\text{Prop.}}$ between propagated and target latents, corresponding with checkpoints θ' and θ. These checkpoints are sampled along a training run within a single domain.

(hard to estimate in the real world) and, more importantly, many trajectories of the current policy to reduce variance.

Conditional adaptation (Fig. 1 (a)) instead augments policies with learned latents y_t from state-action histories, which are predictive of the underlying domain variable δ, *i.e.* $\pi_\theta(a_t|s_t, y_t) \approx \pi_\theta(a_t|s_t, \delta)$ mimicking specialist policy $\pi_{\theta^*_\delta}(a_t|s_t)$. By minimizing equation 2, they learn to perform online system identification, either explicitly (Lee et al., 2020; Kumar et al., 2021; Zhang et al., 2025) or implicitly (Duan et al., 2016; Liu et al., 2025; Zhou et al., 2024; 2025a; Eschmann et al., 2026). However estimating y_t at every step is challenging when the context lacks sufficient information about the domain. This is often the case when, due to computational constraints, the context window is limited to the last few seconds of transitions (Eschmann et al., 2026; Zhou et al., 2024; Kumar et al., 2021; Zhang et al., 2025). Moreover, estimating y_t online induces additional computation. Next, we show how to combine the advantages of both methods for few-shot domain adaptation (Fig. 1 (c)).

3.1 Generative On-Policy Learning

Inspired by gradient-based adaptation, we train a generative model f to perform the iterative gradient ascent in equation 3. In detail, given a target domain δ and an initial checkpoint θ we generate new checkpoints θ_δ with

$$\theta_\delta = f(\theta, \tau, r_\delta) \quad \text{where} \quad \tau \sim p_{\theta,\delta}(\tau). \tag{4}$$

Generation is conditioned on a single trajectory $\tau \sim p_{\theta,\delta}(\tau)$ from the current policy θ, and a scalar target $r_\delta \approx \mathcal{R}_\delta(\theta_\delta)/\mathcal{R}_\delta(\theta^*_\delta) \in [0, 1]$ that quantifies the desired performance of the generated policy θ_δ relative to a domain expert θ^*_δ. We thus term f a *generative on-policy learner* (GOL). Note that f is agnostic to the origin of θ which has two implications: Firstly, f can be applied autoregressively using checkpoints from previous iterations. Secondly, as shown later, f can be trained to use checkpoints θ *in the same domain* as θ_δ, but at different training iterations.

By leveraging θ and τ in the target domain δ, equation 4 uses the key ingredients for computing policy gradients but does not do so explicitly. Instead, the desired improvement is specified through the target performance indicator r_δ. This design eliminates the need to *(i)* compute rewards or model accurate dynamics in the target domain, and *(ii)* condition on ground-truth domain parameters, as in prior weight generation work for domain adaptation (Beck et al., 2023; Rezaei-Shoshtari et al., 2023; Ren et al., 2025). Similar to online adaptation, f infers δ from τ to improve θ, but does so on a sufficiently long context without sacrificing the policy network's capacity or efficiency.

As illustrated in Fig. 2, f combines an auto-encoder $q_\phi, p_\psi$with evolution module h mapping policy embeddings z to the desired performance in the latent space.

$$\theta_\delta = p_\psi(h(z, r_\delta)) \quad \text{with} \quad z = q_\phi(\theta, \tau) \tag{5}$$

We represent $\theta := [\theta_1, \theta_2, ..., \theta_{2L}]$ where group θ_i corresponds with a flattened weight matrix or bias and $L = 3$ is the number of layers. This latent-generation process, inspired by image generation systems (Rombach et al., 2022), improves training efficiency at very small performance costs in comparison to end-to-end generation, as done in prior work (see Appendix F).

Trajectory-Conditioned AE. Compared to prior policy generation work (Peebles et al., 2022; Ren et al., 2025; Rezaei-Shoshtari et al., 2023; Liang et al., 2024), we predict policies in latent space instead of parameter space. We model each parameter group $\theta_i = p_\psi(z_i)$ as arising from deterministic decoder $p_\psi(z_i) := V_i z_i$, where $V_i \in \mathbb{R}^{d_i \times C}$ denotes the learnable projection, number of parameters d_i in group i, and $C = 256$ dimensional latents $z_i = q_\phi(\theta_i, \tau)$.

Our latent $q_\phi(\theta_i, \tau)$ captures parameter group interdependence via trajectory-conditioning. It's modeled by

$$q_\phi(\theta_i, \tau) := W_q z_i' \quad \text{where} \quad z_i' = \alpha(\tau) \odot (W_i \theta_i) + \beta(\tau). \tag{6}$$

Here $W_q \in \mathbb{R}^{C \times C}$ and $W_i \in \mathbb{R}^{C \times d_i}$ are learnable components and z_i' are modulated features. Affine FiLM (Perez et al., 2018) parameters $\alpha(\tau), \beta(\tau) \in \mathbb{R}^C$ are derived from trajectories $\tau \in \mathbb{R}^{T \times (d_\mathcal{A} + d_\mathcal{S})}$ comprising concatenated actions and states. We map τ through a linear projection, layer normalization, temporal average pooling, followed by a 2-layer MLP with ReLU.

Latent Evolution. We implement a latent evolution module mapping latents $z := \{z_i\}_{i=1}^{2L} \in \mathbb{R}^{2L \times C}$ to $h(z, r_\delta)$ corresponding with the desired performance $r_\delta \in [0, 1]$ in domain δ. We treat z, together with a frequency embedding $\text{emb}(r_\delta) \in \mathbb{R}^{2F+1}$ (with $F = 128$) as inputs to a transformer:

$$z_\delta = z + \text{Transformer}(z, \text{emb}(r_\delta)). \tag{7}$$

The model performs multi-head attention (12 heads) across 12 blocks with dimension of $d = 1536$.

3.2 Training Recipe

Our method is trained on checkpoints $\mathcal{D} = \mathcal{D}_{\delta_1} \cup ... \cup \mathcal{D}_{\delta_D}$ comprising training runs of off-the-shelf RL algorithms in D domains $\delta_j \sim p_\Delta(\delta)$. Each run $D_{\delta_j} := \{(\theta_{\delta_j}^{(i)}, r_{\delta_j}^{(i)}, \delta_j)\}_{i=1}^{M}$ comprises $M = 200$ checkpoints $\theta_{\delta_j}^i$ at training iteration i, and expected performance estimates $r_{\delta_j}^{(i)} = \mathcal{R}_{\delta_j}^{(i)}(\theta_{\delta_j}) / \max_i \mathcal{R}_{\delta_j}^{(i)}(\theta_{\delta_j})$. We train in two stages, with details discussed in Appendix D.

Behavior-aligned AE Training. We aim primarily at constructing latents z which are predictive of a checkpoint's expected behavior. We introduce an on-policy behavioral cloning loss

$$\mathcal{L}_{\text{BA}} = \mathbb{E}_{\tau \sim p_{\hat{\theta}}} \left[\sum_{t=1}^{T} \mathbb{E}_{a_t' \sim \pi_\theta} \left[\|a_t' - a_t\|^2 \right] \right] = \mathbb{E}_{\tau \sim p_{\hat{\theta}}} \left[\sum_{t=1}^{T} \|\mu_\theta(s_t) - \mu_{\hat{\theta}}(s_t)\|^2 \right] + \text{c.} \tag{8}$$

The equation 8 aims to align actions a_t by predicted policy $\pi_{\hat{\theta}}$ with actions a_t', by ground-truth policy π_θ, on states s_t visited by the predicted policy.

Averaging over trajectories observed by the predicted policy has been shown to yield provably lower regret compared to sampling over trajectories observed by the ground-truth policy (Ross et al., 2011). This DAgger-style procedure benefits our method in a similar way, and departs from the approach common in prior work of aligning behaviors on the states observed by the ground truth policy (Rezaei-Shoshtari et al., 2023; Ren et al., 2025; Zhou et al., 2025b). To offset the sampling cost during training, we only sample a single trajectory per generated checkpoint.

Latent Evolution Training. Next, we freeze the encoder, and train h. At each training iteration, we sample checkpoint pairs θ', θ with the *same* domain δ, and θ appearing later during training than θ'. We then train h to move latent $q_\phi(\theta', \tau')$ to latent $q_\phi(\theta, \tau)$:

$$\mathcal{L}_e = \mathbb{E}_{\delta \sim p_\Delta, (\theta', \tau'), (\theta, \tau) \sim \mathcal{D}_\delta, \tau \sim p_{\theta, \delta}, \tau' \sim p_{\theta', \delta}} \left[\rho \left(h(q_\phi(\theta', \tau'), r) - q_\phi(\theta, \tau) \right) \right] \tag{9}$$

Here ρ denotes the Huber norm (with threshold 2). Again, we sample single trajectories τ, τ'.

4 Experiments

Our experimental design seeks to answer the following questions: (1) *How effective is GOL on few-shot domain adaptation?* (2) *Can our method be used for simulation-to-real-world transfer?* (3) *What is the impact of each of our design decisions?*

Setup. We evaluate our method on three challenging *dynamic* tasks (see Fig. 3): the *ant* environment from the Brax GDM benchmark (Freeman et al., 2021), the *legged robot* environment from MuJoCo Playground (Zakka et al., 2025), and a *drone racing* environment where we show results in both a custom-built simulator in MJX based on (Pasumarti et al., 2025), and the real world. For each task, we collect $D = 500$ training runs with $M = 200$ checkpoints each. For the ant and legged robot, a domain is characterized by a front and back leg rescaling factor $\delta = \{r_f, r_b\}$ with ranges $r_f, r_b \sim \mathcal{U}[0.5, 1.0]$. For drone racing a domain is characterized by thrust reductions $\delta = \{r_{p_i}\}_{i=1}^{4}$ where $r_{p_i} \sim \mathcal{U}[0, 0.2]$ describes a force reduction of propeller i by a factor of $1 - r_{p_i}$. We use SAC (Haarnoja et al., 2018) training algorithm for all tasks. We represent the policy as an MLP with hidden layers $[32, 32]$ for the ant task, and $[64, 64]$ for the remaining tasks.

Evaluation Procedure. We validate the effectiveness of GOL in adapting policies over a small number of iterations ($N = 10$) of equation 4, setting $r_\delta = 1$, and collecting a single trajectory each time. We report the mean and variance of episode rewards for adapted policies to a set of 256 fixed target domains δ. We average over 128 rollouts, and 16 initial checkpoints $\theta_{\delta'}$ pretrained in domains δ', which are sampled from held-out test runs. For ant and legged robot, a trajectory for adaptation consists of $T_a = 200$ environment steps, while for drone racing $T_a = 300$. We compare results for target domains sampled in distribution (ID, mimicking the source domain, which can overlap with training domains) and out of distribution (OOD, $r_f, r_b \sim \mathcal{U}[0.25, 1.75]$ for the ant and legged robot, $r_{p_i} \sim \mathcal{U}[0.2, 0.4]$ for drone racing).

Baselines. We evaluate against representative baselines from each adaptation category (Fig. 1); implementation details and data requirements are provided in Appendix C. Note that our amortized-inference approach requires significantly more source-domain training samples than the baselines.

Conditional adaptation. `GENTLE` (Zhou et al., 2024) conditions policies on trajectory-derived embeddings that are predictive of successor states and rewards. `CERTAIN+UNICORN` (Zhou et al., 2025a) augments the Unicorn (Li et al., 2024) base policy with an uncertainty-based adaptation module that identifies ambiguous context in the trajectories. The latter two methods require rewards from the target domains, which ours and the following baselines do not. `RMA` (Kumar et al., 2021) trains the adaptation module in a supervised way by predicting a latent representation of the ground-truth domain parameters from the state-action history. `RAPTOR*` is our simplified implementation of RAPTOR (Eschmann et al., 2026) which distills domain-specific controllers into a single policy conditioned on the state-action history. `DR-H` also fuses domain estimation and control into a single history-conditioned policy, represented by a GRU (Cho et al., 2014). Akin to (Liu et al., 2025), `DR-H` optimizes the policy directly with reinforcement learning and domain randomization.

Adaptation by Policy Generation. From this category, we select `HyPoGen` (Ren et al., 2025) as a representative since it significantly outperforms prior methods (Rezaei-Shoshtari et al., 2023; Beck et al., 2023) and does not rely on expert trajectories from the target distribution (Liang et al., 2024). Following their implementation, we use it for domain adaptation by conditioning it on the ground-truth domain parameters. Therefore, it has privileged information that ours and other baselines do not. We are unaware of methods from this category operating on assumptions similar to ours.

Gradient-Based Adaptation. Works from this category require an order of magnitude more target-domain data than our model and the baselines, making them unsuitable for the main experiments. Nonetheless, we compare to `CompFlow` (Kong et al., 2026), the state-of-the-art from this category, in Appendix E. The results show significant performance degradation in the low-data regime.

4.1 Comparison to Prior Adaptation Methods

Figure 3 summarizes the results. For in-distribution (ID) settings, most methods achieve comparable performance. A notable exception is `UNICORN+CERTAIN`. `GENTLE` performs well for Ant, but still struggles in the more dynamic tasks. Although the latter two are designed for one-step adaptation, we evaluated them recursively for 10 steps to expose them to the same amount of data as ours, but their performance remained unchanged or slightly degraded. Out-of-distribution (OOD) we observe a larger gap between approaches. On the simplest task (Ant), `DR-H` outperforms all baselines, though our approach maintains the narrowest performance gap. In the more dynamic robot locomotion and drone flight tasks, however, GOL outperforms all baselines OOD, with the largest gains (24% w.r.t. `DR-H`) observed in racing. Finally, while the policy generation baseline (`HyPoGen`) performs well ID, it struggles OOD, despite receiving ground-truth domain parameters.

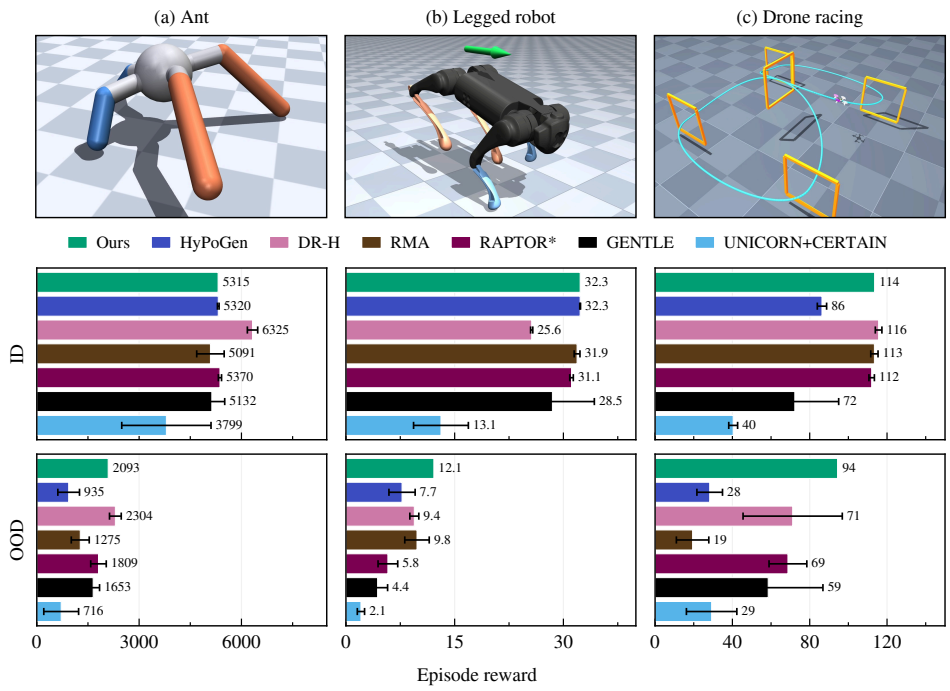


Figure 3: Average episode reward of policies adapted to unseen target domains for (a) an Ant locomotion with randomized leg lengths, (b) a quadruped locomotion with randomized leg lengths, and (c) drone racing with random propeller thrust reductions. For GOL, we use 1 training seed and the results are averaged over 16 initial policies sampled from held-out test runs. We evaluated across 256 target domains, sampled either in-distribution (ID) or out-of-distribution (OOD; see Sec. 4). The results are averaged over 4 different training seeds for `GENTLE`, `UNICORN+CERTAIN`, and `HyPoGen`; and 16 for the other baselines. More details are in Appendix C. While most baselines perform similarly ID, there are larger gaps OOD. Our approach is second-best on the simple Ant task, but outperforms all baselines on the more dynamic quadruped and drone tasks.

4.2 Application to Simulation-to-Real-World Transfer

We use GOL from the previous section to adapt three simulation-trained specialists, π^*_δ (with δ sampled in-distribution), to a real-world Crazyflie 2.1 Brushless drone. We don't perform any identification to fit the simulator to this platform. Experiments are conducted on a real-world lemniscate track matching the simulation geometry, using a motion-capture system for state estimation, and offboard computation for policy inference. We perform three sequential adaptation steps per policy and record lap times and failure rates after each step (Fig. 4, left), since we can't measure rewards in the real world. While initial policies fail to complete the track, our method enables all policies to fly reliably after just two adaptation steps (each step uses the first 6s, or the time from starting to crash, of real-world data as τ in the evolution module). The adapted policies achieve an average lap slightly better than `RAPTOR*` and much faster than `DR-H`, which exhibits high variance between episodes. Furthermore, trajectory plots show that successive adaptation steps yield progressively smoother flight paths (Fig. 4, right).

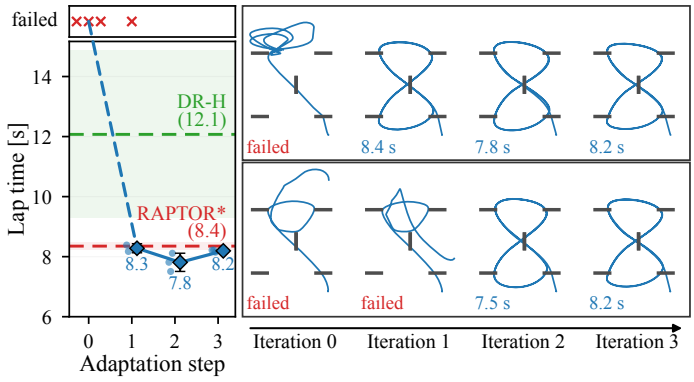
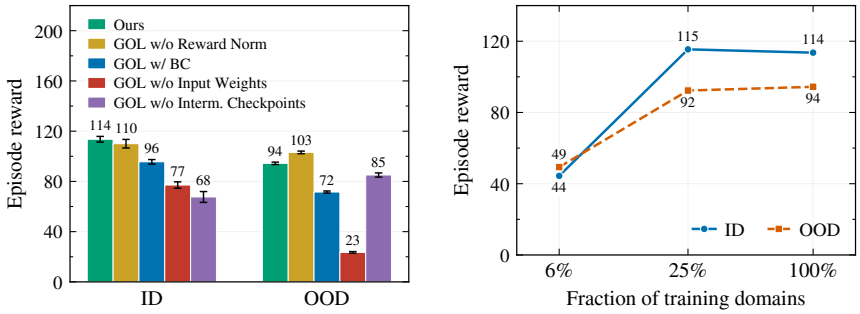


Figure 4: Sim-to-real adaptation: GOL adapts randomly-sampled specialist policies trained in simulation $\pi^*_{\delta_{1,2,3}}$) to the real world. We compare against `RAPTOR*` and `DR-H`, the strongest baselines in OOD tests in simulation. Both `RAPTOR*` and `DR-H` use 2 seeds and finish 3 runs per seed and 3 laps per run without failure. Left shows lap-times, while right shows flight profiles.



(a) Method ablation (b) % of training data

Figure 5: (a) Ablations on the drone racing task. We observe a large in-distribution performance drop but a surprisingly smaller out-of-distribution performance drop when training the evolution module without using policies from intermediate stages of training as target (see equation 9). Another large drop both in and out of distribution derives from removing the parameters θ from the input of GOL . Given the non-stationary nature of policy generation, using our DAgger-style loss (equation 8) for AE-training outperforms using a behavioral-cloning (BC) objective. Finally, normalizing rewards slightly affects OOD performance, but enables adaptation to target domains without knowledge of the episode reward attained by a specialist. (b) Our model is very data efficient, and achieves near-peak performance with only 25% of the training domains.

4.3 Ablation and Sensitivity Studies

Impact of core design choices. We evaluate four ablations on the drone racing task. First, `GOL w/ BC Loss` samples states in equation 8 from $p_{\theta,\delta}$ instead of $p_{\hat{\theta},\delta}$, following a behavioral-cloning paradigm. Second, `GOL w/o Reward Norm` replaces normalized rewards r_δ with raw episode rewards, a variant requiring privileged access to a specialist's target domain rewards. Third, `GOL w/o Interm. Checkpoints` samples the target checkpoint in equation 9 from the top 10% most trained checkpoints, drawing the initial checkpoint from the remaining 90%. Finally, `GOL w/o Input Weights` removes the current policy parameters θ from the input in equation 4, forcing the model to infer target parameters solely from the state-action trajectory generated by θ.

The results of these ablations, shown in Fig. 5(a), indicate that all design choices are important, but some more than others. The most critical is including partially-trained checkpoints as targets for training the evolution module. As also observed by prior work (Peebles et al., 2022), this provides GOL with richer learning signal than only the strongest policies. Similarly important are the input weights θ, which indicates that a trajectory alone, for dynamic environments, might be insufficient to identify the domain and predict a better policy. The drop in performance observed when using a BC loss instead of our proposed DAgger-style loss to train the AE highlights the importance of accounting for the non-stationary nature of policy learning in weight generation. Finally, normalizing the rewards leads to no noticeable performance change, enabling GOL to adapt policies without knowledge of target-domain specialist rewards.

Training Data Efficiency: Fig. 5(b) illustrates that GOL reaches almost peak performance in both ID and OOD settings with only 25% of the training data, obtained by subsampling the number of training domains. This highlights the data efficiency of our approach.

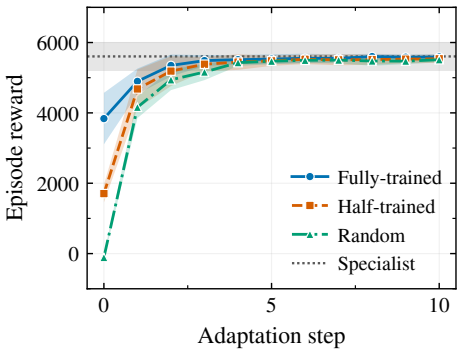


Figure 6: Policies in the ant task adapted with GOL to random target domains different from their training ones. All policies reach near-expert rewards in the first steps.

Sensitivity to initialization: Next, we study how sensitive GOL-based adaptation is to the initial policy proficiency (Fig. 6). We report episode rewards for adapted policies starting from random, halfway trained, and fully converged initial parameters from 16 randomly-sampled source domains and evaluated in 16 different randomly-sampled target domains. Regardless of initial policy quality, GOL consistently achieves near-specialist performance within a

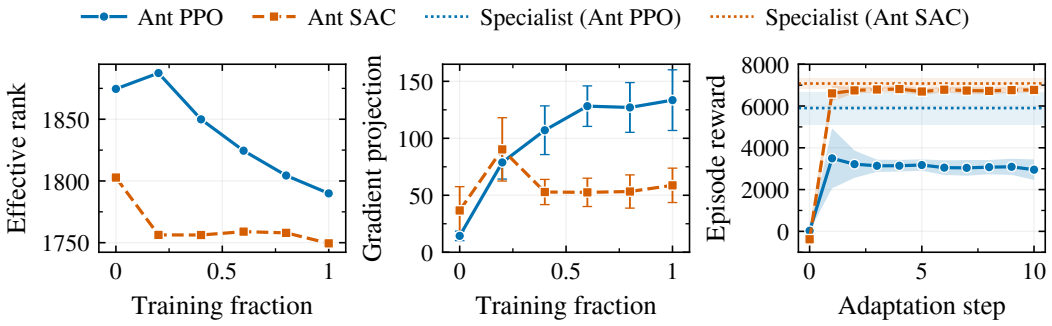


Figure 7: We find that the performance of our method improves if trained on checkpoints generated by SAC instead of PPO. A weight-space analysis shows that SAC has a smoother optimization landscape, as shown by a lower effective rank (Roy & Vetterli, 2007) and gradient magnitude (Ahmed et al., 2019) over training. A smoother landscape is easier to fit for our method since small errors do not largely alter performance.

few adaptation steps. The performance improvement in the first adaptation step is remarkable given that GOL was trained on single-domain RL optimizations and not on cross-domain transfer.

Impact of RL algorithm for training data collection: We investigate the impact of the RL algorithm used for data generation. To this end, we train GOL using datasets generated via either PPO (Schulman et al., 2017) or SAC (Haarnoja et al., 2018). To control for weight-space variations induced by different domains, we train and evaluate GOL on the default Ant environment. As illustrated in Fig. 7, we observe a large difference in our method's few-shot adaptation ability when changing the training dataset. This divergence cannot be explained away by the slightly inferior performance of specialists trained with PPO and is striking given that the weight distributions of the two datasets are similar (see Appendix for comparison).

To explain this gap, we perform spectral analysis of the two datasets. First, we compute the effective rank (Roy & Vetterli, 2007) of 3000 policies sampled across various training stages. A lower effective rank indicates that policies reside in a smoother region of the optimization landscape (Roy & Vetterli, 2007). As shown in Fig. 7, SAC policies have lower effective ranks than PPO policies throughout training. Second, we average the gradient projection magnitude g (Ahmed et al., 2019) over 32 training runs, defined as $g := \frac{1}{N_g} \sum_{i=1}^{N_g} |\nabla \mathcal{R} \cdot u_i|$ along $N_g = 128$ random directions with unit vector u_i. Post 20% training progress, PPO exhibits larger average gradient magnitudes than SAC. As observed in prior work (Ahmed et al., 2019), the difference in gradient magnitude could be explained by SAC's maximum-entropy objective. We hypothesize that this regularization, potentially smoothing out the optimization landscapes (Chaudhari et al., 2019), favors weight generation, as small prediction errors do not drastically degrade performance.

5 Discussion

State-of-the-art robotics controllers have shown impressive performance, but exhibit large performance drops in previously unseen conditions (Barreiros et al., 2026). We envision a future where the vast repository of policies trained daily by the research community serves as the foundation for weight generators that quickly specialize (part of) generalist policies to the task at hand. While this goal remains on the horizon, our framework represents a concrete step forward, offering insights into the data requirements, architectural constraints, and training objectives necessary for scaling future weight generators for few-shot adaptation.

Limitations. Our current framework optimizes policy performance along a single axis (episodic normalized reward), whereas practical deployments may require balancing competing objectives, such as safety or energy efficiency. In addition, similarly to prior work, our policy generator is task-specific. A promising future direction would study how to train generalist weight generators on policies designed for different tasks, embodiments and network architectures. This would unlock a vast reservoir of existing datasets for training which will enable scalable, general-purpose, and few-shot adaptation.

AI USE STATEMENT

In this work, we used generative AI tools for designing or providing feedback on research methodology or experiments, implementing methods, supporting qualitative and thematic data analysis, assisting with translation. We have not used generative AI tools for generating synthetic data sets, helping develop theoretical models or conceptual frameworks, proposing or refining hypotheses, cleaning and reformatting dataset, and interpreting results. Formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs are not applicable to this work. Additionally, we used generative AI tools for creating or modifying scientific figures or images, creating or edit software code, creation of artifacts, summarizing or analyzing existing literature, brainstorming, sourcing/searching for information, editing a research paper to improve readability. We have reviewed all AI-assisted work. LLM-generated code was carefully reviewed, verified and tested for correctness before reporting the results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *International conference on machine learning*, pp. 151–160. PMLR, 2019.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Jose Barreiros, Andrew Beaulieu, Aditya Bhat, Rick Cory, Eric Cousineau, Hongkai Dai, Ching-Hsin Fang, Kunimatsu Hashimoto, Muhammad Zubair Irshad, Masha Itkina, et al. A careful examination of large behavior models for multitask dexterous manipulation. *Science Robotics*, 11 (113):eaea6201, 2026.

Jacob Beck, Matthew Thomas Jackson, Risto Vuorio, and Shimon Whiteson. Hypernetworks in meta-reinforcement learning. In *Conference on Robot Learning*, pp. 1478–1487. PMLR, 2023.

Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. *Advances in neural information processing systems*, 29, 2016.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, eighth workshop on syntax, semantics and structure in statistical translation*, pp. 103–111, 2014.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl ^{2}: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

Jonas Eschmann, Dario Albani, and Giuseppe Loianno. Raptor: A foundation policy for quadrotor control. *Science Robotics*, 11(114):eaec1481, 2026.

Benjamin Eysenbach, Swapnil Asawa, Shreyas Chaudhari, Sergey Levine, and Ruslan Salakhutdinov. Off-dynamics reinforcement learning: Training for transfer with domain classifiers. *arXiv preprint arXiv:2006.13916*, 2020.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

Yunkai Gao, Rui Zhang, Jiaming Guo, Fan Wu, Qi Yi, Shaohui Peng, Siming Lan, Ruizhi Chen, Zidong Du, Xing Hu, et al. Context shift reduction for offline meta-reinforcement learning. *Advances in Neural Information Processing Systems*, 36:80024–80043, 2023.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Shashank Hegde and Gaurav S Sukhatme. Efficiently learning small policies for locomotion and manipulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5909–5915. IEEE, 2023.

Shashank Hegde, Zhehui Huang, and Gaurav S Sukhatme. Hyperppo: A scalable method for finding small policies for robotic control. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10821–10828. IEEE, 2024.

Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. A benchmark comparison of learned control policies for agile quadrotor flight. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 10504–10510, 2022. doi: 10.1109/ICRA46639.2022.9811564.

Donghu Kim, Youngdo Lee, Minho Park, Kinam Kim, I Nahendra, Takuma Seno, Sehee Min, Daniel Palenicek, Florian Vogt, Danica Kragic, et al. Flashsac: Fast and stable off-policy reinforcement learning for high-dimensional robot control. *arXiv preprint arXiv:2604.04539*, 2026.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lingkai Kong, Haichuan Wang, Tonghan Wang, Guojun Xiong, and Milind Tambe. Composite flow matching for reinforcement learning with shifted-dynamics data. *Advances in Neural Information Processing Systems*, 38:92500–92541, 2026.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *Robotics: Science and Systems XVII*, 2021.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

Lanqing Li, Hai Zhang, Xinyu Zhang, Shatong Zhu, Yang Yu, Junqiao Zhao, and Pheng-Ann Heng. Towards an information theoretic framework of context-based offline meta-reinforcement learning. *Advances in Neural Information Processing Systems*, 37:75642–75667, 2024.

Yongyuan Liang, Tingqiang Xu, Kaizhe Hu, Guangqi Jiang, Furong Huang, and Huazhe Xu. Make-an-agent: A generalizable policy network generator with behavior-prompted diffusion. *Advances in Neural Information Processing Systems*, 37:19288–19306, 2024.

Min Liu, Deepak Pathak, and Ananye Agarwal. Locoformer: Generalist locomotion via long-context adaptation. In *9th Annual Conference on Robot Learning*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`.

Fan-Ming Luo, Zuolin Tu, Zefang Huang, and Yang Yu. Efficient recurrent off-policy rl requires a context-encoder-specific learning rate. *Advances in Neural Information Processing Systems*, 37: 48484–48518, 2024.

Jiafei Lyu, Chenjia Bai, Jingwen Yang, Zongqing Lu, and Xiu Li. Cross-domain policy adaptation by capturing representation mismatch. *arXiv preprint arXiv:2405.15369*, 2024.

Iven M.Y. Mareels, Brian D.O. Anderson, Robert R. Bitmead, Marc Bodson, and Shankar S. Sastry. Revisiting the mit rule for adaptive control. *IFAC Proceedings Volumes*, 20(2):161–166, 1987. ISSN 1474-6670.

Luke Metz, James Harrison, C Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022.

Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In *International Conference on Machine Learning*, pp. 25790–25816. PMLR, 2023.

Haoyi Niu, Yiwen Qiu, Ming Li, Guyue Zhou, Jianming Hu, Xianyuan Zhan, et al. When to trust your simulator: Dynamics-aware hybrid offline-and-online reinforcement learning. *Advances in Neural Information Processing Systems*, 35:36599–36612, 2022.

Vineet Pasumarti, Lorenzo Bianchi, and Antonio Loquercio. Agile flight emerges from multi-agent competitive racing. *arXiv preprint arXiv:2512.11781*, 2025.

William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints. *arXiv preprint arXiv:2209.12892*, 2022.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: visual reasoning with a general conditioning layer. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.

Haozhi Qi, Ashish Kumar, Roberto Calandra, Yi Ma, and Jitendra Malik. In-hand object rotation via rapid motor adaptation. In *Conference on Robot Learning*, pp. 1722–1732. PMLR, 2023.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pp. 5331–5340. PMLR, 2019.

Hanxiang Ren, Li Sun, Xulong Wang, Pei Zhou, Zewen Wu, Siyan Dong, Difan Zou, Youyi Zheng, and Yanchao Yang. Hypogen: Optimization-biased hypernetworks for generalizable policy generation. In *International Conference on Learning Representations*, volume 2025, pp. 70586–70622, 2025.

Sahand Rezaei-Shoshtari, Charlotte Morissette, Francois R Hogan, Gregory Dudek, and David Meger. Hypernetworks for zero-shot transfer in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 9579–9587, 2023.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Olivier Roy and Martin Vetterli. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*, pp. 606–610. IEEE, 2007.

Gianluca Sabatini, Chenhao Li, and Marco Hutter. Bridging the gap: Enabling soft actor critic for high performance legged locomotion. *arXiv preprint arXiv:2605.24975*, 2026.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. *Advances in Neural Information Processing Systems*, 35:27906–27920, 2022.

Konstantin Schürholt, Michael W Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In *International Conference on Machine Learning*, pp. 43947–43966. PMLR, 2024.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Kai Wang, Dongwen Tang, Wangbo Zhao, Konstantin Schürholt, Zhangyang Wang, and Yang You. Recurrent diffusion for large-scale parameter generation. *arXiv preprint arXiv:2501.11587*, 2025.

Kang Xu, Chenjia Bai, Xiaoteng Ma, Dong Wang, Bin Zhao, Zhen Wang, Xuelong Li, and Wei Li. Cross-domain policy adaptation via value-guided data filtering. *Advances in Neural Information Processing Systems*, 36:73395–73421, 2023.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.

Dingqi Zhang, Antonio Loquercio, Jerry Tang, Ting-Hao Wang, Jitendra Malik, and Mark W Mueller. A learning-based quadcopter controller with extreme adaptation. *IEEE Transactions on Robotics*, 2025.

Andrey Zhmoginov, Mark Sandler, and Maksym Vladymyrov. Hypertransformer: Model generation for supervised and semi-supervised few-shot learning. In *International Conference on Machine Learning*, pp. 27075–27098. PMLR, 2022.

Hongtu Zhou, Ruiling Yang, Yakun Zhu, Haoqi Zhao, Hai Zhang, Di Zhang, Junqiao Zhao, Chen Ye, and Changjun Jiang. Certain: Context uncertainty-aware one-shot adaptation for context-based offline meta reinforcement learning. In *Forty-second International Conference on Machine Learning*, 2025a.

Pei Zhou, Wanting Yao, Qian Luo, Xunzhe Zhou, and Yanchao Yang. Hyper-goalnet: Goal-conditioned manipulation policy learning with hypernetworks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b.

Renzhe Zhou, Chen-Xiao Gao, Zongzhang Zhang, and Yang Yu. Generalizable task representation learning for offline meta-reinforcement learning with data limitations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17132–17140, 2024.

Luisa Zintgraf, Sebastian Schulze, Cong Lu, Leo Feng, Maximilian Igl, Kyriacos Shiarlis, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: Variational bayes-adaptive deep rl via meta-learning. *Journal of Machine Learning Research*, 22(289):1–39, 2021.

A Data Collection

A.1 Ant Policy

Training details We train policies using Brax SAC Freeman et al. (2021) algorithm in randomly sampled domains from training distribution specified by the front- and back-leg rescaling factors. We train each policy for 2×10^6 environment steps with the discount factor $\gamma = 0.997$. We use AdamW Loshchilov & Hutter (2019) with learning rate 6×10^{-4} for the actor, Adam Kingma & Ba (2014) with learning rate 6×10^{-4} for the critic, and Adam with learning rate 3×10^{-4} for the entropy temperature initialized at $\alpha = 1.0$. The actor is parameterized by an MLP with two hidden layers of width $[32, 32]$, while the critic with two hidden layers of width $[256, 256]$; both networks use ReLU activations. We use $8{,}192$ parallel environments to collect data, and minibatch size $16{,}384$ for each gradient update. The number of gradient updates per collection step is 128. The algorithm starts with a replay buffer prefilled with $16{,}384$ transitions, and maintains the replay buffer with at maximum 1×10^6 transitions. For evaluation, we compute the average episodic return using an episode length of $T = 1000$ for $N = 128$ parallel environments.

Training cost It takes 2 minutes to finish training, evaluate checkpoints and save them for a single seed on a Google TPU v4-8 pod.

A.2 Drone Racing Policy

Training details We train policies for the drone racing task on a lemniscate track, and we use dense reward shaping for faster convergence Pasumarti et al. (2025). Specifically, a progress reward at each rollout step encourages the drone to fly toward the next gate, while penalty terms discourage collisions with gates or the ground and regularize large body rates and tilt angles. The quadrotor is controlled at $48\,\mathrm{Hz}$ using collective-thrust-and-body-rate (CTBR) commands Kaufmann et al. (2022). At each control step, the policy receives a 21-dimensional observation expressed entirely in the drone body frame. Let $p_d \in \mathbb{R}^3$ denote the drone position and $R_d \in SO(3)$ its body-to-world rotation matrix. The observation consists of the linear velocity $R_d^\top v_d \in \mathbb{R}^3$, the angular velocity $R_d^\top \omega_d \in \mathbb{R}^3$, the world vertical axis $R_d^\top e_z \in \mathbb{R}^3$, which encodes roll and pitch, the relative positions of the current and next target gates, $R_d^\top (p_g^{\mathrm{cur}} - p_d) \in \mathbb{R}^3$ and $R_d^\top (p_g^{\mathrm{next}} - p_d) \in \mathbb{R}^3$, and the unit normals of these two gates, $R_d^\top n_g^{\mathrm{cur}} \in \mathbb{R}^3$ and $R_d^\top n_g^{\mathrm{next}} \in \mathbb{R}^3$, which point against the direction of travel.

We train policies using SAC algorithm Kim et al. (2026) in randomly sampled domains from training distribution specified by thrust force reductions. We use $16{,}384$ parallel environments. We use a replay buffer of 10^7 transitions; the agent then performs 32 gradient updates, each on a minibatch of $2{,}048$ transitions sampled uniformly from the buffer. Training runs for 2×10^8 environment steps. We use a discount factor $\gamma = 0.99$ with 4-step returns, clipped double-Q targets with soft target updates ($\tau = 0.005$), and the Adam optimizer with a learning rate of 3×10^{-4} for the actor, the critics and the entropy temperature. Gradient norms of the actor and the critics are clipped at 0.5. The temperature is tuned automatically toward a target entropy of $-|\mathcal{A}| = -4$, starting from $\alpha = 1$. Neither observations nor rewards are normalized. The actor is a squashed-Gaussian policy parameterized by an MLP with hidden layer widths $[64, 64]$, and each of the two critics is an MLP with hidden layer widths $[256, 256]$. We use tanh activations for both networks. For evaluation, we compute the average episodic return using an episode length of $T = 1440$ for $N = 128$ parallel environments.

Training cost It takes 8 minutes to finish training, evaluate checkpoints and save them for a single seed on a Google TPU v4-8 pod.

A.3 Legged Robot Policy

Training details We train policies on the Go1 Joystick task on the flat terrain from MuJoCo playground (Zakka et al., 2025), and we use the same observations, actions, reward designs, and environment parameters as reported in the paper. We use the default target tracking forward speed at $0.5\,\mathrm{m/s}$, with zero lateral or rotational speeds.

We adopt the SAC algorithm implementation from Sabatini et al. (2026) to train in MuJoCo playground with JAX and MJX. We train each policy for 3.9×10^7 environment steps and 400 iterations

with the discount factor $\gamma = 0.96$ and 5-step returns. We use Adam with learning rate 2×10^{-4} for the actor, Adam with learning rate 2×10^{-4} for the critic, both with gradient-norm clipping at 1.0, and Adam with learning rate 2×10^{-5} for the entropy temperature initialized at $\alpha = 0.001$. The actor is parameterized by an MLP with two hidden layers of width $[64, 64]$, while the critic consists of two Q-networks (clipped double-Q) with three hidden layers of width $[512, 256, 128]$ and a target network updated with $\tau = 0.003$; both networks use Swish activations and running observation normalizer. We use 4,096 parallel environments to collect data, and minibatch size 8,192 for each gradient update. Each iteration collects 24 steps from every environment and then performs 200 gradient updates. The algorithm starts updating once the replay buffer holds 196,608 transitions, and maintains the replay buffer with at maximum 5×10^6 transitions. For evaluation, we compute the average episodic return using an episode length of $T = 1000$ for $N = 128$ parallel environments.

Training cost It takes 11 minutes to finish training, evaluate checkpoints and save them for a single seed on a Google TPU v4-8 pod.

B Data Visualization

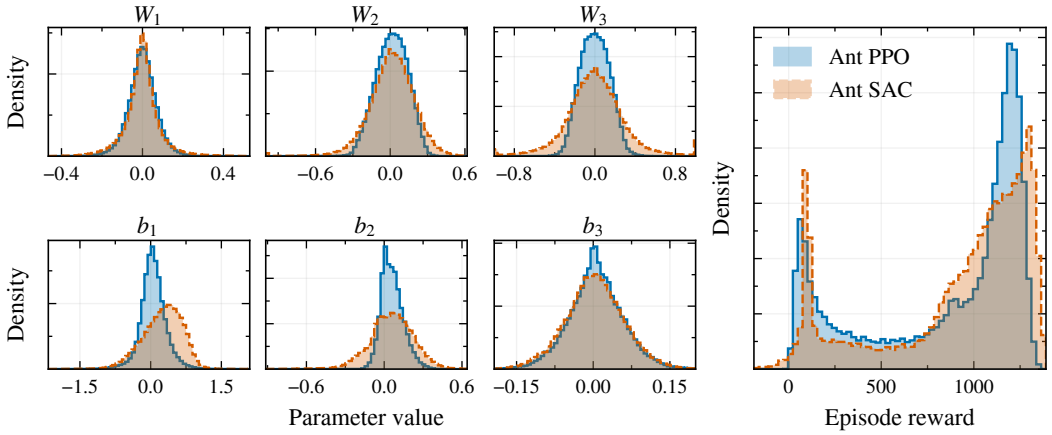


Figure 8: Visualization of the single-domain SAC and PPO policies on the ant environment. (Left) The weight distribution of the kernel W_i and bias b_i of the ith hidden layer. (Right) The 200-step episode reward distribution of the policies. Both the SAC and PPO policies have similar weight distribution for the input kernel and output bias, which indicates the input and output weights are more task-relevant. The similar reward distribution shows the specialist policies effectively learn good performance.

To better understand the differences in the weight-space structure of the SAC and PPO policies trained in the single domain, we uniformly sample 200 training runs and 25 checkpoints from each training run for visualization. We flatten the weight matrices in each layer and plot their distributions respectively, and we evaluate all policies using $T = 200$ step to get the episode reward distributions. As shown in Figure 8, both learning algorithms could effectively learn good performance in the ant environment. Given a good policy generator learned on both datasets, it could generate equally well-performed SAC and PPO policies. Besides, both policies have similar input kernel and output bias weights, which are highly task-dependent in continuous control problems. The differences in the middle layers further indicate potential deviations in their weight-space structures.

C Baseline Methods Training Details

C.1 Conditional Adaptation

DR−H. We train the `DR−H` policy using the same environment-step budget as the specialist policies with domain randomization. We train a recurrent SAC policy following the RESeL recipe Luo et al. (2024). Both the actor and the critic contain a context encoder: the current observation o_t, the previous observation o_{t-1} and the previous action a_{t-1} are linearly projected to 128 dimensions, concatenated, and passed through a fully connected layer of width 32, a GRU with hidden width 128

and a linear layer, which yields a 32-dimensional context embedding. The actor head is an MLP with the same hidden layer widths as the specialist policy's actor network, and it maps the concatenation of o_t and the embedding to a squashed-Gaussian policy. The critic has its own context encoder and two Q-heads with hidden layer widths $[256, 256]$ that take o_t, the action and the embedding. All hidden layers use ReLU activations. Training uses $8{,}192$ parallel environments and a per-environment replay buffer of $2{,}048$ steps. After every parallel environment step, the agent performs 16 gradient updates, each on a minibatch of $4{,}096$ subsequences of length 64. Over 2×10^8 environment steps this amounts to 3.9×10^5 gradient updates. We use clipped double-Q targets with soft target updates ($\tau = 0.005$), and the Adam optimizer with learning rates of 6×10^{-5} for the actor, 2×10^{-4} for the critic and 10^{-4} for the entropy temperature. Following RESeL, the GRU parameters of both networks use a much smaller learning rate of 2×10^{-6}. Gradient norms are clipped at 1.0. The actor and the temperature are updated once every two critic updates; the temperature starts at $\alpha = 0.2$.

GENTLE. We follow the experiment setup reported in (Zhou et al., 2024) to collect expert demonstrations as the offline dataset. We use the best checkpoint for each training run saved in the training dataset to rollout trajectories in the specific domain it was trained on. For each domain, we use 256 parallelized environment to collect demonstrations and save only the first episode in each environment to the buffer. We use a fixed buffer size for all tasks with 1×10^5 transitions. For the training details, we use the same hyperparameters as reported in the original paper Zhou et al. (2024) for all tasks. During test-time adaptation, we use the same 10 adaptation steps.

CERTAIN. We follow the data mixture methods reported in (Zhou et al., 2025a) to collect offline dataset. As mentioned in the paper, mixture of low-, mid- and high-quality improves the generalization ability of the method. We observed the similar phenomenon in the drone racing and legged robot environments, so we use mixed offline dataset for these two tasks. However, in the ant environment, offline dataset from the expert demonstrations gives the best performance, so we use the same expert dataset as `GENTLE`. We collect 1×10^5 transitions for each domain using four different checkpoints in the training run. We sample these four checkpoints whose evaluation returns are closest to 25%, 50%, 75% and 100% of the run's best return. During test-time adaptation, we use the same 10 adaptation steps.

RMA. We follow the implementation of the two-phase training regime as reported in Kumar et al. (2021). In the first phase, we train a teacher policy $\pi(x_t, \mu(e))$ with SAC, where e is the privileged domain parameter and the factor encoder μ is an MLP with hidden widths $[64, 64]$ that outputs an 8-dimensional latent. We use the same hyperparameters as described in Section A with domain randomization for each task training. In the second phase, we freeze the teacher's policy head and trains only ϕ to minimize $\|\phi(h_t) - \mu(e)\|^2$ on the student's own rollouts, where we use 50-step history observations h_t as inputs to the student policy. Transitions and their histories are stored in a per-environment replay buffer of 256 training steps, we use 32 updates per collection step. We use the same number of domains, environment steps, and episode lengths as described in Section A. We use Adam with learning rate 1×10^{-3} and minibatch size $4{,}096$.

RAPTOR*. We adopt a simple implementation of `RAPTOR` (Eschmann et al., 2026), where we use the same history observation encoder as `RMA`'s student policy and same actor head as the expert policy for each task. The policy is trained in each domain with its own rollouts supervised by the actions from the expert in that domain. Neither the expert or the `RAPTOR*` has privileged domain parameters as inputs. We use the same training recipe as the `RMA`'s phase-2 training.

C.2 Adaptation by Policy Generation

HyPoGen. We use the same expert demonstrations as `GENTLE` for the `HyPoGen` Ren et al. (2025) training for all tasks. Following the original implementation, we generate only the policy parameters and train with behavior cloning loss. The policy parameters have the same dimensions as ours in each task. We use the same hyperparameters as reported in the original paper Ren et al. (2025).

C.3 GRADIENT-BASED ADAPTATION

`CompFlow`. The policy is trained from scratch in the target domain with SAC using a fixed entropy temperature $\alpha = 0.2$, $\tau = 0.005$, and Adam with learning rate 3×10^{-4} for the actor and the critic. The actor is an MLP and the critic consists of two Q-networks, each with two hidden layers of width 256 and ReLU activations. For the first 10^5 gradient steps all source transitions are used; afterwards, each source minibatch keeps only the 70% with the smallest estimated dynamics gap. The policy collects one target transition every 200 gradient steps, which gives online budgets of 2×10^3 target transitions; the flow minibatch size is 128.

D GOL TRAINING DETAILS

Autoencoder We train with a batch size of 1024, AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay 5×10^{-3}) and cosine annealing, warming up over 10 epochs to a learning rate of 10^{-3} and then decaying to 0 over 90 epochs, where one epoch is $806{,}400$ samples. We clip gradients to a global norm of 0.1. We train on 16 TPU v5e chips, with the parameters sharded across chips (FSDP). We select the checkpoint with the highest alignment score, i.e., the R^2 between the returns of the input and the reconstructed policies over checkpoints of held-out test runs, each rolled out in its own domain. As shown in Table 1, the autoencoder effectively reconstruct high-fidelity policies with strong generalization ability.

Table 1: Alignment scores on the ant for GOL training. R_{rec} uses $N = 128$ checkpoints, and R_Δ uses $N = 128$ checkpoint pairs. Checkpoints come either from training runs or from held-out runs, whose domains lie inside the training range but were not seen in training. Higher is better, and 1 is perfect.

	Score	Training runs	Held-out runs
Autoencoder	R_{rec}	0.98	0.97
Evolution Module	R_Δ	0.98	0.85

Evolution Module We train with a batch size of 8192, AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay 0.1), and cosine annealing, warming up over 10 epochs to a learning rate of 10^{-4} and then decaying to 0 over the remaining 1.95×10^5 steps (2000 epochs). We clip gradients to a global norm of 0.02, and we maintain an exponential moving average (EMA) of the model parameters with decay 0.9999 during training and use the EMA weights for evaluation. We train on 16 TPU v5e chips with 512 samples per chip, sharding the parameters across chips (FSDP).

We use alignment score R_Δ as evaluation metrics for evolution module training as well. We take pairs of checkpoints from the same training run, give the model the earlier checkpoint, and prompt it with the normalized return of the later one. We then roll out the generated policy and compare the performance changes relative to the starting checkpoint, and the desired changes. As shown in Table 1 and Figure 9, the evolution module learns to generate policies with desired reward prompts perfectly in training domains and generalize well.

Training Cost We use a Google Cloud TPU v5e Pod slice with 16 TPU chips for both the behavior-aligned AE and the evolution module training. It takes around 10 hours to train AE for 10^5 steps, and around 20 hours to train the evolution module for 2×10^5 steps.

E GRADIENT-BASED ADAPTATION EXPERIMENT RESULTS

We compare against the state-of-the-art gradient-based adaptation method, `CompFlow` (Kong et al., 2026), on the ant environment. We use 256 in-distribution domains and 256 out-of-distribution domains and at most 10 rollouts are used for adaptation in the target domain δ for the other baseline methods. However, these make the comparison infeasible because `CompFlow` trains from scratch when transferring to a target domain δ.

In the few-shot adaptation setup for the ant task, we use 200-step rollout at each iteration and 10 iterations in total, which sums up to 2000 transitions in the target domain. As shown in Figure 10,

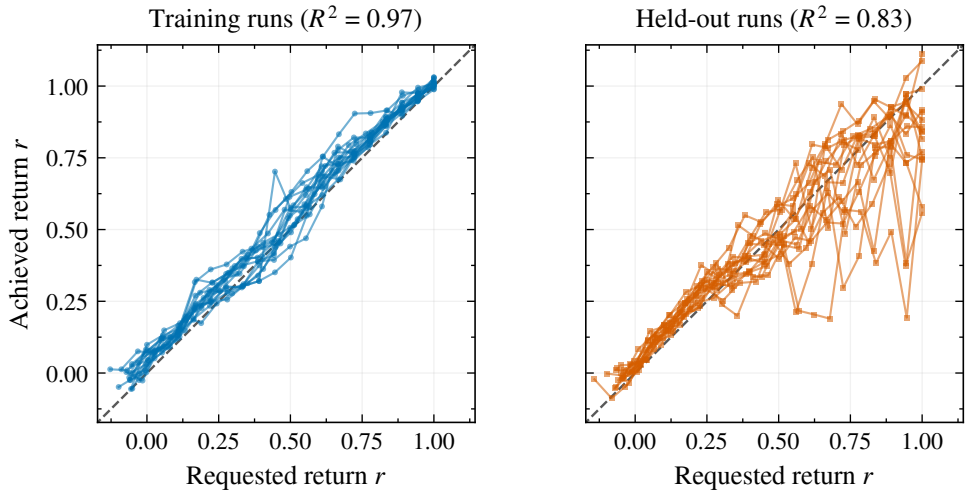


Figure 9: Alignment plot on the ant for the GOL evaluation. We sample initial checkpoints from training and held-out test training runs, and then prompt with various desired performance $r_d \in [0, 1]$. The plot shows the correlation between the evaluated performance normalized by the maximum rewards achieved in each domain, and the desired performance.

the performance of the policy degrades during the training due to large value estimation errors given 2000 target-domain interactions budget. The performance improves when increasing the budget to $40,000$, which is the same budget as reported in the original paper, but far more than our budget.

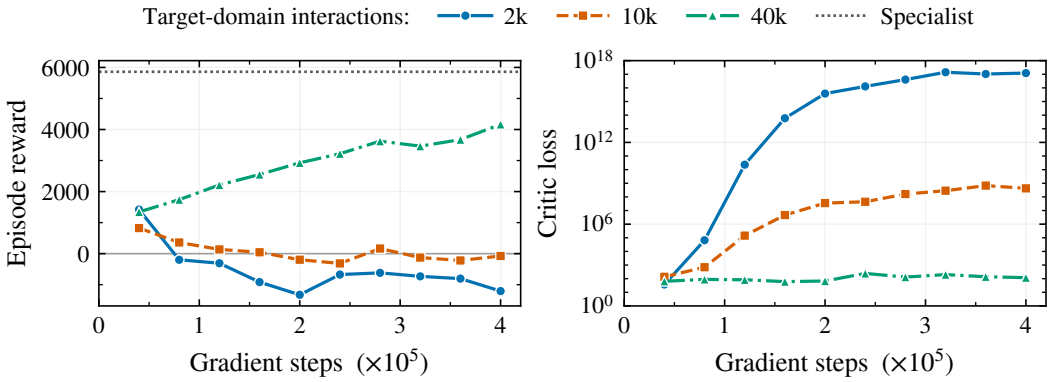


Figure 10: Training results of CompFlow (Kong et al., 2026) on the ant environment. We use the same number of gradient updates as reported in the original paper, and vary the target-domain interactions budget. (Left) Evaluated performance of the policy checkpoints saved during training. (Right) Critic prediction error during training.

F Ablation Study: Latent-Space Generation

Previous policy generation methods generate neural network weights and supervise on the action outputs induced by the generated policy on the expert demonstrations using BC loss Ren et al. (2025); Rezaei-Shoshtari et al. (2023). While the performance improves when using DAgger loss as shown in Section 4.3, it's computationally expensive to rollout policies in simulation for contact-rich environments, for example, legged robot. Therefore, we decouple training into two stages. In the first stage, we train a light-weighted trajectory-conditional autoencoder using fewer online rollouts and supervise using DAgger loss. Before the second-stage training, we collect rollouts for each policy checkpoint in the training dataset. In the second stage, we encode weights using the pretrained

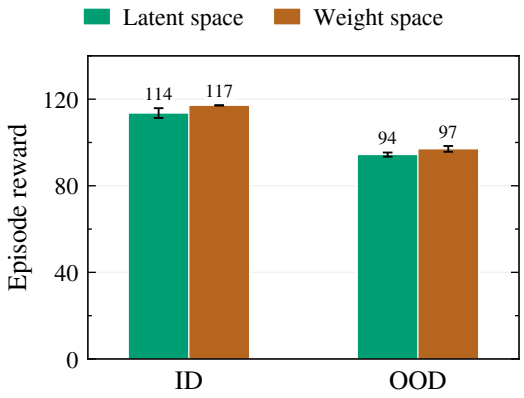


Figure 11: In the drone racing environment, policy generation in weight space and latent space give similar performance in both in-distribution and out-of-distribution domains.

autoencoder and train the evolution module in the latent space using MSE on the latents. Compared to the end-to-end training in the weight space, two-stage training in the latent space is $20\times$ faster in the legged robot environment, and $3\times$ faster in the drone racing environment. We compare the performance of our methods using end-to-end training and two-stage training regimes on the drone racing task, they are close in both in-distribution and out-of-distribution domains.