ADAPTIVE EXPERT GUIDANCE FOR EFFICIENT ON-POLICY REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

With massively parallel simulation, on-policy reinforcement learning (RL) methods such as proximal policy optimization (PPO) have become standard in many domains. However, learning from scratch is sample-inefficient and fails to exploit the potential existence of a suboptimal expert, such as a heuristic, a model-based controller, or a policy trained on a related task. Such an expert is often available and can guide early training, but its sub-optimality limits final performance. The challenge then becomes balancing expert guidance against learning from rewards. Existing methods set the expert's influence through a blending weight, a schedule, or an evaluation-driven curriculum. Alternatively, they adapt it with additional learned components such as critics over expert actions or auxiliary agents. However, none optimizes it using the same on-policy objective as the policy itself. We propose a method in which the learner and the expert alternate control within each training episode, and the expert's share of control is a single learnable parameter optimized jointly with the policy. The learner benefits from the expert early in training, but its share of control declines as the learner becomes more competent, until eventually vanishing completely. This leaves the learner acting alone and better than the suboptimal expert. We evaluate our method on 34 tasks across two benchmarks, spanning discrete and continuous action spaces, using both learned and model-based experts. Our method improves sample efficiency over guided and unguided baselines while requiring minimal hyperparameter variation. The expert's share decays to zero as the learner improves, vanishing when the expert is no longer useful.

1 INTRODUCTION

Reinforcement learning (RL) has demonstrated remarkable ability to solve complex sequential decision-making problems (Silver et al., 2018a; Andrychowicz et al., 2020; Kaufmann et al., 2023; Guo et al., 2025), but it requires many environment interactions, as the agent must explore the usually high-dimensional state space before it can discover and exploit high-reward behaviors. GPU-parallel simulation (Makoviychuk et al., 2021) favors on-policy methods such as proximal policy optimization (PPO) (Schulman et al., 2017), because a single GPU can step thousands of environments in parallel and cut the wall-clock cost of collecting experience. Off-policy and model-based methods need fewer interactions (Haarnoja et al., 2018; Hansen et al., 2024) but require more computation per interaction. Their higher per-interaction cost therefore limits the wall-clock gains they can obtain from massively parallel simulation. In this paper, we therefore focus on on-policy methods.

In practice, training does not need to start from scratch (Agarwal et al., 2022) when an *expert* is available, that is, an algorithm that delivers a useful, though not necessarily optimal, policy. Examples include controllers and planners built on approximate physical models (Carius et al., 2020), hand-engineered policies (Uchendu et al., 2023), previously trained agents (Rusu et al., 2016; Schmitt et al., 2018), and agents trained on a related task (Taylor & Stone, 2009; Zhu et al., 2023). Imitating the expert's actions limits the learner to the expert's quality (Ross et al., 2011; Carius et al., 2020). Other methods keep the expert in the control loop, and therefore in the deployed system, through residual control (Silver et al., 2018b; Johannink et al., 2019; Jeon et al., 2025), uncertainty-weighted fusion of expert and learner actions (Rana et al., 2020; 2023), model predictive control (MPC) as an optimization layer (Reiter et al., 2023; Romero et al., 2026; Adabag et al., 2025), or MPC whose value estimates are blended with a learned value function (Bhardwaj et al., 2021).

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.

Since the expert is suboptimal, training should exploit its guidance early and then hand control to the learner, so that the learner can improve beyond the expert and act alone after training. Most existing approaches set the expert's influence independently of the learning objective. Expert-regularized RL decays the imitation weight on a schedule (Rajeswaran et al., 2018) or tunes it by population-based training (Schmitt et al., 2018). When the expert contributes experience during training, its share is shortened on an evaluation-driven curriculum (Uchendu et al., 2023), decayed on a schedule (Yang et al., 2023), or kept at a fixed ratio of pre-collected expert traces to learner samples (Yan et al., 2025). Methods that adapt the expert's role to its value rely on a learned selector or a critic over the expert's actions (Nair et al., 2018; Xie et al., 2018; Kurenkov et al., 2019; Zhang et al., 2023), or on a counterfactual agent that learns from rewards alone (Shenfeld et al., 2023).

We instead propose a guided policy-gradient algorithm in which the expert and learner share control during training, with the expert's share represented by a single parameter optimized under the same return objective as the policy. We show that its gradient corresponds to the *expert advantage*: the expected gain in return from letting the expert act instead of the learner, from the same states. Naively learning this share is ineffective: switching policies at every step makes the comparison too noisy, while optimizing return alone does not necessarily withdraw an expert. We therefore commit to one policy for blocks of k steps, yielding a more reliable comparison, and introduce a price on expert use, making withdrawal part of the objective. An imitation term on expert blocks further trains the learner whenever the expert acts. Once the learner catches up, the expert's share decreases to zero and training reduces to standard policy gradient.

The proposed algorithm is evaluated on 34 tasks across two common benchmarks spanning discrete and continuous action spaces, with minimal hyperparameter variation. The experts, learned or model-based, range in quality from near-optimal to substantially worse than the converged learner. Our method improves sample efficiency over guided and unguided baselines, and the learner asymptotically recovers the performance of Proximal Policy Optimization (PPO) or better, so the expert's suboptimality does not limit the final policy. The rate of expert withdrawal adapts to expert quality, with weaker experts withdrawn faster.

2 Related Work

We group prior work by how the expert enters training: as a source of target actions (imitation), as an extra term in the objective (regularization), or as a policy that collects part of the rollouts. Each group must decide how strongly and for how long the expert should influence the learner, the question that our method answers with a learned expert share. Our method belongs to the third group, and the last paragraph relates its block structure to temporal abstraction.

Imitating an expert. Behavior cloning suffers from compounding errors off the demonstration distribution, which dataset aggregation (DAgger) (Ross et al., 2011) corrects by querying the expert on the states the learner visits, and a learner that regresses onto a fixed expert's actions inherits the expert's ceiling. Guided policy search (Levine & Koltun, 2013; Levine et al., 2016) and expert iteration (Anthony et al., 2017; Silver et al., 2018a) avoid that ceiling with a teacher that improves during training, whereas our expert is fixed and imitation acts only as a regularizer that vanishes once the expert is withdrawn. Other methods surpass a fixed expert by combining imitation with reward-driven updates: LOKI (Cheng et al., 2018) imitates for a randomized number of iterations before switching to policy gradient, while MAMBA (Cheng et al., 2020) and RPI (Liu et al., 2024) imitate black-box oracles state-wise, wherever an oracle's estimated value exceeds the learner's. RPI is the closest of these to our setting, but its handover compares value functions learned from each oracle's rollouts, whereas we use the learner's own critic on the training rollouts.

Expert-regularised reinforcement learning. A second family adds an expert-derived term to the reward objective: a large-margin (Hester et al., 2018) or behavior-cloning (Nair et al., 2018) loss on demonstrations, an imitation term in the policy gradient (Rajeswaran et al., 2018), or distillation from a teacher on the student's rollouts (Schmitt et al., 2018; Agarwal et al., 2022). The learner can surpass the expert, so the design question is the regularizer weight. The Q-filter of Nair et al. (2018) applies imitation only where the critic prefers the demonstrated action, and TGRL (Shenfeld et al., 2023) adapts the weight by comparing the guided agent with a reward-only counterfactual agent.

Experts that act during training. Several methods let the expert control part of the rollout and shrink that part over training: a prefix of each episode shortened by a curriculum in Jump-Start

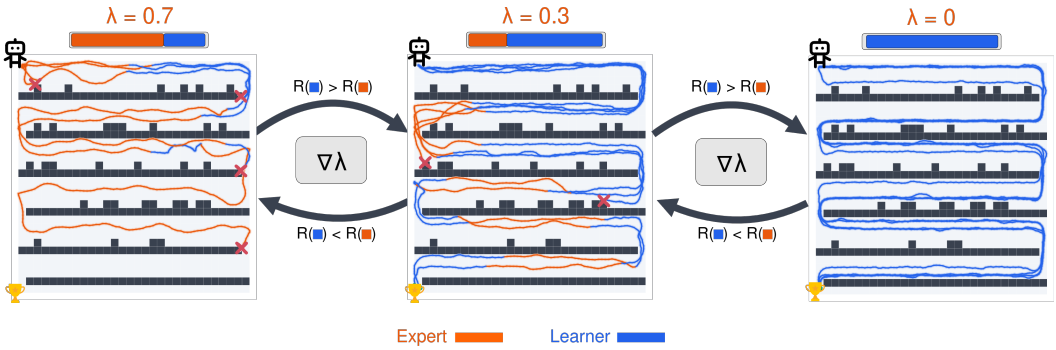


Figure 1: **The expert share is learned from block-level performance.** The expert (orange) and learner (blue) alternate control in temporally committed blocks, with λ determining the expert's share. The difference between expert- and learner-block advantages drives the update of λ: as the learner improves, the expert is progressively withdrawn until $\lambda = 0$.

Reinforcement Learning (JSRL) (Uchendu et al., 2023), or a probability of acting decayed on a schedule (Yang et al., 2023). Others arbitrate between expert and learner with a learned selector or critic, at every step (Xie et al., 2018; Zhang et al., 2023) or with commitment until a Bayesian critic favors switching (Kurenkov et al., 2019). In our method, blocks have fixed length and the switch ignores the state, so the share gradient reduces to a comparison of expert and learner block advantages, which we estimate from the training rollouts with the learner's own critic. No selector, critic over expert actions, evaluation phase, or auxiliary agent is needed, and the share follows the same objective as the learner, so withdrawal is part of it.

Temporal abstraction. Our method can be connected to the options framework (Sutton et al., 1999), in which an agent chooses among temporally extended actions. Each of our blocks can be seen as the execution of one of two such options, the expert or the learner, which last a fixed number of steps and of which only the learner is trained. Under this view, our learner update corresponds to the intra-option policy gradient of option-critic (Bacon et al., 2017), which learns a set of options that together solve the task, whereas we learn to withdraw one of ours.

3 Method

Our approach belongs to the policy gradient family of algorithms, with the learner's parameters optimized from on-policy rollouts. The key difference is that the expert and the learner take turns controlling the agent during these rollouts. As shown in Fig. 1, we split each training episode into blocks of a fixed number of steps; at the start of each block, a random draw decides whether the expert or the learner acts for the entire block. We refer to the probability of selecting the expert for a given block as the *expert share*, which is automatically adjusted as training proceeds. It starts high, so that the expert's blocks can bring the agent to states the learner is unlikely to reach on its own. For instance, in tasks with sparse rewards, these are states where rewards appear and generate the non-zero gradients the learner can benefit from. As the learner becomes more proficient, the expert share progressively diminishes, eventually vanishing altogether.

This block structure is what makes the automated adjustment of the expert share informative. Because each block is executed entirely by either the learner or the expert, we can attribute its outcome to one of the two, and compare how well the expert's blocks and the learner's blocks perform within the same episode. We update the expert share with the same policy-gradient rule that trains the policy network weights. These updates have an intuitive interpretation: they raise the share when the expert's blocks outperform the learner's and lower it once the learner catches up. The expert thus hands over control as the learner improves, without requiring a predetermined schedule. Blocks are necessary because switching at every step would blur this comparison, since the outcome of a single action depends mostly on the actions that follow it. Two additional mechanisms complete the method. First, we impose a cost on every block played by the expert, so that the expert retains control only while it is clearly better than the learner. This is necessary because, without it, the expert would keep its share as long as the learner merely matches it, and the learner might never act alone. Second, on the expert's blocks, we train the learner to imitate the expert's actions, so that the

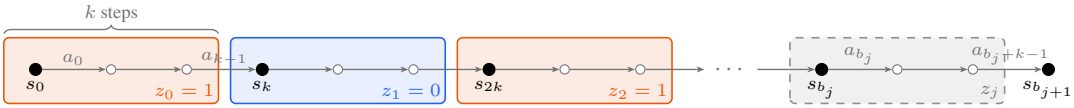


Figure 2: **Temporally committed mixture.** The horizon is split into blocks of k steps, starting at $b_j = jk$ (filled states). At each boundary, a switch $z_j \sim \text{Bernoulli}(\lambda)$ hands the whole block to the expert $\hat{\pi}$ (orange, $z_j = 1$) or to the learner π_θ (blue, $z_j = 0$).

experience the expert collects also contributes to training the learner; this ensures fuller use of each episode for learner training. Once the expert share reaches zero, the expert is no longer used. Training continues as in ordinary policy gradient, and the final learner acts entirely on its own. Sect. 3.1 formalizes the problem. Sect. 3.2 derives and interprets the gradients for the learner and the expert share. Sect. 3.3 adds the cost and the imitation term.

3.1 Problem Formulation

Consider a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$ with state space $\mathcal{S} \subseteq \mathbb{R}^n$, action space $\mathcal{A} \subseteq \mathbb{R}^m$, transition kernel $P(\cdot \mid s, a)$, reward $r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, initial state distribution ρ_0, and discount $\gamma \in [0, 1)$. Let $\hat{\pi}$ be a fixed *expert* policy and π_θ a *learner* policy with parameters $\theta \in \mathbb{R}^d$. The objective of learning is to maximize the expected discounted return under the learner, i.e., $\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t\geq 0}\gamma^t r_t\right]$.

We partition each episode into blocks of k steps and denote $b_j = jk$ for the start time of block j. At the beginning of each block, a switch variable $z_j \sim \text{Bernoulli}(\lambda)$, where λ is the expert share, determines whether the expert or the learner rolls out that block. The switches are independent across blocks and independent of the states. Formally, for $z_j \in \{0, 1\}$, the rollout within block j is given by

$$a_{b_j+i} \sim \pi_{z_j}(\cdot \mid s_{b_j+i}), \quad s_{b_j+i+1} \sim P(\cdot \mid s_{b_j+i}, a_{b_j+i}), \quad i \in \{0, \ldots, k-1\}, \tag{1}$$

where $\pi_1 := \hat{\pi}$ and $\pi_0 := \pi_\theta$. We constrain $\lambda \in [0, \lambda_{\max}]$ with $\lambda_{\max} < 1$, so that the learner has a nonzero probability of acting in every block. We refer to the policy yielding these interleaved rollouts as the *mixture policy*. Fig. 2 illustrates the block structure. Unless indicated otherwise, expectations are over the joint distribution of the switches $\boldsymbol{z} = (z_0, z_1, \ldots)$ and the trajectories $\tau = (s_0, a_0, s_1, a_1, \ldots)$ they induce, with $s_0 \sim \rho_0$.

Denote the discounted reward of block j by $\mathcal{R}_j := \sum_{i=0}^{k-1}\gamma^i r_{b_j+i}$ and the block discount factor by $\Gamma := \gamma^k$, and note that the discounted return of an episode can be written with block rewards, $\sum_{t\geq 0}\gamma^t r_t = \sum_{j\geq 0}\Gamma^j \mathcal{R}_j$. We optimize θ and λ for the discounted return of the mixture policy,

$$J(\lambda, \theta) = \mathbb{E}\Big[\sum_{j\geq 0}\Gamma^j \mathcal{R}_j\Big], \tag{2}$$

which equals the learner's objective when $\lambda = 0$. At block boundaries, the rollouts form an MDP with state s_{b_j}, action z_j, reward $\mathcal{R}_j$, and discount Γ.

3.2 Policy Gradient Estimator under Block Rollouts

On the MDP at block boundaries, define the block-wise switch-value function and value function

$$Q(s, z) := \mathbb{E}\Big[\sum_{m\geq 0}\Gamma^m \mathcal{R}_{j+m} \;\Big|\; s_{b_j} = s,\ z_j = z\Big], \qquad V(s) = \lambda Q(s, 1) + (1-\lambda)Q(s, 0), \tag{3}$$

where $V(s)$ is derived by taking the expectation of $Q(s, z)$ with respect to z_j. The block advantage $A(s, z) := Q(s, z) - V(s)$ compares committing the block at s to policy z with the average over the switch. We estimate it with the block temporal-difference residual $\widehat{A}_j := \mathcal{R}_j + \Gamma V(s_{b_{j+1}}) - V(s_{b_j})$.

The gradients of the objective equation 2 with respect to the learner parameters θ and the expert share λ are given by

$$\boxed{\nabla_\theta J = \mathbb{E}\big[\sum_{j\geq 0}\Gamma^j(1 - z_j)\widehat{A}_j \sum_{i=0}^{k-1}\nabla_\theta \log \pi_\theta(a_{b_j+i} \mid s_{b_j+i})\big]} \tag{4}$$

$$\boxed{\frac{\partial J}{\partial \lambda} = \mathbb{E}\big[\sum_{j\geq 0}\Gamma^j \frac{z_j - \lambda}{\lambda(1-\lambda)}\widehat{A}_j\big]} \tag{5}$$

Eq. 4 implies that the learner is updated only on the blocks it rolled out, with the block advantage in place of the per-step advantage, and Prop. 1 below gives the λ-gradient a simple reading. When $\lambda = 0$, we have $z_j = 0$ for every block, and equation 4 reduces to the standard actor-critic gradient for π_θ at block resolution. At this point, the block structure is unnecessary: no switching occurs and a single policy acts throughout, so training can revert to per-step updates with the per-step advantage. We now derive eq. 4 and 5.

Derivation. Write $\tau_j := (a_{b_j}, s_{b_j+1}, \ldots, a_{b_{j+1}-1}, s_{b_{j+1}})$ for the trajectory within block j, excluding the initial boundary state. The joint density of τ_j and the switch z_j, conditioned on the boundary state s_{b_j} is given by

$$p(z_j, \tau_j \mid s_{b_j}) = \underbrace{\lambda^{z_j}(1-\lambda)^{1-z_j}}_{p(z_j)} \cdot \underbrace{\prod_{i=0}^{k-1} \pi_{z_j}(a_{b_j+i} \mid s_{b_j+i})\, P(s_{b_j+i+1} \mid s_{b_j+i}, a_{b_j+i})}_{p(\tau_j \mid s_{b_j}, z_j)}, \tag{6}$$

where the first factor does not depend on s_{b_j} because the switch is independent of the state. The joint density of the trajectory τ and the switches $\boldsymbol{z}$ is then the block-wise product $p(\boldsymbol{z}, \tau) = \rho_0(s_0) \prod_{j \geq 0} p(z_j, \tau_j \mid s_{b_j})$. Writing $\log \pi_{z_j}(a \mid s) = z_j \log \hat{\pi}(a \mid s) + (1 - z_j) \log \pi_\theta(a \mid s)$ and noting that $\nabla_\theta \log \hat{\pi}(a \mid s) = 0$ for the expert for all a, s, it follows that the score for θ is

$$\nabla_\theta \log p(\boldsymbol{z}, \tau) = \sum_{j \geq 0} \nabla_\theta \log p(z_j, \tau_j \mid s_{b_j}) = \sum_{j \geq 0} (1 - z_j) \sum_{i=0}^{k-1} \nabla_\theta \log \pi_\theta(a_{b_j+i} \mid s_{b_j+i}), \tag{7}$$

and the score for λ is

$$\frac{\partial \log p(\boldsymbol{z}, \tau)}{\partial \lambda} = \sum_{j \geq 0} \frac{z_j - \lambda}{\lambda(1-\lambda)}. \tag{8}$$

Neither score contains the expert's action probabilities, so the expert can be a black box that only returns actions. Writing ∇ for either ∇_θ or ∂_λ, the policy gradient theorem (Sutton et al., 2000) yields $\nabla J = \mathbb{E}\big[\sum_{j \geq 0} \Gamma^j \nabla \log p(z_j, \tau_j \mid s_{b_j})\, \widehat{A}_j\big]$, and substituting eq. 7 and 8 yields equation 4 and equation 5.

Reading the λ-gradient. The λ-gradient in equation 5 has a simple reading once the switch is marginalized out. Let

$$a_E(s) := \mathbb{E}\big[\widehat{A}_j \mid s_{b_j} = s,\ z_j = 1\big], \qquad a_A(s) := \mathbb{E}\big[\widehat{A}_j \mid s_{b_j} = s,\ z_j = 0\big] \tag{9}$$

be the expected advantage of a block that starts in s and is executed by the expert or by the learner.

Proposition 1. *Because the switch is independent of the boundary state,*

$$\frac{\partial J}{\partial \lambda} = \sum_{j \geq 0} \Gamma^j\, \mathbb{E}_{s_{b_j}}\big[a_E(s_{b_j}) - a_A(s_{b_j})\big], \tag{10}$$

where the expectation is over the marginal distribution of the boundary states that the mixture visits.

We prove Prop. 1 in App. A. Because the switch ignores the state, the expert's blocks and the learner's blocks start from the same distribution of states, so equation 10 compares the two policies on equal footing: the gradient is positive when, on the states the mixture visits, a block executed by the expert earns more in expectation than one executed by the learner, and negative otherwise. Gradient ascent therefore raises λ while the expert is ahead on average and lowers it once the learner is ahead. Since $a_E(s) = A(s, 1)$ and $a_A(s) = A(s, 0)$, the difference also equals $Q(s, 1) - Q(s, 0)$: the extra return from s when the expert, not the learner, plays the next block.

3.3 Penalty on expert use and imitation

Penalizing expert use. Without a penalty, λ decreases only when the learner's blocks strictly outperform those of the expert. Once the learner matches the expert, which the imitation term

below encourages, the gradient equation 5 vanishes and λ remains at its current value, potentially preventing the learner from acting independently. We therefore maximize

$$\Phi(\lambda, \theta) = J(\lambda, \theta) - c_E \, \sigma \, \lambda, \qquad c_E > 0, \tag{11}$$

where σ is the sample standard deviation of the block advantage over a batch, treated as a constant in the gradient, and c_E is the price of expert use in units of σ. Expressing the price in units of σ avoids tuning it per task. The price does not depend on θ, so it changes only the λ-gradient: the expert share increases only while the expert's blocks outperform the learner's on average by more than a margin proportional to $c_E\sigma$, and decreases once the margin falls below it.

Imitation on expert blocks. The factor $(1 - z_j)$ in equation 4 removes the expert's blocks from the learner update, so a fraction λ of the collected experience does not train the learner. We use it through an imitation term on those blocks,

$$\mathcal{I}(\theta) = \mathbb{E}\Big[\sum_{j \geq 0} \Gamma^j \, z_j \sum_{i=0}^{k-1} \log \pi_\theta(a_{b_j+i} \mid s_{b_j+i}) \Big], \tag{12}$$

weighted by $c_I \geq 0$ and differentiated with the sampling distribution held fixed, as is standard for auxiliary losses. As in DAgger Ross et al. (2011), imitation uses states the learner helps reach rather than fixed demonstrations. Imitating a mediocre expert can hold the learner back, so we set c_I per benchmark (Sect. 4.5), and the term fades with the expert because it acts only on expert blocks. We do not differentiate $\mathcal{I}$ with respect to λ: doing so would let λ rise to generate more blocks to imitate, whereas it should respond only to the block comparison and its price.

Resulting update. The learner ascends $J + c_I \, \mathcal{I}$ and the expert share ascends Φ so the complete gradients are:

$$\begin{aligned} \nabla_\theta \big(J + c_I \, \mathcal{I}\big) &= \mathbb{E}\Big[\sum_{j \geq 0} \Gamma^j \big[(1 - z_j) \, \widehat{A}_j + c_I \, z_j\big] \sum_{i=0}^{k-1} \nabla_\theta \log \pi_\theta(a_{b_j+i} \mid s_{b_j+i}) \Big], \\ \frac{\partial \Phi}{\partial \lambda} &= \mathbb{E}\Big[\sum_{j \geq 0} \Gamma^j \, \frac{z_j - \lambda}{\lambda(1 - \lambda)} \, \widehat{A}_j \Big] - c_E \, \sigma, \end{aligned} \tag{13}$$

with λ projected onto $[0, \lambda_{\max}]$ after each update. On learner blocks, the learner follows the policy gradient; on expert blocks, it imitates the expert. Once $\lambda = 0$, no expert block is sampled, the imitation term vanishes, and training is standard policy gradient.

4 Experiments

We address four questions. **(Q1)** Does the learned handover yield higher returns with fewer environment steps than guided and unguided baselines? (Sect. 4.2) **(Q2)** Does λ correlate well with the expert's declining superiority over the learner and eventually vanish, as equation 13 predicts? (Sect. 4.3) **(Q3)** Can a weak expert that often fails help and enable the learner to surpass it? (Sect. 4.4) **(Q4)** How much does each hyper-parameter, the block length k, the imitation weight c_I, and the expert price c_E, matter? (Sect. 4.5)

4.1 Experimental Setup

Benchmarks. We evaluate on 34 tasks from two common benchmarks, both in the GPU-parallel implementations that motivate our on-policy setting. DMControl (Tunyasuvunakool et al., 2020), through MuJoCo Playground (Zakka et al., 2025), covers continuous control at varying difficulty, including sparse-reward tasks. MiniGrid (Chevalier-Boisvert et al., 2023), through NAVIX (Pignatelli et al., 2026), adds sparse-reward gridworlds with discrete actions. Each method trains for 20M environment steps on 5 seeds per task.

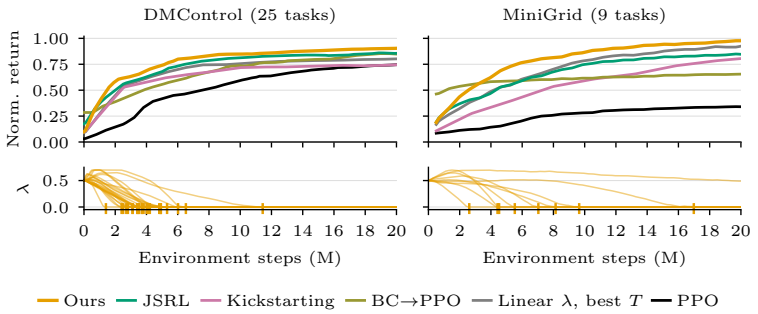


Figure 3: **Aggregate results on both benchmarks.** Normalized return (min-max per task, averaged over tasks and seeds) and the expert share λ per task.

Baselines. All methods share the same PPO learner, network, and optimizer settings (App. C). They differ only in how they use the expert. *PPO* (Schulman et al., 2017) is the unguided baseline. *BC → PPO* follows Ramrakhya et al. (2023) in its simplest form: M steps of behavior cloning (BC), then PPO. *Scheduled* λ uses the same block structure and imitation component, but we modify the learned share with the schedule $\lambda(t) = \max(0, 1-t/T)$, for $T \in \{1, 2.5, 5\} \times 10^6$ steps as proposed by Yang et al. (2023). *JSRL* (Uchendu et al., 2023) lets the expert act for a prefix of each episode and shortens the prefix as the learner improves. JSRL was designed for off-policy learning, so we exclude expert actions from the policy-gradient loss and keep them for value learning and BC. *Kickstarting* (Schmitt et al., 2018) adds a distillation loss toward the expert on the learner's rollouts and with a population-based adaptation.

Experts. Our target setting is an expert that comes from outside the training run, such as a controller, a planner, or a heuristic, available as a black box that returns actions. On MiniGrid, the expert is of this kind: a depth-first-search (DFS) planner with injected noise, tuned to reach about 70% of the maximum return on average across tasks. On DMControl, handcrafting such an expert for each task is impractical, so we emulate it with learned experts: PPO checkpoints stopped before convergence and TD-MPC2 policies (Hansen et al., 2024), which reach about 60% of the best final return of any method on average. Our method never reads their parameters or resumes training from them; it only queries them for actions. App. D reports per-task expert returns, and we will release all experts. Sects. 4.3 and 4.4 add two model-based experts that plan with inaccurate models: an MPC controller with a perturbed dynamics model and a grid planner with a PD tracker.

Metrics. Asymptotic return is the deterministic return averaged over the final 20% of training. Steps to bar is the number of environment steps needed to first reach 75% of the best asymptotic return any method achieved on that task, so the bar is the same for all methods; runs that never reach it count as 20M in medians. In tables, normalized return is the return divided by the best method's return on that task. Aggregate curves normalize each task by its observed range.

4.2 Comparison of Return Performance

Tab. 1 summarizes both metrics by task family, and Fig. 3 shows the aggregate learning curves and the learned expert share λ; per-task tables, curves, and λ trajectories are in App. B. On DMControl, our method has the lowest median steps to bar, 4.8M against 5.6M for JSRL, 6.1M for BC→PPO, 9.1M for Kickstarting, and 10.1M for PPO, a median per-task speed-up of 2.55× over PPO. It also has the highest mean normalized return, and it is the fastest method on 16 of 25 tasks, while no other method is fastest on more than 5 (Tab. 2). On MiniGrid, reward is sparse, and the methods separate more sharply. PPO reaches the bar on only 3 of 9 tasks and ends with zero return on three of the four KeyCorridor variants. BC→PPO reaches the bar fastest when cloning covers most of the state space, but it collapses on the two largest KeyCorridor variants and on DoorKey-Random-16x16. Because our method queries the expert on the states the learner itself visits, it avoids this collapse: it obtains the highest final return on 4 tasks and ties for the highest on 3 more (Tab. 3). The scheduled-λ arms show why the share has to be learned rather than fixed in advance. No single linear schedule horizon T works across all tasks at the same time. Our method beats every baseline on both metrics in every family except on Finger and steps to bar in MiniGrid, and the λ trajectories it learns share no common shape (Fig. 3).

Table 1: **Results by task family.** Median steps to bar (M) / mean normalized return; runs that never reach the bar count as 20M. **Bold**: best, underline: second best. Group definitions and per-task results in App. B.

	DMControl				
	Low-dim.	Finger	Loco.	MiniGrid	All
Ours	**1.7/.952**	7.7/.981	**5.5/.957**	7.8/**.960**	**5.1/.958**
JSRL	3.3/.902	7.4/.838	5.6/.933	7.9/.835	5.8/.889
Kick	4.2/.820	14.8/.744	16.0/.781	16.7/.775	10.1/.789
BC→PPO	5.7/.920	**5.7/.989**	9.2/.856	**4.3**/.649	6.0/.834
λ1M	3.3/.944	8.7/.891	7.4/.726	8.7/.771	7.0/.823
λ2.5M	4.6/.944	7.4/.942	7.4/.683	9.1/.906	6.6/.849
λ5M	5.0/.944	10.3/.934	10.3/.718	12.4/.907	8.7/.860
PPO	6.0/.807	10.1/.899	12.6/.730	20.0/.336	11.9/.665

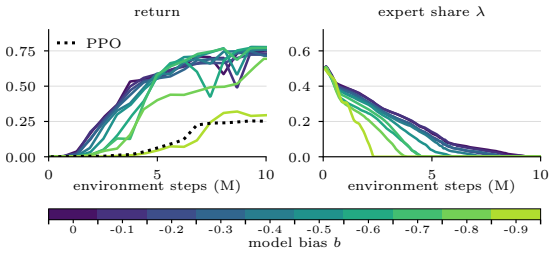


Figure 4: CartpoleSwingupSparse with an MPC expert whose model masses are scaled by $(1 + b)$. **Left:** learner return. **Right:** learned expert share λ. $b = 0$ is the exact model; as the error grows, λ reaches zero sooner.

4.3 Does λ track expert quality?

Sect. 4.2 varies the expert and the task together, so here we fix the task and vary only the expert's quality, using a different type of expert to also test generality. Following Bhardwaj et al. (2021), we use a model predictive path integral (MPPI) controller on CartpoleSwingupSparse whose internal model scales the cart and pole masses by $(1 + b)$, with $b \in \{0, -0.1, \ldots, -0.9\}$; at $b = 0$, the model is exact. The environment itself never changes. Fig. 4 shows that the handover adapts to expert quality: λ reaches zero at 9.8M steps for the exact model and at 2.2M for $b = -0.9$, with the intermediate levels ordered in between. No hyperparameters change across the sweep. The learner reaches about 0.7 normalized return at every level of mismatch except $b = -0.9$. There, the expert helps little, the handover withdraws it within 2.2M steps, and the learner ends still above PPO: with a poor expert, the method automatically falls back to PPO rather than being held back.



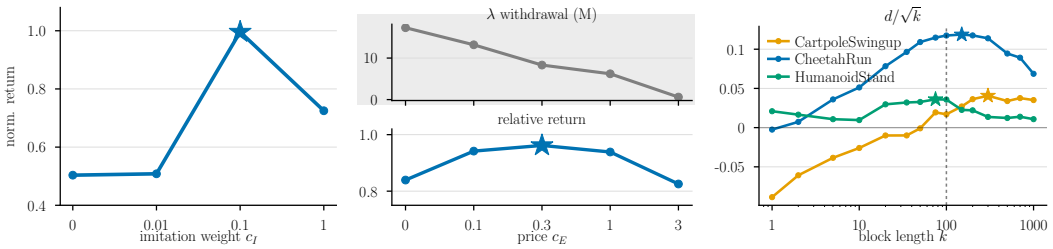
Figure 5: **Frozen Lake with a failing expert** ($p_{\text{slip}} = 0.3$). 60 evaluation episodes per method on the same map, with the success rate above each panel. Only our learner reliably completes the route.

4.4 Can a failing expert help?

In the above examples, we make the experts suboptimal on purpose. Here we explore a scenario common in practice: the expert can fail because it plans with an oversimplified model. In our continuous version of the *Frozen Lake Problem*, a point mass must cross a 36×36 lake by commanding 2D accelerations. A third of the cells are holes that end the episode, some lined up in rows with

alternating gaps that force a serpentine route. Reward comes only at the goal. The ice is slippery: with probability $p_{\text{slip}} = 0.3$, the commanded acceleration is rotated by $\pm 90°$. The expert uses a breadth-first search PD controller to track the shortest grid path. Because this path grazes the holes and the plan ignores the slip, the expert never reaches the goal (Fig. 5, full setup in App. B.3). Exploration is hard in this continuous task, with PPO never leaving the start. Our learner keeps the expert's route but learns to avoid the holes. It reaches the goal in 98.5% of the episodes, against 4% for Kickstarting, 28% for JSRL, and 0% for BC→PPO. Thus, a failing expert can help if the learner may deviate where it fails.

4.5 Ablations



(a) (b) (c)

Figure 6: **Hyperparameter ablations.** (a) Imitation weight: normalized return vs. c_I (Fig. 11). (b) Expert price: withdrawal step of λ (top) and relative return (bottom) vs. c_E (Fig. 12). (c) Block size trade-off: dotted line marks k=100. Stars mark the maximum. Per-task plots in App. B.4.

Imitation regularization. We sweep $c_I \in \{0, 0.01, 0.1, 1\}$ (Fig. 6a). The term matters where exploration is hard: for instance, on the Humanoid tasks, removing it leaves the learner at the floor for the entire budget. Excessive imitation instead anchors the learner to the expert and hinders exploration, so $c_I = 0.1$ gives the best trade-off across tasks (per-task curves in Fig. 11).

Block size. By Prop. 1, λ moves with the gap $a_E - a_A$. We measure its visibility on a block by the per-block signal-to-noise ratio $d(k) := \mathbb{E}[a_E - a_A]/\sigma$, where σ is the standard deviation of the block advantage (untrained learner, $\lambda = 0.5$, no critic). Maximizing d alone favors long blocks, but a batch of N steps contains N/k blocks, giving gradient signal-to-noise $d(k)\sqrt{N/k}$. Thus, at fixed budget, we maximize $d(k)/\sqrt{k}$ (Fig. 6c). At $k = 1$, d is near zero or negative, so single steps provide little information about which policy is better; it then grows with k on every task. In contrast, $d(k)/\sqrt{k}$ peaks at intermediate lengths, $k \approx 75$–300, with our $k = 100$ near the peak.

Price on expert use. Without the price ($c_E = 0$), λ stays above 0 on 13 of 25 DMC tasks, even once the learner matches the expert, capping performance at the expert's level. Raising c_E withdraws the expert sooner: mean withdrawal falls from 17.4M steps at $c_E = 0$ to 0.6M at $c_E = 3$. However, too high a price withdraws it before it pays off, so relative return peaks at the default $c_E = 0.3$, the value we use on every task without per-task tuning (Fig. 6b and 12).

5 Conclusion

We presented a policy-gradient method in which a fixed expert and the learner share each episode in blocks, with the expert's share optimized by the same objective as the policy. Committing to blocks reduces the λ-gradient to the gap between what expert and learner blocks earn. A price on expert use turns withdrawal into part of the objective, and an imitation term on expert blocks lets every block train the learner. Our method reaches a given return in fewer environment steps than guided and unguided baselines, with a median per-task speed-up of $2.55\times$ over PPO on DMControl and the largest margins under sparse rewards, even with an expert that never succeeds. The handover needs no per-task tuning, and once λ reaches zero the deployed policy runs without the expert. Two directions follow. The comparison of block advantages extends to several experts, each with its own share. And since λ measures what the expert is worth, it could budget queries to expensive experts, such as the model-based controllers common in robotics.

AI USE STATEMENT

We used generative AI tools to polish the writing of the manuscript and as a coding assistant. The authors reviewed all AI-edited text and tested all AI-assisted code, and take full responsibility for the content of this work.

REPRODUCIBILITY STATEMENT

Sec. 3 and App. A give the derivations and proofs. Sec. 4 and App. C describe the setup, hyperparameters, and experts, and App. B.3 describes the FrozenLake task. We will release the code and all experts upon publication.

REFERENCES

Emre Adabag, Marcus Greiff, John Subosits, and Thomas Lew. Differentiable model predictive control on the GPU, 2025. Accepted at the International Conference on Learning Representations (ICLR) 2026.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Reincarnating reinforcement learning: Reusing prior computation to accelerate progress. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

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.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Mohak Bhardwaj, Sanjiban Choudhury, and Byron Boots. Blending MPC & value function approximation for efficient reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2021.

Jan Carius, Farbod Farshidian, and Marco Hutter. MPC-Net: A first principles guided policy search. *IEEE Robotics and Automation Letters*, 5(2):2897–2904, 2020. doi: 10.1109/LRA.2020.2974653.

Ching-An Cheng, Xinyan Yan, Nolan Wagener, and Byron Boots. Fast policy learning through imitation and reinforcement. *arXiv preprint arXiv:1805.10413*, 2018.

Ching-An Cheng, Andrey Kolobov, and Alekh Agarwal. Policy improvement via imitation of multiple oracles. *Advances in Neural Information Processing Systems*, 33:5587–5598, 2020.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. In *Advances in Neural Information Processing Systems 36, New Orleans, LA, USA*, December 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, 2024.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep Q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Se Hwan Jeon, Ho Jae Lee, Seungwoo Hong, and Sangbae Kim. Residual MPC: Blending reinforcement learning with gpu-parallelized model predictive control, 2025.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6023–6029, 2019.

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

Andrey Kurenkov, Ajay Mandlekar, Roberto Martin-Martin, Silvio Savarese, and Animesh Garg. Ac-teach: A bayesian actor-critic method for policy learning with an ensemble of suboptimal teachers. *arXiv preprint arXiv:1909.04121*, 2019.

Sergey Levine and Vladlen Koltun. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, volume 28 of *Proceedings of Machine Learning Research*, pp. 1–9, 2013.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

Xuefeng Liu, Takuma Yoneda, Rick L. Stevens, Matthew R. Walter, and Yuxin Chen. Blending imitation and reinforcement learning for robust policy improvement. In *International Conference on Learning Representations*, 2024.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299, 2018.

Eduardo Pignatelli, Jarek Liesen, Robert Lange, Chris Lu, Pablo Samuel Castro, and Laura Toni. Navix: Scaling minigrid environments with jax. *Advances in Neural Information Processing Systems*, 38, 2026.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems (RSS)*, 2018.

Ram Ramrakhya, Dhruv Batra, Erik Wijmans, and Abhishek Das. PIRLNav: Pretraining with imitation and RL finetuning for ObjectNav. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Krishan Rana, Vibhavari Dasagi, Ben Talbot, Michael Milford, and Niko Sünderhauf. Multiplicative controller fusion: Leveraging algorithmic priors for sample-efficient reinforcement learning and safe sim-to-real transfer. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

Krishan Rana, Vibhavari Dasagi, Jesse Haviland, Ben Talbot, Michael Milford, and Niko Sünderhauf. Bayesian controller fusion: Leveraging control priors in deep reinforcement learning for robotics. *The International Journal of Robotics Research*, 2023. doi: 10.1177/02783649231167210.

Rudolf Reiter, Jasper Hoffmann, Joschka Boedecker, and Moritz Diehl. A hierarchical approach for strategic motion planning in autonomous racing. In *European Control Conference (ECC)*, pp. 1–8, 2023. doi: 10.23919/ECC57647.2023.10178143.

Angel Romero, Elie Aljalbout, Yunlong Song, and Davide Scaramuzza. Actor-critic model predictive control: Differentiable optimization meets reinforcement learning for agile flight. *IEEE Transactions on Robotics*, 42:673–692, 2026. doi: 10.1109/TRO.2025.3644945.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635, 2011.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Çağlar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *International Conference on Learning Representations*, 2016.

Simon Schmitt, Jonathan J. Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M. Czarnecki, Joel Z. Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, and S. M. Ali Eslami. Kickstarting deep reinforcement learning, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Idan Shenfeld, Zhang-Wei Hong, Aviv Tamar, and Pulkit Agrawal. Tgrl: An algorithm for teacher guided reinforcement learning. In *International Conference on Machine Learning*, pp. 31077–31093. PMLR, 2023.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018a. doi: 10.1126/science.aar6404.

Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning, 2018b.

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

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 12, 2000.

Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7):1633–1685, 2009.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2020.100022.

Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. Jump-start reinforcement learning. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, 2023.

Linhai Xie, Sen Wang, Stefano Rosa, Andrew Markham, and Niki Trigoni. Learning with training wheels: speeding up training with a simple controller for deep reinforcement learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6276–6283. IEEE, 2018.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to Reason under Off-Policy Guidance. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3904.

Qisong Yang, Thiago D Simão, Nils Jansen, Simon H Tindemans, and Matthijs TJ Spaan. Reinforcement learning by guided safe exploration. In *ECAI 2023*, pp. 2858–2865. SAGE Publications 1 Oliver's Yard, 55 City Road, London, EC1Y 1SP, 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.

Haichao Zhang, Wei Xu, and Haonan Yu. Policy expansion for bridging offline-to-online reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2023.

Zhuangdi Zhu, Kaixiang Lin, Anil K. Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(11): 13344–13362, 2023.

A Proof of Proposition 1

Proof. The switch z_j is drawn independently of the boundary state, so (s_{b_j}, z_j) has a product law, and the expectation in equation 5 can be taken in three stages: over the boundary state s_{b_j} under the mixture, then over $z_j \sim \mathrm{Bernoulli}(\lambda)$, which the conditioning leaves unchanged, then over the execution of the block:

$$\mathbb{E}\Big[\frac{z_j-\lambda}{\lambda(1-\lambda)}\,\widehat{A}_j\Big] = \mathbb{E}_{s_{b_j}}\Big[\mathbb{E}_{z_j}\Big[\mathbb{E}\Big[\frac{z_j-\lambda}{\lambda(1-\lambda)}\,\widehat{A}_j \;\Big|\; s_{b_j}, z_j\Big]\Big]\Big]. \tag{14}$$

In the innermost expectation, the score depends only on z_j, so it moves outside:

$$\mathbb{E}\Big[\frac{z_j-\lambda}{\lambda(1-\lambda)}\widehat{A}_j \;\Big|\; s_{b_j}=s,\, z_j\Big] = \frac{z_j-\lambda}{\lambda(1-\lambda)}\,\mathbb{E}[\widehat{A}_j \mid s_{b_j}=s, z_j] = \begin{cases} \dfrac{a_E(s)}{\lambda}, & z_j=1, \\ \dfrac{-\,a_A(s)}{1-\lambda}, & z_j=0. \end{cases} \tag{15}$$

Averaging over z_j, each outcome's probability cancels the denominator of its own branch:

$$\lambda\cdot\frac{a_E(s)}{\lambda} + (1-\lambda)\cdot\frac{-a_A(s)}{1-\lambda} = a_E(s) - a_A(s). \tag{16}$$

Substituting into equation 5 for every block gives equation 10. □

B Additional Results

Task groups. Table 1 summarizes the per-task results of Tables 2 and 3. It splits the DMControl tasks into three families: *Low-dim. control* (11 tasks): AcrobotSwingup, AcrobotSwingupSparse, CartpoleBalance, CartpoleBalanceSparse, CartpoleSwingup, CartpoleSwingupSparse, PendulumSwingup, BallInCup, PointMass, ReacherEasy, ReacherHard; *Finger* (3 tasks): FingerSpin, FingerTurnEasy, FingerTurnHard; *Locomotion* (11 tasks): CheetahRun, FishSwim, HopperHop, HopperStand, SwimmerSwimmer6, WalkerRun, WalkerStand, WalkerWalk, HumanoidRun, HumanoidStand, HumanoidWalk. The MiniGrid column covers the 9 tasks of Table 3, and the All column covers all 34 tasks. For each column, we report the median steps to bar over tasks, counting a run that does not reach the bar within 20M steps as 20M. We also report the mean over tasks of the return divided by the best method's return on that task, using the asymptotic return on DMControl and the final return on MiniGrid.

B.1 DMControl

Table 2 reports the steps to bar and the asymptotic return on every DMControl task. Fig. 7 shows the learning curves, and Fig. 8 shows the expert share λ of our method over training.

B.2 MiniGrid

Table 3 reports the steps to bar and the final return on every MiniGrid task. Fig. 9 shows the learning curves, with the expert's return as a dotted line, and Fig. 10 shows the expert share λ.

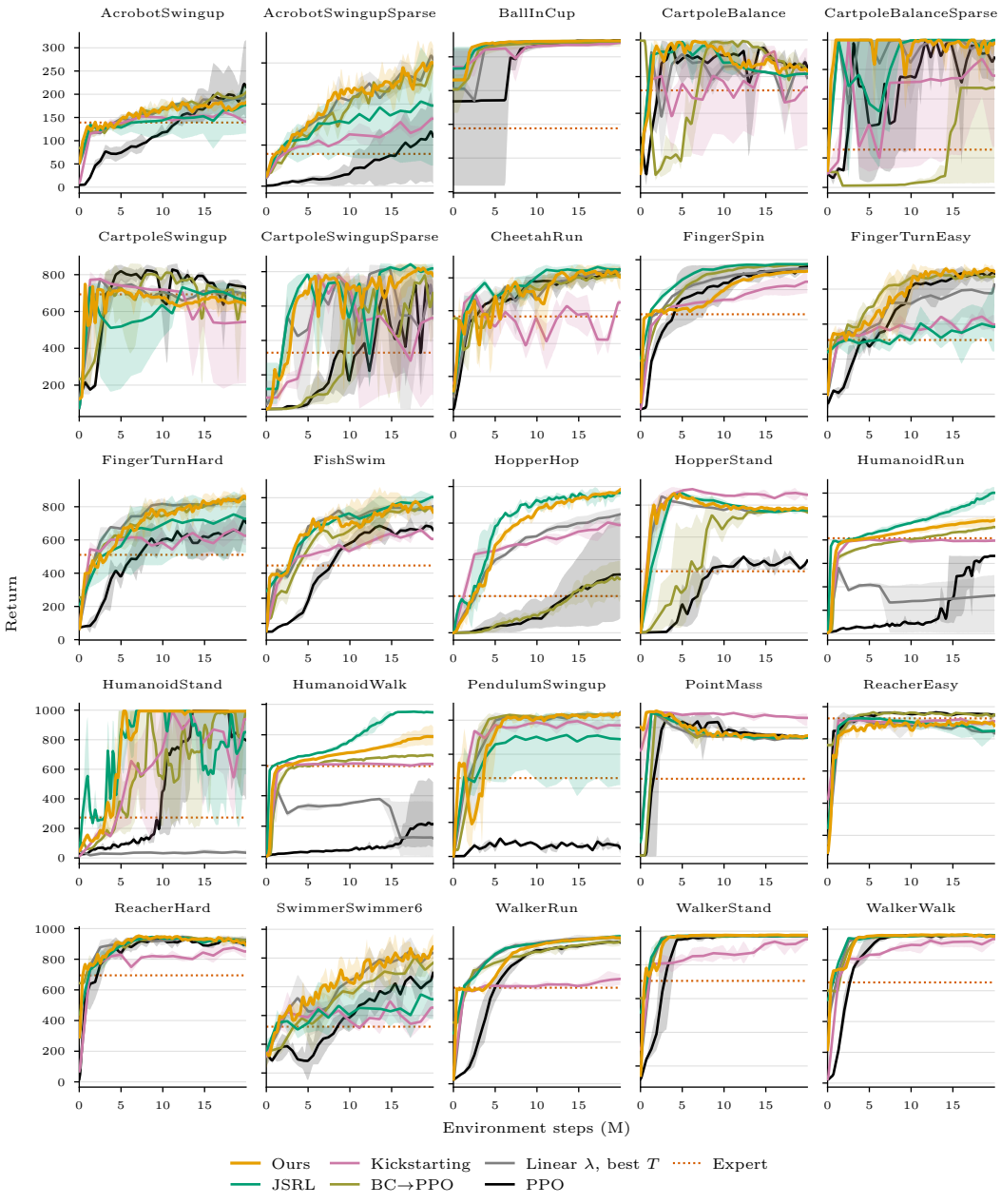


Figure 7: **DMControl learning curves per task.** Return of the learner acting alone with deterministic actions, for each method.

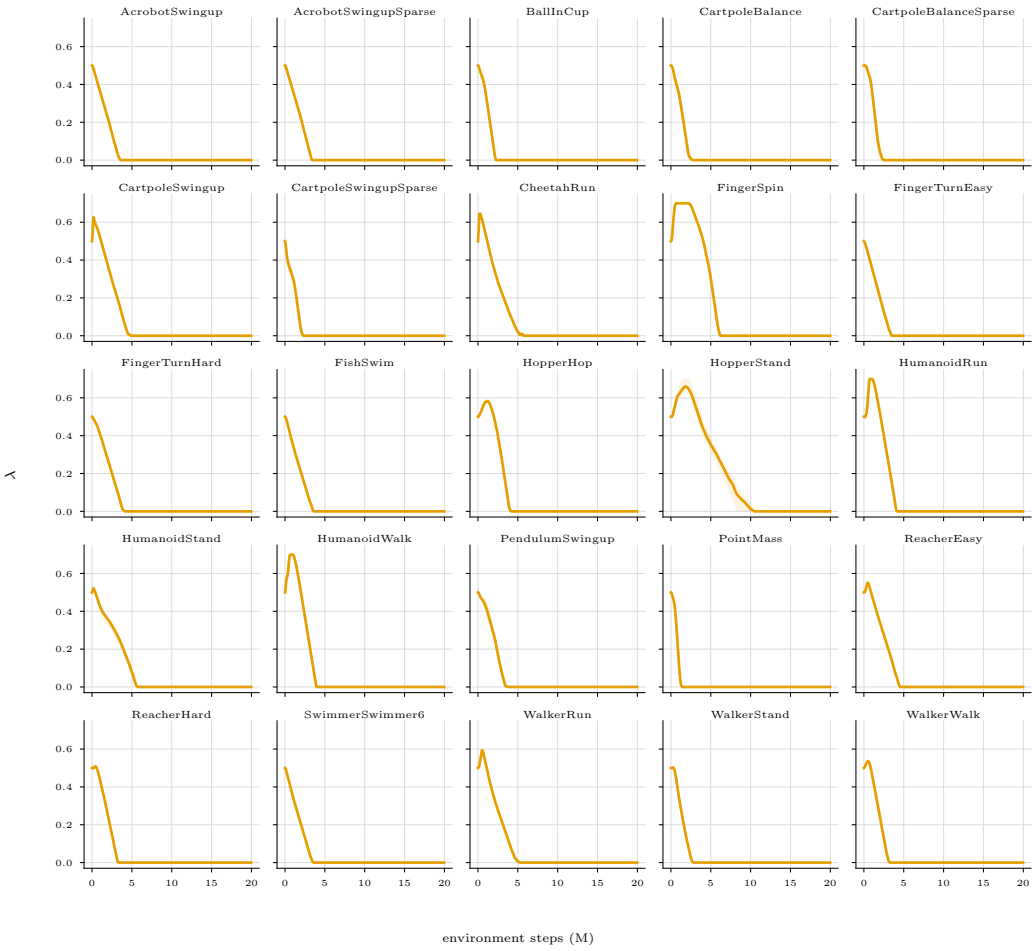


Figure 8: **DMControl expert share per task.** Expert share λ of our method over training.

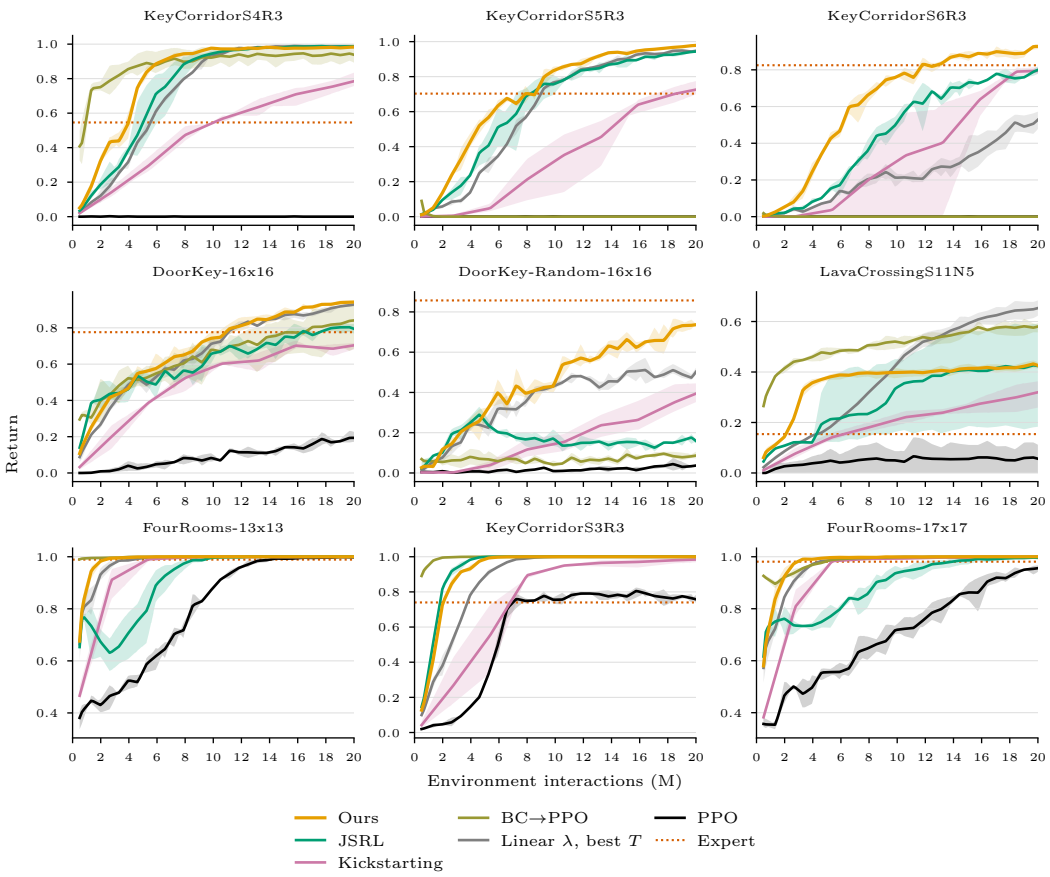


Figure 9: **MiniGrid learning curves per task.** Return of the learner acting alone with deterministic actions, for each method. The dotted line is the expert's return.

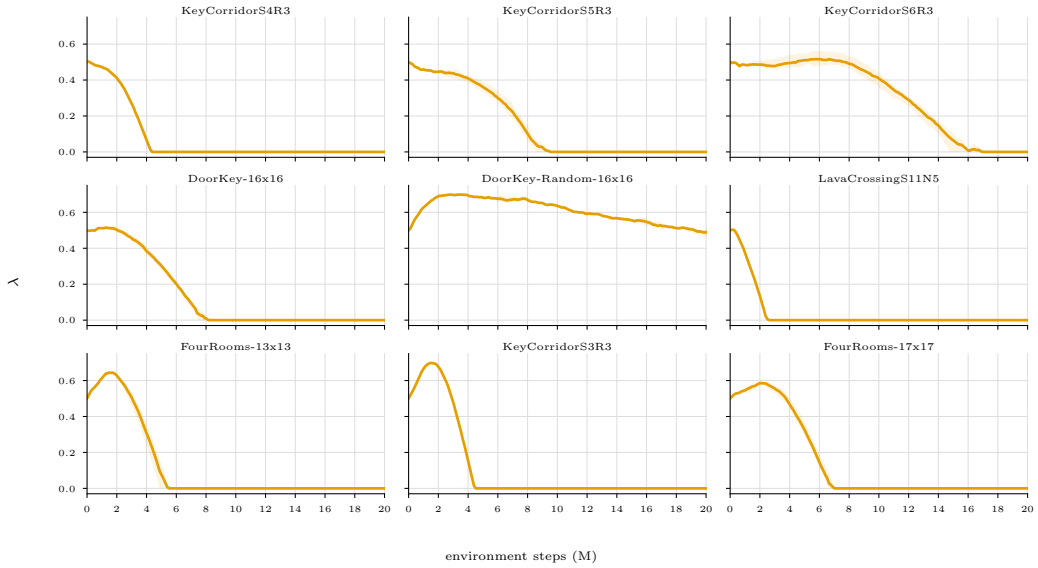


Figure 10: **MiniGrid expert share per task.** Expert share λ of our method over training.

Table 2: **Sample efficiency and final performance per task on DMC.** Each cell shows *steps to bar* (M) / *asymptotic return* (median over 5 seeds; "–" if not reached within 20M steps). In the median-steps row, runs that do not reach the bar count as 20M. Normalized return is the return divided by the best method's return on that task, averaged over tasks. **Bold**: best. Underline: second best. Ties are marked for every tied method and counted for each of them in the last row.

Task	bar	**Ours**	JSRL	Kick	BC→PPO	λ1M	λ2.5M	λ5M	PPO
AcrobotSwingup	145	**5.8** / 175	9.3 / 156	9.1 / 152	7.7 / 192	8.7 / 185	7.8 / **193**	8.7 / 189	14.8 / 189
AcrobotSwingupSparse	41	**8.6** / **54**	12.3 / 36	– / 27	12.4 / 50	14.0 / 52	14.0 / 50	11.9 / 49	– / 19
BallInCup	730	**1.9** / **974**	3.3 / 973	4.2 / 968	2.2 / 971	2.9 / 973	3.3 / **974**	4.6 / 973	6.6 / **974**
CartpoleBalance	665	1.0 / 854	1.3 / 822	1.3 / 834	**0.1** / 884	1.3 / 817	1.3 / 814	1.7 / 821	2.5 / **886**
CartpoleBalanceSparse	750	**1.1** / 987	3.3 / **1000**	8.7 / 743	16.5 / 642	4.2 / 996	5.0 / 998	5.0 / 933	6.0 / 944
CartpoleSwingup	566	**1.5** / 661	6.6 / 704	2.5 / 572	5.7 / 719	3.3 / 697	4.6 / 690	2.5 / 693	4.2 / **754**
CartpoleSwingupSparse	611	**4.1** / 783	5.8 / **815**	9.1 / 432	13.7 / 635	7.4 / 717	7.4 / 729	7.0 / 788	15.0 / 566
CheetahRun	636	5.4 / 829	**3.7** / **848**	7.0 / 576	6.1 / 801	7.0 / 815	5.8 / 826	9.5 / 823	5.2 / 816
FingerSpin	727	9.3 / 914	**5.4** / **969**	11.9 / 808	5.7 / 962	9.1 / 792	6.2 / 926	8.7 / 931	8.0 / 923
FingerTurnEasy	672	7.7 / **896**	18.5 / 570	– / 600	**5.7** / 874	8.7 / 777	11.9 / 787	16.0 / 773	10.1 / 880
FingerTurnHard	625	**5.7** / **833**	7.4 / 730	14.8 / 606	7.1 / 831	6.2 / 824	7.4 / 827	10.3 / 815	11.1 / 635
FishSwim	429	**4.6** / 567	7.4 / **572**	16.0 / 444	7.9 / 552	7.4 / 558	7.4 / 550	10.3 / 544	12.6 / 471
HopperHop	283	7.2 / **378**	**5.6** / 374	18.5 / 276	– / 137	14.8 / 262	17.8 / 288	15.2 / 304	19.7 / 134
HopperStand	647	**2.0** / 778	5.0 / 768	3.3 / **863**	9.2 / 763	5.8 / 745	2.9 / 766	5.8 / 770	– / 436
HumanoidRun	210	7.7 / 238	**5.4** / **280**	– / 198	14.1 / 221	– / 76	– / 7	– / 62	– / 144
HumanoidStand	745	**5.5** / **993**	5.7 / 754	11.1 / 892	10.6 / 951	– / 36	– / 35	– / 35	11.9 / 936
HumanoidWalk	712	14.4 / 768	**9.1** / **950**	– / 606	– / 663	– / 213	– / 18	– / 238	– / 162
PendulumSwingup	629	**4.8** / 828	6.2 / 698	5.0 / 774	5.9 / 832	7.0 / **839**	5.8 / 829	7.4 / **839**	– / 63
PointMass	655	**0.1** / 748	2.5 / 743	0.9 / **873**	3.0 / 740	2.5 / 733	2.9 / 739	5.8 / 741	3.3 / 751
ReacherEasy	716	1.7 / 899	2.5 / 881	2.5 / 910	**0.6** / **955**	2.5 / 864	2.5 / 872	2.5 / 865	1.3 / **955**
ReacherHard	704	**1.3** / 926	3.3 / 922	2.5 / 836	1.6 / 932	2.5 / 930	4.2 / 925	4.6 / **938**	3.3 / 909
SwimmerSwimmer6	379	**6.9** / **505**	8.7 / 326	– / 292	11.6 / 434	10.7 / 491	11.9 / 447	16.0 / 415	16.0 / 403
WalkerRun	524	6.0 / **698**	5.8 / 691	– / 480	**4.0** / 674	5.0 / 696	7.0 / 679	9.1 / 667	7.2 / 672
WalkerStand	732	**1.7** / **977**	2.5 / 973	2.5 / 923	2.2 / 973	2.5 / 973	5.4 / 974	7.0 / 973	4.6 / 973
WalkerWalk	721	**1.5** / 961	2.5 / 959	4.2 / 916	3.0 / 957	2.9 / 959	6.2 / 957	7.8 / 957	4.4 / **962**
Median steps / norm. return		**4.8** / **0.957**	5.6 / 0.908	9.1 / 0.794	6.1 / 0.901	7.0 / 0.841	6.2 / 0.829	8.7 / 0.843	10.1 / 0.784
# fastest / # best return		**16** / **9**	5 / 7	0 / 2	4 / 1	0 / 1	0 / 2	0 / 2	0 / 5

Table 3: **Sample efficiency and final performance per task on MiniGrid.** Each cell shows *steps to bar* / *final return*. Steps to bar is the number of environment steps (in millions) needed to reach 75% of the best final return on that task (median over 5 seeds; "–" if not reached within 20M steps). Final return is the deterministic return after 20M steps (mean over 5 seeds). In the median-steps row, runs that do not reach the bar count as 20M. Normalized return is the final return divided by the best method's final return on that task, averaged over tasks. **Bold**: best. Underline: second best. Ties are marked for every tied method and counted for each of them in the last row.

Task	bar	**Ours**	JSRL	Kick	BC→PPO	λ1M	λ2.5M	λ5M	PPO
KeyCorridorS4R3	0.75	4.7 / 0.98	5.9 / 0.99	16.7 / 0.75	**2.2** / 0.94	4.2 / 0.92	4.9 / **1.00**	7.4 / 0.99	– / 0.00
KeyCorridorS5R3	0.72	8.5 / **0.96**	**7.9** / 0.92	17.9 / 0.69	– / 0.00	9.1 / 0.77	9.1 / 0.94	12.4 / 0.90	– / 0.00
KeyCorridorS6R3	0.67	**7.8** / **0.90**	12.3 / 0.77	15.7 / 0.75	– / 0.00	– / 0.00	– / 0.46	14.8 / 0.77	– / 0.00
DoorKey-16x16	0.70	**8.5** / **0.93**	10.5 / 0.79	16.7 / 0.69	8.6 / 0.81	9.0 / 0.90	9.6 / 0.90	12.5 / 0.77	– / 0.17
DoorKey-Random-16x16	0.52	**10.3** / **0.69**	– / 0.15	– / 0.34	– / 0.09	– / 0.21	15.8 / 0.49	14.5 / 0.61	– / 0.03
LavaCrossingS11N5	0.48	– / 0.42	12.5 / 0.42	– / 0.30	**4.3** / 0.58	8.7 / 0.56	10.9 / **0.64**	18.9 / 0.45	– / 0.06
FourRooms-13x13	0.75	0.7 / **1.00**	0.7 / **1.00**	1.9 / **1.00**	**0.5** / **1.00**	0.7 / **1.00**	0.7 / **1.00**	0.7 / **1.00**	8.2 / **1.00**
KeyCorridorS3R3	0.75	2.1 / **1.00**	1.9 / **1.00**	6.7 / 0.98	**0.5** / **1.00**	3.0 / **1.00**	3.8 / **1.00**	4.6 / **1.00**	6.9 / 0.77
FourRooms-17x17	0.75	1.0 / **1.00**	1.2 / 0.99	2.6 / **1.00**	**0.5** / **1.00**	1.5 / **1.00**	1.6 / **1.00**	1.0 / **1.00**	11.9 / 0.93
Median steps / norm. return		7.8 / **0.960**	7.9 / 0.835	16.7 / 0.775	**4.3** / 0.649	8.7 / 0.771	9.1 / 0.906	12.4 / 0.907	20.0 / 0.336
# fastest / # best return		3 / **7**	1 / 2	0 / 2	**5** / 3	0 / 3	0 / 5	0 / 3	0 / 1

B.3 FROZENLAKE

Task. A point mass moves on a continuous 36×36 grid in which 32.9% of the cells are holes. The action is a two-dimensional acceleration. Entering a hole ends the episode. The reward is $+1$ on the step that enters the goal and 0 otherwise. With probability p_{slip}, the commanded acceleration is rotated by a quarter turn. Five full-width walls with alternating doorways force a serpentine route of 123 cells with 25 turns. Episodes last at most 700 steps.

Expert. Breadth-first search over the free cells returns a path, and a PD controller tracks it. The expert is suboptimal by construction: the shortest path passes close to the corners of holes, and the planner has no model of the slip. It never reaches the goal (Fig. 5).

Protocol. All methods share the environment, the expert, the network, the PPO hyperparameters, and the budget of 2×10^7 transitions. They differ only in how they use the expert. We report the success rate of the learner acting alone with deterministic actions.

Result. Table 4 and Fig. 5 give the results at $p_{\text{slip}} = 0.3$. PPO never reaches the goal, so its advantages are zero for the whole run and the reward gives it no learning signal. BC→PPO follows a route that ends in a hole. JSRL and Kickstarting get further along the route than the expert, but they usually fall into a hole before the goal. Our learner reaches the goal in 98.5% of the episodes.

Table 4: **Final success rate on FrozenLake** ($p_{\text{slip}} = 0.3$), mean over 5 seeds.

Method	Success rate
PPO	0.000
BC→PPO	0.000
Kickstarting	0.037
JSRL	0.276
Ours	**0.985**

B.4 Ablations per task

Fig. 11 and Fig. 12 show per-task runs for the imitation-weight and expert-price ablations of Sec. 4.5 (Figs. 6a and 6b).

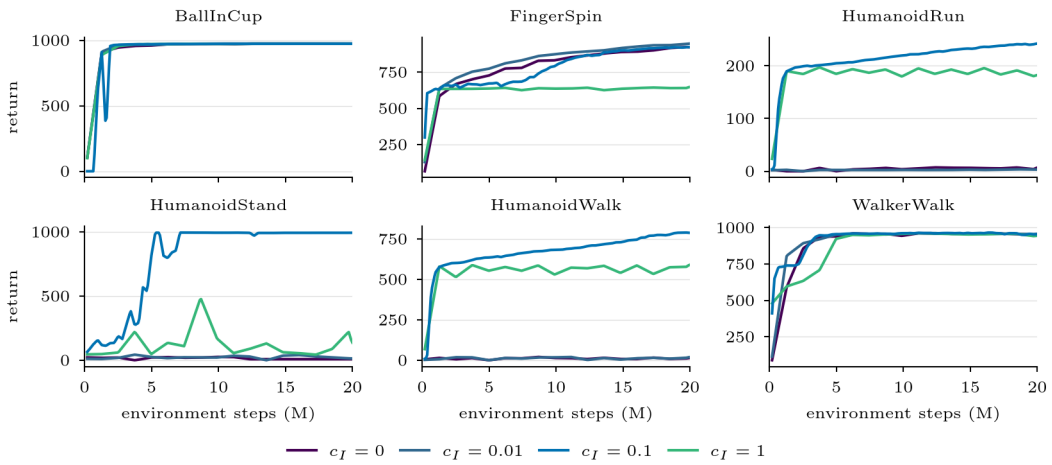


Figure 11: **Imitation weight per task.** Return over training on six DMControl tasks for each c_I. With $c_I \in \{0, 0.01\}$, the learner stays near zero return on the three Humanoid tasks. With $c_I = 1$, it stays near the expert's return on FingerSpin and ends lower on the Humanoid tasks. On BallInCup and WalkerWalk, c_I has little effect.

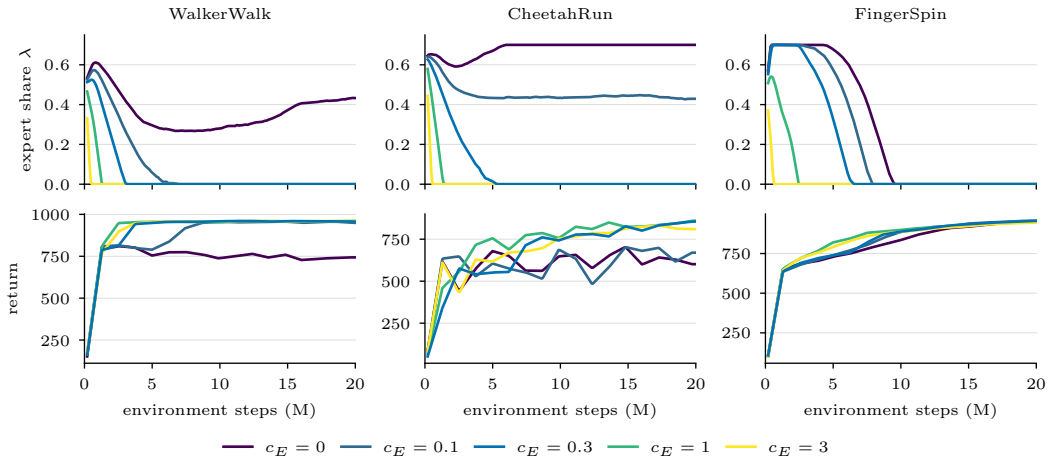


Figure 12: **Expert price per task.** Expert share λ (top) and return (bottom) over training on three DMControl tasks for each c_E. With $c_E = 0$, λ stays above zero until the end of training on WalkerWalk and CheetahRun, and reaches zero on FingerSpin after about 10M steps. With $c_E = 3$, λ reaches zero within 1M steps on all three tasks.

C Hyperparameters

Table 5 reports the hyperparameters used for DMControl, while Table 6 reports only the differences with respect to DMControl. All baselines use the same network architecture, with two hidden layers of 256 units for both the actor and critic, and $\tanh$ activations.

Table 5: **Hyperparameters.** The PPO backbone is identical for all methods; only the guidance mechanism differs. All methods use the same frozen guide π_E with no per-task tuning.

Parameter	Value
Shared PPO backbone	
N parallel environments	2048
Rollout length T	30
Minibatches / epochs	32 / 16
γ / λ_{GAE}	0.995 / 0.95
Clip ϵ	0.2
Value / entropy coefficient	0.5 / 0.01
Max gradient norm	0.5
Activation	tanh
Observation normalisation	yes
Environment steps	2×10^7
Evaluation	128×1000, deterministic
Ours	
Block length k	100
λ_0 / $\lambda_{\max}$	0.5 / 0.7
η_λ / c_E / c_I	0.03 / 0.3 / 0.1
JSRL	
Tolerance τ / stages	0.2 / 10
Guide horizon / MA window	1000 / 0.3
BC coefficient	0.1
BC→PPO	
Transitions / epochs	5×10^5 / 60
Learning rate / batch size	3×10^{-4} / 4096
Kickstarting	
Distillation coefficient	1.0
Anneal fraction / teacher samples	0.5 / 8
PBT population	4
Linear λ	
Schedule	$1 \to 0$
Decay horizon	1M (also 2.5M, 5M)
Imitation weight c_I	0.1

Table 6: **MiniGrid hyperparameters.** Only differences from the DMControl settings in Table 5 are reported.

Parameter	Value
Shared PPO backbone	
N parallel environments	512
Rollout length T	64
Minibatches / epochs	8 / 2
Learning rate	10^{-3}
Observation normalisation	no
Ours	
Block length k	30
Imitation weight c_I	1.0
JSRL	
Tolerance τ	0.05
BC coefficient	1.0
Linear λ	
Imitation weight c_I	1.0

D GUIDES

Tables 7 and 8 provide the reward achieved by each guide.

Table 7: Guide quality on DMControl. The ceiling is the best asymptotic return of any method on that task (Table 2). Guides are deliberately sub-optimal (mean $0.60\times$ the ceiling, range 0.26–0.97).

Task	Guide π_E	ceiling	π_E/ceil
AcrobotSwingup	139.1	193	0.72
AcrobotSwingupSparse	15.7	54	0.29
BallInCup	844.0	974	0.87
CartpoleBalance	724.9	886	0.82
CartpoleBalanceSparse	256.2	1000	0.26
CartpoleSwingup	693.1	754	0.92
CartpoleSwingupSparse	328.9	815	0.40
CheetahRun	565.0	848	0.67
FingerSpin	635.2	969	0.66
FingerTurnEasy	509.1	896	0.57
FingerTurnHard	510.5	833	0.61
FishSwim	330.3	572	0.58
HopperHop	100.4	378	0.27
HopperStand	386.1	863	0.45
HumanoidRun	203.7	280	0.73
HumanoidStand	273.4	993	0.28
HumanoidWalk	594.8	950	0.63
PendulumSwingup	460.0	839	0.55
PointMass	482.0	873	0.55
ReacherEasy	927.4	955	0.97
ReacherHard	694.8	938	0.74
SwimmerSwimmer6	261.1	505	0.52
WalkerRun	461.0	698	0.66
WalkerStand	705.1	977	0.72
WalkerWalk	654.1	962	0.68
mean			0.60

Table 8: Guide quality on MiniGrid. Guides are deliberately sub-optimal (mean $0.73\times$ the attainable performance).

Task	Guide π_E	ceiling	π_E/ceil
KeyCorridorS4R3	0.546	1.000	0.55
KeyCorridorS5R3	0.703	1.000	0.70
KeyCorridorS6R3	0.825	1.000	0.83
DoorKey-16x16	0.776	1.000	0.78
DoorKey-Random-16x16	0.857	1.000	0.86
LavaCrossingS11N5	0.154	1.000	0.15
FourRooms-13x13	0.989	1.000	0.99
KeyCorridorS3R3	0.740	1.000	0.74
FourRooms-17x17	0.981	1.000	0.98
mean			0.73