CAPTURING INTER-TASK REINFORCEMENT LEARNING DYNAMICS FOR ADAPTIVE TASK-SAMPLING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-Task Reinforcement Learning (MTRL) trains a single agent on multiple tasks simultaneously, potentially improving performance and sample efficiency through positive transfer. However, MTRL suffers from *imbalanced learning*: tasks progress at different rates and may interfere with one another. Prior work addresses this problem using specialized task-sampling strategies that prescribe, *a priori*, how experience should be allocated across tasks. We instead argue that task-sampling strategies should be learned directly from data. For that end, we introduce DynATS, a learned sampler that captures inter-task learning **D**ynamics for **A**daptive **T**ask **S**ampling. DynATS is trained across sets of tasks using Evolution Strategies to maximize the performance of the multi-task agent as a whole. On MTRL problems, DynATS outperforms existing sampling strategies on average. By construction, DynATS transfers seamlessly across task distributions and domains. We transfer a learned sampler trained only on discrete, grid-based task sets to unseen task sets from the same distribution and to unseen continuous-control tasks in MuJoCo and Meta-World, demonstrating that the learned strategies can generalize beyond the environments on which they were trained.

1 INTRODUCTION

Multi-Task Reinforcement Learning (MTRL) extends Reinforcement Learning (RL) (Sutton & Barto, 2018) to learning several tasks jointly with a shared agent, enabling transfer of experience and representations and, when transfer is beneficial, improving both performance and sample-efficiency (Brunskill & Li, 2013; D'Eramo et al., 2024). However, the shared agent can also suffer from negative interference: updates that benefit one task may harm others. Most prior work addresses this through specialized architectures (Yang et al., 2020; Sodhani et al., 2021; Sun et al., 2022; Hendawy et al., 2024) or optimization methods that mitigate conflicting gradients and balance task losses (Chen et al., 2018; Yu et al., 2020; Liu et al., 2021; Navon et al., 2022; Liu et al., 2023).

An alternative approach is to focus on *which tasks generate the experience used for learning*. Since all tasks update a shared policy, collecting data from one task can affect performance on all others. Uniform sampling allocates the same budget to every task, even though their usefulness may differ and change throughout training. Existing adaptive sampling methods outperform uniform sampling by prioritizing, for example, tasks with higher TD-error of Bellman updates (D'Eramo & Chalvatzaki, 2022), harder tasks (Cho et al., 2024) or tasks furthest from a target performance (Corrado et al., 2026). However, these methods specify a priori what makes a task worth sampling. This can be restrictive for two reasons: (i.) the most useful strategy may change throughout a single training run, and (ii.) different task sets may benefit from different, potentially even opposing, strategies. This motivates learning the sampling strategy itself from data rather than choosing it a priori.

In this paper, we treat task-sampling as a dynamic decision problem whose objective is to improve the multi-task agent as a whole. The main challenge is that the value of sampling a task is not directly known: it depends on how the resulting experience affects the shared policy, and this effect may only become apparent after many policy updates. Moreover, the relevant learning dynamics can change throughout training and across task sets. Therefore, we propose **DynATS**, a learned sampler that captures inter-task learning **Dyn**amics for **A**daptive **T**ask **S**ampling. Due to the long horizon trajectories, we use Evolution Strategies (ES) (Salimans et al., 2017) to learn the sampler, evaluating each candidate through a complete MTRL training run and using the resulting agent performance

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.

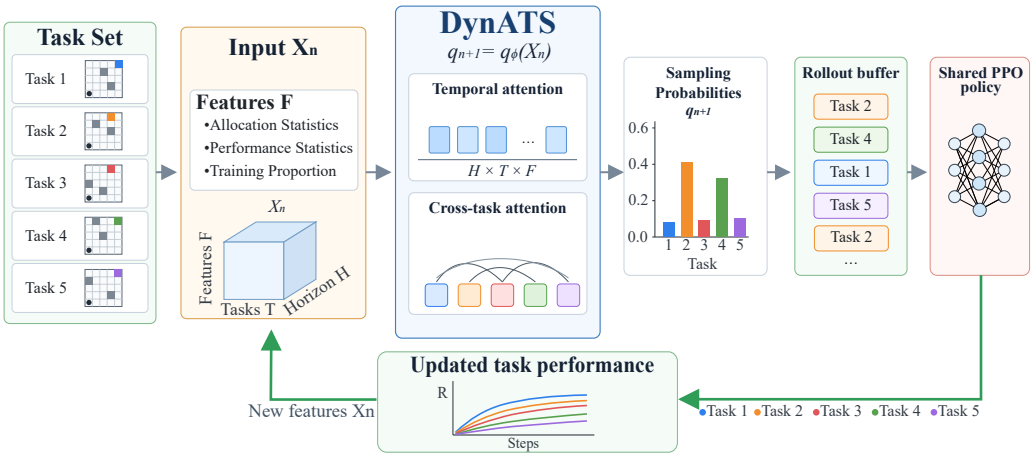


Figure 1: Overview of DynATS. At PPO epoch n, tasks are sampled according to the current distribution q_n until the rollout buffer is full, after which the shared policy is updated. The resulting task-level learning state X_n is computed from generic statistics collected during training. DynATS processes the temporal history of these statistics and exchanges information across tasks through cross-task attention to produce the sampling distribution q_{n+1} for the next PPO epoch. The procedure repeats until PPO training is complete.

as fitness. ES avoids differentiating through the RL optimization process, scales naturally through parallel evaluation, and has been used for meta-optimization in meta-RL (Jackson et al., 2024) and learned optimization for RL (Goldie et al., 2024).

DynATS should receive per-task learning signals, adapt its allocation throughout training, and transfer across task sets without depending on task identities or ordering. It should also exploit both the recent history of each task and the relationships between tasks. We use a Transformer (Vaswani et al., 2017) to satisfy these requirements: shared parameters and attention across tasks yield a permutation-equivariant mapping from task features to sampling probabilities, while temporal context allows sampling decisions to depend on the evolution of each task's learning signals. An overview of DynATS can be seen in Figure 1.

Our main *contributions* are: 1) we formulate task-sampling in MTRL as a learned adaptive allocation problem, where sampling decisions are optimized for their effect on learning across the entire task set, and introduce DynATS, a learned task-sampler; 2) we show that DynATS can learn effective sampling strategies on a fixed MTRL task set and outperform existing adaptive sampling methods; and 3) we show that DynATS can train a task-sampler across distributions of task sets and subsequently transfer zero-shot to held-out task sets, to sets containing more tasks than those seen during training, and even across domains. In particular, we train DynATS on discrete grid-based task sets and deploy it without finetuning on continuous-control benchmarks including MuJoCo6, Meta-World10, and Meta-World50.

2 Related Work

Multi-Task Interference and Task Weighting Sharing parameters across tasks enables transfer but can also cause negative interference. Prior MTRL work addresses this through architectures that separate, route, or compose shared and task-specific representations (Yang et al., 2020; Sodhani et al., 2021; Sun et al., 2022; Hendawy et al., 2024), and through optimization methods that modify conflicting gradients, balance task losses, or optimize multi-task scalarizations (Yu et al., 2020; Liu et al., 2021; Navon et al., 2022; Liu et al., 2023; Royer et al., 2023). Related questions have been studied extensively in supervised multi-task learning. Dynamic Task Prioritization adjusts task weights according to evolving task difficulty (Guo et al., 2018), while work on task relationships studies which tasks benefit from being learned together (Standley et al., 2020). Auto-λ goes further by meta-learning task weights that vary throughout training (Liu et al., 2022). These methods primarily control how available task data contributes to optimization. Instead, we control which tasks generate the experience used for learning.

Curriculum Learning and Task-Selection Outside MTRL Task selection has a long history in curriculum learning (CL) (Bengio et al., 2009; Matiisen et al., 2017; Portelas et al., 2020), e.g. Graves et al. (2017) formulate curriculum selection as a non-stationary bandit problem. However, CL methods typically aim at learning a target complex task via positive transfer from focusing on easier sub-tasks first. We instead focus on MTRL, where tasks may also exhibit negative interference, such that an easy-tasks-first approach may not be suitable, as shown by Cho et al. (2024). Task sampling has also been researched outside MTRL, e.g. Yao et al. (2021) train a neural scheduler to select meta-training tasks. A closely related problem has recently emerged in language-model training, where a fixed budget must be allocated across different data domains. DoReMi learns domain mixture weights using a proxy model (Xie et al., 2023), Skill-It performs online sampling using dependencies between skills and their learning state (Chen et al., 2023), and PiKE dynamically adjusts data mixtures using inter-task gradient interactions (Li et al., 2025). However, these methods mix examples from existing datasets. We instead tackle task-sampling in on-policy RL, which affects both the training data and the policy that generates future data.

Task-Sampling in Multi-Task Reinforcement Learning Several methods adapt task allocation directly in MTRL. CAMRL constructs a curriculum using asymmetric transfer between tasks (Huang et al., 2023), while PRISM uses prioritized sampling with intrinsic motivation (D'Eramo & Chalvatzaki, 2022). Hard Tasks First prioritizes tasks according to estimated difficulty (Cho et al., 2024); STARS combines task-aware prioritized sampling with shared–unique representations to focus learning on poorly performing tasks (Lin et al., 2024); more recently, DRATS prioritizes tasks according to their normalized gap from a target performance (Corrado et al., 2026). These approaches demonstrate the importance of non-uniform allocation, but prescribe a priori a sampling strategy. Our approach instead learns the mapping from generic training signals to sampling probabilities by optimizing their long-horizon effect on aggregate MTRL performance, allowing task priorities to vary with both the task set and the phase of training.

Learned Scheduling and Meta-Optimization Because sampling decisions affect subsequent policy updates, we optimize the scheduler through complete training trajectories. For this end, we use Evolution Strategies (Salimans et al., 2017), following their use for meta-optimization of RL algorithms and optimizers (Jackson et al., 2024; Goldie et al., 2024). However, our objective is different: we learn a sampling policy instead of an update rule for the RL agent itself.

3 Preliminaries

3.1 Multi-Task Reinforcement Learning

We model each individual task via a finite-horizon Markov Decision Process (MDP) (Sutton & Barto, 2018), defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \gamma)$. Here, $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces; $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ represents the transition probability distribution; $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function; $\rho_0 \in \Delta(\mathcal{S})$ is the initial state distribution; and $\gamma \in [0, 1)$ the discount factor.

In the MTRL paradigm, we consider a set of T distinct tasks indexed by $\mathcal{T} = \{1, \dots, T\}$. We assume a shared state space $\mathcal{S}$ across all tasks, while allowing the action space $\mathcal{A}$, transition dynamics $\mathcal{P}_i$, reward functions $\mathcal{R}_i$, and initial state distributions $\rho_{0,i}$ to vary for each task $i \in \mathcal{T}$.

Our objective is to optimize a task-conditioned policy $\pi_\theta(a|s, i)$, parameterized by θ. Let $J_i(\theta)$ denote the expected discounted return achieved by π_θ on task i:

$$J_i(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{H} \gamma^t \mathcal{R}_i(s_t, a_t) \right], \tag{1}$$

where $\tau = (s_0, a_0, s_1, a_1, \dots)$ represents a trajectory generated under policy π_θ in task i, and H denotes the episodic horizon. The MTRL optimization objective seeks the θ that maximizes the average performance across the entire task set:

$$\max_\theta J(\theta) = \max_\theta \frac{1}{T} \sum_{i=1}^{T} J_i(\theta). \tag{2}$$

3.2 Evolution Strategies

ES are a class of stochastic, black-box, and gradient-free optimization methods (Hansen et al., 2015). In our setting, ES optimizes the sampler's parameters ϕ. At each generation, we construct a population of candidate samplers by perturbing the current parameters with Gaussian noise,

$$\phi_k = \phi + \sigma\epsilon_k, \qquad \epsilon_k \sim \mathcal{N}(0, I),$$

where σ controls the perturbation scale. Each candidate is evaluated according to a scalar fitness F, and these evaluations determine the update of the population mean ϕ.

We use OpenAI ES (Salimans et al., 2017), following its use for meta-optimization in prior work (Jackson et al., 2024; Goldie et al., 2024). Rather than differentiating through the evaluation procedure, OpenAI ES estimates the gradient of the Gaussian-smoothed fitness objective as

$$\nabla_\phi \mathbb{E}\epsilon \sim \mathcal{N}(0, I)\left[F(\phi + \sigma\epsilon)\right] = \frac{1}{\sigma}\mathbb{E}\epsilon \sim \mathcal{N}(0, I)\left[F(\phi + \sigma\epsilon)\epsilon\right]. \tag{3}$$

The expectation is approximated using the finite ES population, and we use Adam (Kingma & Ba, 2017) to update ϕ from the resulting estimate.

To optimize the sampler over a distribution of task sets, we adopt antithetic task-sampling (Jackson et al., 2024). For each perturbation ϵ_k, the paired candidates $\phi + \sigma\epsilon_k$ and $\phi - \sigma\epsilon_k$ are evaluated on the same task set: the winner is assigned a fitness of $+0.5$ and the loser -0.5.

4 DynATS: Learning a Multi-Task Sampler

In MTRL, task-sampling determines not only how frequently each task is trained but also, since parameters are shared, how experience collected from one task affects the performance on the entire task set. We therefore view task-sampling as a sequential allocation problem: *at each stage of training, the goal is to allocate the interaction budget to the tasks whose data is most useful for improving the multi-task agent as a whole*.

4.1 Problem Formulation and Theoretical Motivation

Problem Formulation We consider multi-task PPO (Schulman et al., 2017) with a shared task-conditioned policy. At epoch n, each task i is represented by a feature vector $x_{n,i} \in \mathbb{R}^d$, and we denote the joint training state by $X_n = \{x_{n,i}\}_{i=1}^T$. A sampler q_ϕ, parameterized by ϕ, maps this state to a distribution over tasks,

$$q_n = q_\phi(X_n) \in \Delta_T. \tag{4}$$

Tasks are sampled from q_n until the rollout buffer is full, after which the shared policy is updated using standard PPO. Thus, q_ϕ determines where the interaction budget is spent and, consequently, the distribution of experience used to update the shared policy.

Theoretical Motivation Existing adaptive sampling strategies encode a fixed interpretation of a signal, e.g., that tasks with lower performance should be sampled more. However, the appropriate interpretation of the same signal can depend on the task set and its learning dynamics. We illustrate this limitation with a simple local allocation model.

Let $\mu_n(i)$ denote the expected local utility of allocating training to task i at round n. For two tasks $A \in \{-1, +1\}$, an observed signal $z_n \in \{-1, +1\}$, and

$$\mu_n^\xi(A) = \frac{1}{2} + \eta\xi z_n A, \qquad \xi \in \{-1, +1\}, \quad 0 < \eta < \frac{1}{2}, \tag{5}$$

the optimal allocation follows z_n when $\xi = +1$ and reverses it when $\xi = -1$.

Proposition 1. *For any fixed, possibly stochastic, rule $q(\cdot \mid z)$, there exists $\xi \in \{-1, +1\}$ for which its regret relative to the best instantaneous allocation satisfies*

$$R_N^\xi(q) \geq \eta N. \tag{6}$$

Group	Feature	Description
Allocation	Relative Allocation	Sampling probability relative to uniform
	Relative Cumulative Allocation	Cumulative allocation relative to uniform
Performance	Performance	Task performance normalized to a common scale
	Performance Change	Change in normalized performance
	Relative Performance	Performance relative to the other tasks
	Relative Performance Change	Performance change relative to the other tasks
Training	Training Proportion	Fraction of the training budget used

Table 1: Task-level features provided to DynATS.

Proof sketch. Let p_n be the probability that the rule selects $A = z_n$. Its instantaneous regret is $2\eta(1 - p_n)$ when $\xi = +1$ and $2\eta p_n$ when $\xi = -1$. The two regrets therefore sum to 2η at every round, so over N rounds at least one setting incurs regret at least ηN.

The proposition is not a guarantee for our learned sampler, it simply shows that no fixed interpretation of a signal is universally good. This motivates learning sampling strategies from data by capturing the inter-task dynamics of a given task set. A complete proof and discussion of the abstraction are given in Appendix A.

4.2 Implementation

Input Features We use per-task statistics freely available during MTRL training, see Table 1 for a summary and Appendix B.1 for their definitions. They capture three aspects of the learning state: how experience has been allocated, how each task is performing, and the current phase of training. We include four performance features, two capture per-task performance metrics while the remaining two capture the same metrics but relative to the other tasks. These features include the main signals exploited by existing adaptive sampling methods, while remaining agnostic to task identity, domain and RL algorithm. An ablation over these inputs can be seen in Section 5.6.

Performance features are computed from the rollouts collected for the PPO update, requiring no additional evaluation episodes. When a task is not sampled during a PPO epoch, its latest performance estimate is carried over. To ensure DyNATS can transfer to unseen task sets, these performance features must be normalized. In many MTRL benchmarks, natural reference values are available. For example, Meta-World's (Yu et al., 2021) success rates are bounded between 0 and 1. Fixed reference values are not required at deployment: we show that DynATS trained using reference values remains competitive when its inputs are instead normalized using running statistics at deployment, see Section 5.5. Moreover, existing adaptive sampling approaches such as DRATS (Corrado et al., 2026) and Hard Tasks First (Cho et al., 2024) also rely on references.

Sampler architecture DynATS uses a Transformer architecture (Vaswani et al., 2017). It processes a temporal context of H PPO epochs for each task and uses attention across tasks to produce one logit $u_{n,i}$ per task. Parameters are shared across tasks and the mapping is permutation equivariant, allowing the same sampler to operate on different task sets. The logits define the task distribution

$$\tilde{q}_{n,i} = \frac{\exp(u_{n,i})}{\sum_{j=1}^{T} \exp(u_{n,j})}, \qquad q_{n,i} = m + (1 - Tm)\tilde{q}_{n,i}, \tag{7}$$

where $m < 1/T$ enforces a minimum sampling probability. Unless stated otherwise, we use $m = 1/(4T)$, preventing task starvation and keeping task-level statistics up to date. The complete training loop is summarized in Algorithm 1.

Learning the sampler For a sampler q_ϕ, we initialize a MTPPO agent and train it while letting q_ϕ control task-sampling. Its fitness is the area under the curve (AUC) of mean normalized task performance,

$$F(\phi) = \mathrm{AUC}\left(\frac{1}{T}\sum_{i=1}^{T} \bar{J}_i(\theta_1), \ldots, \frac{1}{T}\sum_{i=1}^{T} \bar{J}_i(\theta_N)\right), \tag{8}$$

Algorithm 1 Multi-task PPO with DynATS

Initialize $q_{1,i} \leftarrow 1/T$ for all $i \in [T]$
for $n = 1, \ldots, N$ **do**
  $\mathcal{B} \leftarrow \emptyset$
  while $\mathcal{B}$ is not full **do**
    Sample task $i \sim q_n$
    Collect rollout τ from task i
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{\tau\}$
  end while
  Update shared policy π_θ with PPO using $\mathcal{B}$
  Compute task features X_n
  $\tilde{q}_{n+1} \leftarrow q_\phi(X_n)$
  $q_{n+1,i} \leftarrow m + (1 - Tm)\tilde{q}_{n+1,i}$ for all $i \in [T]$
end for

where $\bar{J}_i$ denotes normalized evaluation performance on task i. Evaluations use an equal number of episodes for every task and are used only for calculating the fitness. We optimize $F(\phi)$ with ES (Salimans et al., 2017), avoiding differentiation through the complete RL training process. When optimizing on a distribution of task sets, each antithetic ES pair is evaluated on the same sampled task set. We periodically evaluate the ES mean on validation task sets and retain the checkpoint with the highest validation fitness. The complete ES training procedure is given in Appendix B.2.

5 Experiments

Our experiments address four questions. **Q1: Can DynATS learn inter-task dynamics?** We test whether it can exploit positive/negative transfer across a distribution of task sets. **Q2: Can we learn a sampler for a fixed MTRL task set?** We train and evaluate DynATS on the same task set, testing whether it outperforms existing sampling strategies. **Q3: Can DynATS generalize to unseen task sets?** We train DynATS on a distribution of task sets and evaluate it zero-shot on unseen task sets from the same domain. **Q4: Can DynATS transfer across domains?** We train DynATS on one domain and deploy zero-shot on different domains.

5.1 Baselines

We compare against the task-sampling baselines used in Corrado et al. (2026), alongside their proposed method DRATS. **Uniform** maintains $q_{n,i} = 1/T$ throughout training. The three adaptive baselines construct a scalar priority $g_{n,i}$ for each task and convert these priorities into a sampling distribution using the same KL-regularized exponentiated-gradient update. We report the complete update rule and its hyperparameters in Appendix D.6.

DRATS (Corrado et al., 2026) prioritizes tasks according to their normalized performance gap,

$$g_{n,i}^{\mathrm{DRATS}} = \frac{J_i^{\mathrm{ref}} - \hat{J}_{n,i}}{\left|J_i^{\mathrm{ref}} - J_i^{\mathrm{rand}}\right| + \epsilon}. \tag{9}$$

Learning Progress (LPR) (Portelas et al., 2020) prioritizes tasks whose observed performance is changing,

$$g_{n,i}^{\mathrm{LPR}} = \left|\hat{J}_{n,i} - \hat{J}_{n-1,i}\right|. \tag{10}$$

Learning Potential (LP) (Jiang et al., 2021) uses the mean absolute GAE magnitude,

$$g_{n,i}^{\mathrm{LP}} = \mathbb{E}\left[|\hat{A}_t| \mid i\right], \tag{11}$$

as a proxy for the remaining policy-learning signal.

Thus, these adaptive baselines share the same allocation mechanism but differ in the signal used to determine which tasks should receive additional samples. We do not include Hard Tasks First (Cho et al., 2024), as Corrado et al. (2026) report that DRATS consistently outperforms it in a similar setting to ours (on-policy MTRL).

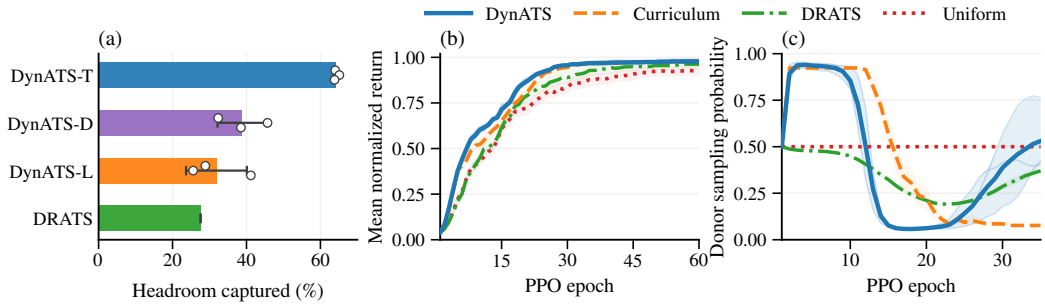


Figure 2: (a) Synthetic experiment comparing DynATS-Transformer (DynATS-T), DynATS-DeepSets (DynATS-D), DynATS-Linear (DynATS-L), and DRATS. Headroom capture is computed from mean AUC over 256 held-out task sets, relative to Uniform and the dynamic oracle. For the learned samplers, bars show the mean over three independent ES seeds, circles show the individual ES runs, and error bars denote one standard deviation across ES seeds. (b) Mean normalized return on Chain. (c) Total sampling probability assigned to the two Chain donor tasks. In (b–c), all curves are averaged over 100 held-out PPO seeds per sampler. Shaded regions denote 95% bootstrap confidence intervals.

5.2 Capturing Inter-Task Dynamics

We first study whether DynATS can learn the dynamics between tasks in a controlled multi-task learning setting. We construct five-task sets in which each task's performance depends on the cumulative samples allocated to all tasks. The interaction matrix W defines how samples allocated to task i affect the performance of task j, allowing for both positive and negative transfer. We use ES to train DynATS with three different architectures on 256 task sets and evaluate on 256 unseen sets. We test a linear architecture, a non-linear MLP based architecture (Deepsets) and a transformer architecture. See Appendix D.1 for the complete experiment details.

Performance is measured as the fraction of *headroom* captured between uniform sampling and an oracle with access to the structure of the task set. Figure 2 (a) compares the learned samplers with DRATS, the best-performing baseline. DRATS captures 26.2% of the available headroom, while DynATS-Linear reaches 31.9%. DeepSets captures 38.9%, while the Transformer captures 64.3%. Unlike the oracle, the learned samplers must infer the tasks' dynamics online, which can partially explain the remaining gap. Nonetheless, this experiment demonstrates that DynATS can capture the dynamics between tasks (**Q1**) and validates the transformer architecture.

5.3 Multi-Task Sampler

We construct a simple multi-task RL benchmark to show that we can learn a multi-task sampler that outperforms existing sampling strategies. *Chain* contains two easier (*donor*) tasks and two harder (*recipient*) tasks with a simple curriculum: early interactions are most useful on the donors, while later interactions should move to the recipients. For more details, see Appendix D.2.

Figures 2(b–c) show that DynATS discovers this behavior. It shifts from donors to recipients as training progresses, learning faster than Uniform, DRATS, and even a hand-designed curriculum that focuses on the two donor tasks until they reach a return threshold of 0.9, after which it switches to the two recipients. DynATS achieves an AUC of 0.838, compared with 0.719 for DRATS and 0.689 for Uniform. It also outperforms the hand-designed curriculum (0.808 AUC), primarily by shifting probability mass from donors to recipients earlier than the curriculum does. The per-task returns and sampling probabilities are shown in Appendix C.1. This experiment demonstrates that DynATS can outperform sampling strategies that fix a prioritization rule a priori for a given MTRL task set **(Q2)**, by learning when the value of sampling different tasks changes over the course of training.

5.4 Same-Domain Generalization

We next ask whether a sampler trained on a distribution over task sets can generalize zero-shot to unseen task sets from the same domain. We use two grid-based domains: the procedural GridWorld

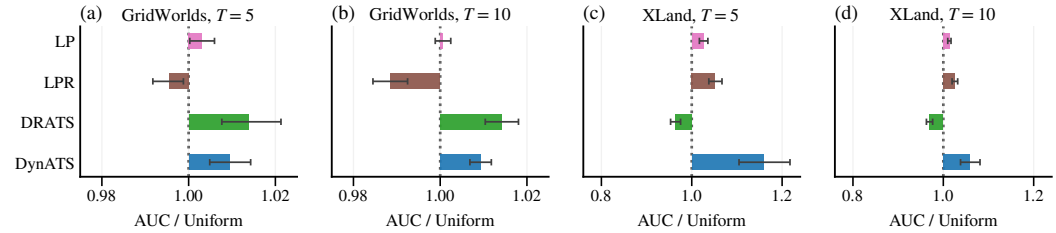


Figure 3: AUC of DynATS and baselines normalized against uniform sampling for 32 held-out sets of $T = 5$ and $T = 10$ tasks from both the Gridworlds and the XLand benchmarks. All AUC results are obtained from the mean normalized returns over 100 random PPO seeds.

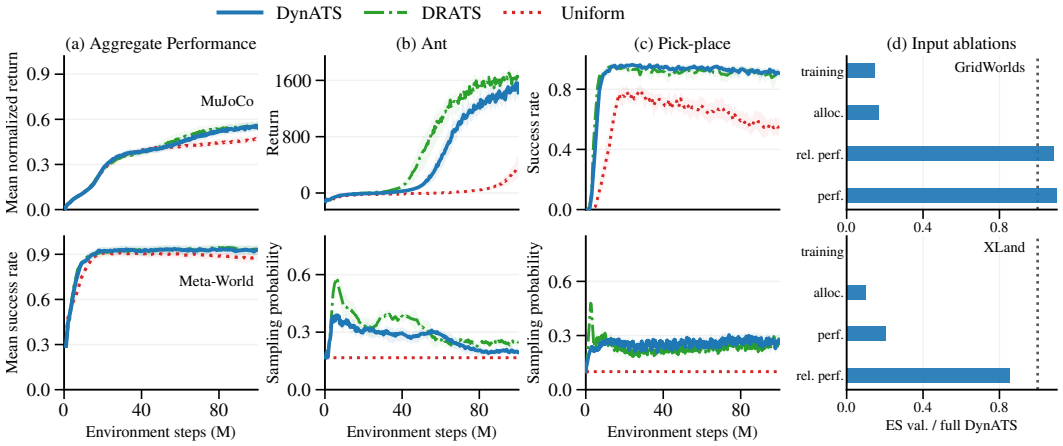


Figure 4: (a): Mean normalized return on all six Mujoco tasks and mean success rate on all ten Meta-World tasks over 10 and 20 seeds, respectively. (b): Individual returns and sampling probabilities on the Ant (Mujoco) task. (c): Individual returns and sampling probabilities on the Pick-place (Meta-World) task. (d): Ablations of the input features on the GridWorlds (mean over 3 seeds) and XLand (only 1 seed) domains.

distribution from Jackson et al. (2024) and Goldie et al. (2024), as well as a custom distribution of tasks generated from the XLand domain (Nikulin et al., 2024). Even though these are RL benchmarks, we construct MTRL task sets by independently sampling T tasks from the given distributions. We keep held-out test datasets of 32 task sets with $T = 5$ and $T = 10$ tasks from both domains, where we evaluate the learned samplers *without finetuning*, while the baselines are tuned for each task set. The full experimental details can be found in Appendix D.3.

Figure 3 shows that DynATS rivals DRATS in the GridWorld domain, even though the margins between different sampling strategies are small. DyNATS also transfers successfully to sets of $T = 10$ tasks, despite only being trained on sets of $T = 5$ tasks. For the XLand benchmark, DynATS outperforms all existing baselines, but there is a performance drop from $T = 5$ to $T = 10$ tasks. Nonetheless, this experiment proves three points: (i.) No sampling strategy that encodes a fixed prioritization rule is universally good: DRATS is the best sampling strategy in the GridWorld domain and worse than uniform in XLand, which motivates DynATS; (ii.) DynATS can generalize to unseen task sets **(Q3)** and (iii.) to larger task sets than those it was trained on.

5.5 Cross-Domain Generalization

Finally, we test whether DynATS trained on one domain can transfer zero-shot to a substantially different one. We reuse DynATS trained on GridWorld with $T = 5$ and deploy it directly on *MuJoCo6* and *Meta-World10* (results for Meta-World50 are also shown in Appendix C.4). Mujoco6 contains 6 tasks with different state and action-spaces. Meta-World10/50 contains $10/50$ robotic manipulation tasks with the same state and action-spaces. This represents a strong distribution shift, since the

training domain consists of sets of $T = 5$ maze-like environments with discrete actions. Despite this shift, DyNATS receives the same task-learning features and *requires no finetuning*. We follow as closely as possible the implementation details and hyperparameters specified by Corrado et al. (2026) and compare DynATS only against uniform sampling and DRATS (since it was shown to be the best performing sampling strategy in Corrado et al. (2026)). The full experimental details can be found in Appendix D.4.

Figures 4 (a-c) show that DynATS substantially outperforms uniform sampling and remains competitive with DRATS, despite DRATS being tuned for the target sets. Interestingly, DynATS adopts a qualitatively similar strategy to DRATS, assigning more samples to the harder tasks, as can be seen for the Ant and Pick-Place tasks from Mujoco and Meta-World, respectively. Both methods improve aggregate performance largely through higher returns on these harder tasks relative to uniform sampling. This experiment demonstrates that: (i.) DynATS can generalize across domains since it's guided only by learning dynamics **(Q4)**; (ii.) DynATS remains competitive with DRATS despite no finetuning. This is advantageous from a runtime viewpoint, as ES training on the GridWorld is substantially faster (roughly 1.3 hours on a H200 GPU) than the several Mujoco6 and MetaWorld10/50 runs DRATS requires to be optimized, see Appendix C.6 for more details; (iii.) DynATS can be successfully deployed using running statistics to normalize the performance metrics, as can be seen from the Mujoco6 results, where no reference values were provided.

5.6 Ablation of Input Features

We study which components of the sampler state are necessary for learning an effective sampling strategy. Starting from the full input set in Table 1, we train separate samplers using only: (i.) training proportion, (ii.) allocation features, (iii.) normalized performance features, or (iv.) relative performance features, and compare them with the full model. We repeat this ablation on GridWorld and XLand. For GridWorld, we report results over three ES seeds; for XLand, we use a single seed due to the substantially higher ES training cost, as is common in computationally intensive meta-learning and learned-optimization settings (Jackson et al., 2024; Goldie et al., 2024).

Results are shown in Figure 4 (d). Training proportion or allocation alone are insufficient on both domains, whereas performance features enable effective adaptive sampling. On GridWorld, either normalized or relative performance features alone are enough to learn a strong strategy, and can even outperform the full input set. On XLand, relative performance is more informative than normalized performance, while the best result is obtained with the complete feature set. This suggests that GridWorld admits simpler sampling rules, consistent with our observation that DynATS learns a DRATS-like strategy, whereas XLand benefits from combining absolute performance, cross-task comparisons, allocation history, and training progress. Overall, the ablation does not indicate that every feature is necessary in every domain, but that the full representation is the most robust choice across domains without assuming in advance which statistics will be informative.

6 Discussion and Conclusion

Existing adaptive task-sampling methods encode an a-priori assumption about what makes a task worth sampling, for example its performance gap, learning progress, or advantage magnitude. Our results show that the usefulness of these signals depends on the underlying task distribution, motivating DynATS: a method to learn a task-sampler that maps MTRL training signals directly to task-allocations. Across controlled experiments, DynATS captures inter-task learning dynamics, discovers useful curricula, and generalizes zero-shot to previously unseen task sets and to larger task sets. Most notably, a sampler trained only on GridWorld in just over an hour (see Appendix C.6) transfers directly to MuJoCo6 and Meta-World10/50 without any finetuning, remaining competitive with tuned DRATS. These results suggest that useful sampling strategies can be learned on inexpensive domains and reused beyond the distribution on which they were learned.

Limitations Training DynATS with ES incurs a substantial one-off ES training cost. Deployment, however, is lightweight and requires only task-level statistics already available during training and a forward pass through the frozen learned sampler. Our experiments also indicate that different domains can favor different sampling strategies. Therefore, a natural next step is to train DynATS on broader mixtures of task distributions and learning dynamics, enabling a single sampler to recognize and adapt to a wider range of domains.

AI USE STATEMENT

In this work, we used generative AI tools to (i) assist in formulating and writing a proof of a mathematical claim, (ii) implement portions of our methods, and (iii) provide feedback on research experiments. We did not use generative AI tools for other applicable tasks requiring disclosure, including developing the main theoretical or conceptual framework, proposing or refining the central research hypotheses, generating synthetic datasets, or assisting with translation. The remaining required-disclosure tasks are not applicable to this work.

Additionally, we used generative AI tools to identify potentially relevant literature, generate and edit code for scientific plots, and improve the readability of text initially written by the authors. Generative AI tools were not used to draft sections of the paper.

For (i), LLMs assisted with a proof used to contextualize and motivate our work. This proof is not a main contribution and it was reviewed by several authors. For (ii), LLMs were used to accelerate experimentation by generating parts of the code. All AI-generated code was reviewed by an author. For (iii.), LLMs were used at times to provide simple feedback about research experiments.

All AI-rewritten text was reviewed and further modified by the authors. For literature research, all references suggested by LLMs were manually checked. Finally, the authors manually searched for relevant work that might have been missed by the LLMs.

We take responsibility for the final content of this work, including all text, claims, code, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have taken steps to ensure that all experiments in this work are reproducible. Appendix D provides the relevant experimental details and Appendix E the hyperparameters. In addition, the code required to reproduce all reported results is provided in the supplementary materials. It includes a README with instructions and scripts for reproducing the experiments and results presented in this work.

REFERENCES

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

Emma Brunskill and Lihong Li. Sample complexity of multi-task reinforcement learning. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, UAI'13, pp. 122–131, Arlington, Virginia, USA, 2013. AUAI Press.

Mayee F. Chen, Nicholas Roberts, Kush Bhatia, Jue Wang, Ce Zhang, Frederic Sala, and Christopher Ré. Skill-it! a data-driven skills framework for understanding and training language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-1562. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/70b8505ac79e3e131756f793cd80eb8d-Abstract-Conference.html.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks, 2018. URL https://arxiv.org/abs/1711.02257.

Myungsik Cho, Jongeui Park, Suyoung Lee, and Youngchul Sung. Hard tasks first: multi-task reinforcement learning through task scheduling. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Nicholas E. Corrado, Wenyuan Huang, and Josiah P. Hanna. Distributionally robust multi-task reinforcement learning via adaptive task sampling, 2026. URL https://arxiv.org/abs/2605.14350. Accepted to NeurIPS 2026.

Carlo D'Eramo and Georgia Chalvatzaki. Prioritized sampling with intrinsic motivation in multi-task reinforcement learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2022. doi: 10.1109/IJCNN55064.2022.9892973.

Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Sharing knowledge in multi-task deep reinforcement learning, 2024. URL https://arxiv.org/abs/2401.09561.

Alexander D. Goldie, Chris Lu, Matthew T. Jackson, Shimon Whiteson, and Jakob N. Foerster. Can learned optimization make reinforcement learning less difficult? In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Alex Graves, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1311–1320. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/graves17a.html.

Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. Dynamic task prioritization for multitask learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Nikolaus Hansen, Dirk V Arnold, and Anne Auger. Evolution strategies. In *Springer handbook of computational intelligence*, pp. 871–898. Springer, 2015.

Ahmed Hendawy, Jan Peters, and Carlo D'Eramo. Multi-task reinforcement learning with mixture of orthogonal experts. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=aZH1dM3GOX.

Hanchi Huang, Deheng Ye, Li Shen, and Wei Liu. Curriculum-Based Asymmetric Multi-Task Reinforcement Learning . *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 45(06): 7258–7269, June 2023. ISSN 1939-3539. doi: 10.1109/TPAMI.2022.3223872. URL https://doi.ieeecomputersociety.org/10.1109/TPAMI.2022.3223872.

Matthew Thomas Jackson, Chris Lu, Louis Kirsch, Robert Tjarko Lange, Shimon Whiteson, and Jakob Nicolaus Foerster. Discovering temporally-aware reinforcement learning algorithms. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=MJJcs3zbmi.

Minqi Jiang, Ed Grefenstette, and Tim Rocktäschel. Prioritized level replay. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139, pp. 4940–4950. PMLR, 2021. URL https://proceedings.mlr.press/v139/jiang21b.html.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Zeman Li, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. PiKE: Adaptive data mixing for large-scale multi-task learning under low gradient conflicts. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-5640. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f7a94134f1c726796c6f81fb946e489d-Abstract-Conference.html.

Po-Shao Lin, Jia-Fong Yeh, Yi-Ting Chen, and Winston H. Hsu. Shared-unique features and task-aware prioritized sampling on multi-task reinforcement learning, 2024. URL https://arxiv.org/abs/2406.00761.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Bo Liu, Yihao Feng, Peter Stone, and Qiang Liu. Famo: fast adaptive multitask optimization. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Shikun Liu, Stephen James, Andrew J. Davison, and Edward Johns. Auto-λ: Disentangling dynamic task relationships. *Transactions on Machine Learning Research*, 2022. URL https://openreview.net/forum?id=KKeCMim5VN.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning, 2017. URL https://arxiv.org/abs/1707.00183.

Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. Multi-task learning as a bargaining game. *CoRR*, abs/2202.01017, 2022. URL https://arxiv.org/abs/2202.01017.

Alexander Nikulin, Vladislav Kurenkov, Ilya Zisman, Artem Agarkov, Viacheslav Sinii, and Sergey Kolesnikov. Xland-minigrid: scalable meta-reinforcement learning environments in jax. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of Deep RL in continuously parameterized environments. In *Proceedings of the 3rd Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 835–853. PMLR, 2020. URL https://proceedings.mlr.press/v100/portelas20a.html.

Amelie Royer, Tijmen Blankevoort, and Babak Ehteshami Bejnordi. Scalarization for multi-task and multi-domain learning at scale. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=TSuq3debnD.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017. URL https://arxiv.org/abs/1703.03864.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9767–9779. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/sodhani21a.html.

Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9120–9132. PMLR, 2020. URL https://proceedings.mlr.press/v119/standley20a.html.

Lingfeng Sun, Haichao Zhang, Wei Xu, and Masayoshi Tomizuka. Paco: parameter-compositional multi-task reinforcement learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy Liang, Quoc V. Le, Tengyu Ma, and Adams Wei Yu. DoReMi: Optimizing data mixtures speeds up language model pretraining. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-3059. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/dcba6be91359358c2355cd920da3fcbd-Abstract-Conference.html.

Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Huaxiu Yao, Yu Wang, Ying Wei, Peilin Zhao, Mehrdad Mahdavi, Defu Lian, and Chelsea Finn. Meta-learning with an adaptive task scheduler. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021. URL https://arxiv.org/abs/1910.10897.

APPENDIX

A Additional Theory **15**

A.1 Fixed Interpretations of a Learning Signal 15

B DynATS Details **16**

B.1 Input Features 16

B.2 ES-Training Algorithm 17

B.3 Transformer Architecture 17

C Additional Experimental Results **19**

C.1 Chain 19

C.2 MuJoCo6 19

C.3 MetaWorld10 19

C.4 MetaWorld50 21

C.5 Worst-Task Performance 23

C.6 DynATS training and deployment: computational effort 24

D Details on Experimental Results **25**

D.1 Capturing Inter-Task Dynamics 25

D.2 Multi-Task Sampler 26

D.3 Same-Domain Generalization 26

D.4 Cross-Domain Generalization 27

D.5 Input Feature Ablations 28

D.6 Baseline Sampling Strategies 28

E Hyperparameters **30**

E.1 DynATS and ES Training 30

E.2 Inner-Loop MTPPO 30

E.3 Evaluation and Baseline Tuning 30

E.4 Cross-Domain Evaluation 31

A Additional Theory

This section formalizes the toy contextual-bandit example used in Section 4. The result is intentionally limited: it shows that a single hand-specified interpretation of an observed learning signal may not transfer across environments. It does not provide a regret guarantee for our learned sampler or characterize the full MTRL optimization dynamics.

A.1 Fixed Interpretations of a Learning Signal

Proof. Consider two tasks indexed by $A \in \{-1, +1\}$. At scheduling round n, the sampler observes a signal $z_n \in \{-1, +1\}$. We compare two environments, indexed by $\xi \in \{-1, +1\}$, with expected task utilities

$$\mu_n^\xi(A) = \frac{1}{2} + \eta \xi z_n A, \qquad 0 < \eta < \frac{1}{2}. \tag{12}$$

The argument holds conditional on any fixed sequence of observed signals $z_1, \ldots, z_N$.

For $\xi = +1$, the optimal task is

$$A_n^\star = z_n, \tag{13}$$

whereas for $\xi = -1$ it is

$$A_n^\star = -z_n. \tag{14}$$

In either environment, choosing the suboptimal task incurs utility loss 2η.

Let $q(\cdot \mid z)$ be any fixed, possibly stochastic, mapping from the observed signal to a distribution over the two tasks. Define its expected instantaneous regret in environment ξ as

$$r_n^\xi(q) = \mu_n^\xi(A_n^\star) - \mathbb{E}_{A \sim q(\cdot \mid z_n)} \left[\mu_n^\xi(A) \right], \tag{15}$$

and its cumulative regret as

$$R_N^\xi(q) = \sum_{n=1}^{N} r_n^\xi(q). \tag{16}$$

Write

$$p_n = q(A = z_n \mid z_n) \tag{17}$$

for the probability that the rule follows the observed signal. In the $\xi = +1$ environment, following the signal is optimal, and hence

$$r_n^+(q) = 2\eta(1 - p_n). \tag{18}$$

In the $\xi = -1$ environment, reversing the signal is optimal, giving

$$r_n^-(q) = 2\eta p_n. \tag{19}$$

Therefore,

$$r_n^+(q) + r_n^-(q) = 2\eta \tag{20}$$

at every round. Summing over N scheduling decisions yields

$$R_N^+(q) + R_N^-(q) = 2\eta N. \tag{21}$$

At least one of these two nonnegative regrets must therefore be at least half their sum:

$$\max_{\xi \in \{-1, +1\}} R_N^\xi(q) \geq \eta N. \tag{22}$$

This proves Proposition 1. □

The result is a minimal no-universal interpretation statement. It does not claim that a learned sampler supplied with the same single signal necessarily avoids this lower bound. Rather, it shows that a hand-specified relationship between one observed learning statistic and task priority can be systematically wrong when the meaning of that statistic differs across environments. Our sampler instead receives multiple task-state features and learns during training how they should be combined. Whether this learned mapping captures useful structure is established empirically.

B DynATS Details

B.1 Input Features

For each task i at sampler epoch n, DynATS receives the seven task-level features summarized in Table 1. Let $\hat{J}_{n,i}$ denote the current performance estimate, and let $J_i^{\min}$ and J_i^{ref} denote lower and upper reference values. We define the performance range

$$\Delta_i = J_i^{\mathrm{ref}} - J_i^{\min}. \tag{23}$$

For any task-level quantity a_i, we denote standardization across the T tasks by

$$z(a_i) = \frac{a_i - \frac{1}{T}\sum_{j=1}^{T} a_j}{\mathrm{std}(a_1, \ldots, a_T) + \epsilon}, \tag{24}$$

where ϵ is a small constant for numerical stability.

Relative Allocation. Let $q_{n,i}$ be the probability of sampling task i. We express this probability relative to uniform sampling:

$$x_{n,i}^{\mathrm{alloc}} = \log(T q_{n,i}). \tag{25}$$

Thus, $x_{n,i}^{\mathrm{alloc}} = 0$ corresponds to uniform sampling, while positive and negative values indicate over- and under-sampling, respectively.

Relative Cumulative Allocation. Let $a_{n,i}$ denote the fraction of the rollout budget actually allocated to task i at epoch n. We define

$$x_{n,i}^{\mathrm{cum}} = \frac{T}{n} \sum_{k=1}^{n} a_{k,i}. \tag{26}$$

A value of one means that task i has received exactly the same budget up to epoch n as would be allocated via uniform sampling.

Performance. We normalize task performance to a common scale:

$$x_{n,i}^{\mathrm{perf}} = \mathrm{clip}\left(\frac{\hat{J}_{n,i} - J_i^{\min}}{\Delta_i}, 0, 1\right). \tag{27}$$

Performance Change. The normalized change in performance since the previous sampler epoch is

$$x_{n,i}^{\Delta\mathrm{perf}} = \mathrm{clip}\left(\frac{\hat{J}_{n,i} - \hat{J}_{n-1,i}}{\Delta_i}, -1, 1\right). \tag{28}$$

This feature is set to zero at the first epoch.

Relative Performance. To describe how a task is performing relative to the other tasks in the current set, we standardize normalized performance across tasks:

$$x_{n,i}^{\text{rel-perf}} = z\left(x_{n,i}^{\mathrm{perf}}\right). \tag{29}$$

Relative Performance Change. Similarly, we standardize performance change across tasks:

$$x_{n,i}^{\text{rel-}\Delta\mathrm{perf}} = z\left(x_{n,i}^{\Delta\mathrm{perf}}\right). \tag{30}$$

Training Proportion. Finally, every task receives the fraction of the training horizon that has elapsed:

$$x_{n,i}^{\mathrm{train}} = \frac{n-1}{N-1}, \tag{31}$$

where N is the total number of PPO epochs.

When a task is not sampled during an epoch, its most recent performance estimate is carried forward. When fixed reference values $J_i^{\min}$ and J_i^{ref} are unavailable, they are replaced by running task-level estimates as described in Section 4.

B.2 ES-TRAINING ALGORITHM

Algorithm 2 ES training of the task-sampler with antithetic sampling

Require: Task-suite distribution $\mathcal{D}$, population size P, generations G, noise scale σ
Initialize sampler center ϕ
for $g = 1, \dots, G$ **do**
  for $j = 1, \dots, P/2$ **in parallel do**
    Sample $\epsilon_j \sim \mathcal{N}(0, I)$
    Sample task suite $\mathcal{E}_j \sim \mathcal{D}$ and PPO seed ξ_j
    $\phi_j^+ \leftarrow \phi + \sigma\epsilon_j$, $\phi_j^- \leftarrow \phi - \sigma\epsilon_j$
    $F_j^+ \leftarrow \text{MTRL-AUC}(\phi_j^+; \mathcal{E}_j, \xi_j)$
    $F_j^- \leftarrow \text{MTRL-AUC}(\phi_j^-; \mathcal{E}_j, \xi_j)$
    $s_j \leftarrow \text{sign}(F_j^+ - F_j^-)$
  end for
  $\widehat{\nabla_\phi F} \leftarrow \frac{1}{P\sigma} \sum_{j=1}^{P/2} s_j \epsilon_j$
  Update ϕ with Adam using $\widehat{\nabla_\phi F}$
  Periodically evaluate and checkpoint ϕ on held-out validation runs
end for
return Best Validation Center ϕ

B.3 TRANSFORMER ARCHITECTURE

DynATS uses a small Transformer architecture with roughly four thousand parameters that jointly models relationships between tasks and their evolution over training. Figure 5 summarizes the architecture. At sampling step n, the input is a history of task-level features

$$X_n \in \mathbb{R}^{H \times T \times F},$$

where H is the history length, T is the number of tasks, and F is the number of input features per task. Each feature vector is first projected independently into a d-dimensional embedding,

$$e_{h,i} = W_{\text{in}} x_{h,i} + b_{\text{in}} + p_h,$$

where p_h is a fixed sinusoidal positional encoding for temporal position h.

The projected representations are processed by two Transformer blocks. First, *task attention* is applied independently at every history position. For each h, the T task representations attend to one another through four-head self-attention. This allows the representation of task i to depend on the state of the other tasks in the current task set.

The resulting representations are then processed by *temporal attention*. For each task independently, four-head self-attention is applied across its H history positions. This allows the sampler to condition its decision not only on the current task statistics, but also on how those statistics have evolved during training. The temporal block has the same layer-normalization, residual, and feed-forward structure as the task-attention block.

After temporal attention, we retain only the representation corresponding to the most recent history position for each task. A final layer normalization and shared linear readout produce one scalar logit $u_{n,i}$ per task. These logits are converted to sampling probabilities using

$$\tilde{q}_{n+1,i} = \frac{\exp(u_{n,i})}{\sum_{j=1}^{T} \exp(u_{n,j})}, \qquad q_{n+1,i} = m + (1 - Tm)\tilde{q}_{n+1,i},$$

where $m < 1/T$ is the minimum sampling probability.

Importantly, no task identities or task positional embeddings are provided. Parameters are shared across tasks and task attention is permutation equivariant: permuting the ordering of tasks therefore produces the same permutation of the output sampling probabilities. In contrast, temporal positional encodings are included because the ordering of observations within the training history is meaningful. This separation allows the same sampler to operate on previously unseen task sets and on different numbers of tasks.

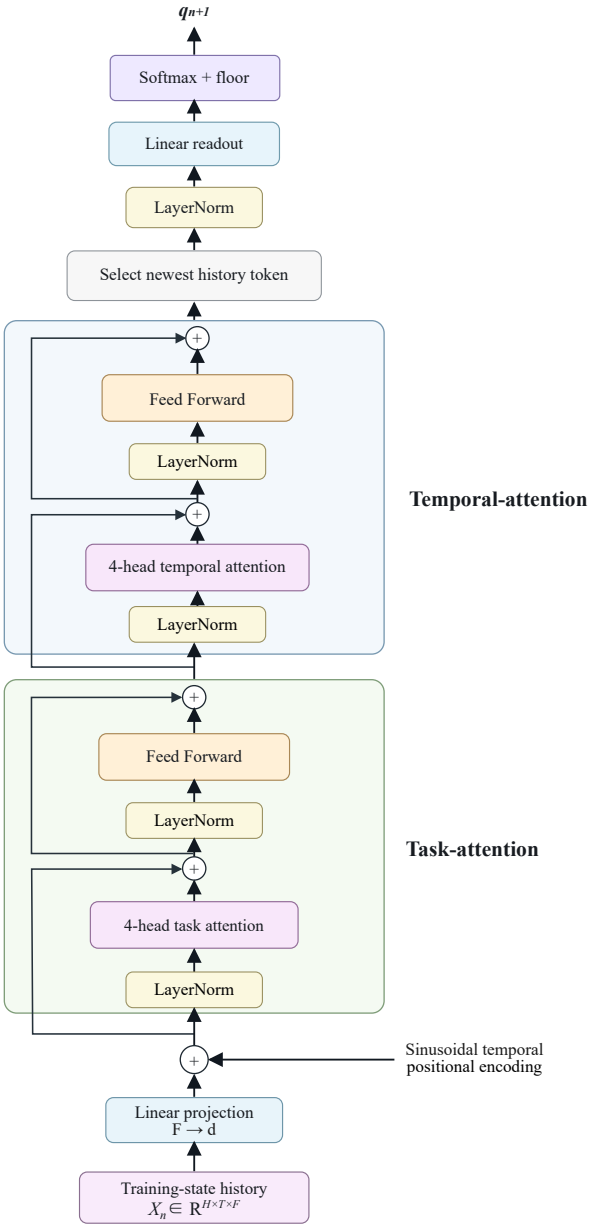


Figure 5: **DynATS Transformer architecture.** The training-state history is projected to a shared latent representation. Task attention first models interactions between tasks at each history position, followed by temporal attention over the history of each task. The most recent representation of each task is mapped to a sampling logit, and a softmax with a minimum-probability floor produces the next sampling distribution q_{n+1}.

C Additional Experimental Results

C.1 Chain

Figure 6 shows the per-task returns and sampling probabilities on all Chain tasks for DynATS, the hand-desgined curriculum, DRATS and uniform sampling.

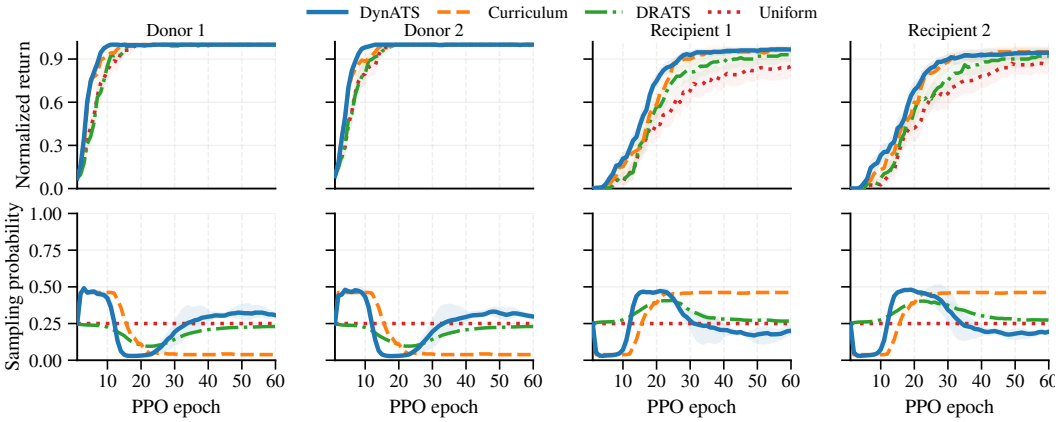


Figure 6: Per-task retuns and sampling probabilities for the Chain benchmark. Curves represent the mean over 100 independent seeds with 95 percentage confidence interval.

C.2 MuJoCo6

Figure 7 shows the per-task returns and sampling probabilities on all MuJoCo6 tasks for DynATS (trained on the GridWorld distribution), DRATS and Uniform Sampling. As stated in Section 5.5, DynATS learned a similar strategy to DRATS, with both methods assigning extremely similar sampling probabilities to all tasks throughout training. We remind the reader that DynATS is deployed *without finetuning*, while DRATS has been finetuned for this specific benchmark by Corrado et al. (2026) by searching over the temperature of the softmax.

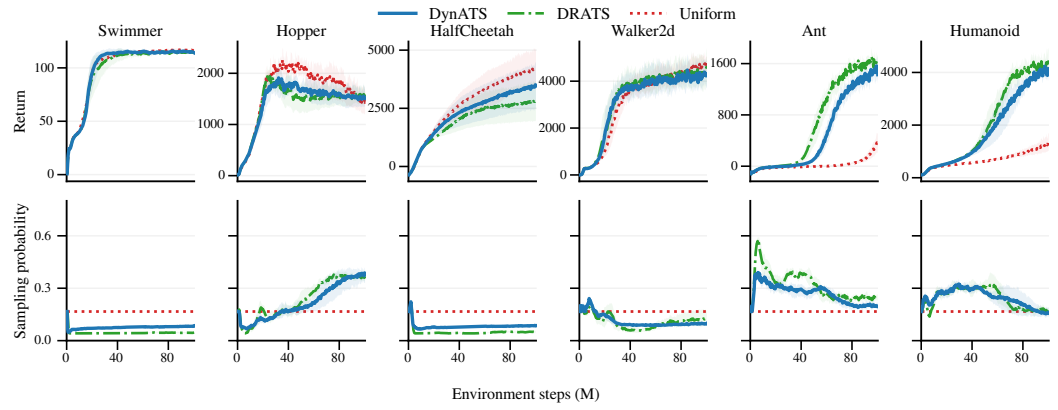


Figure 7: Per-task retuns and sampling probabilities for the MuJoCo6 benchmark. Curves represent the mean over 10 independent seeds with 95 percentage confidence interval.

C.3 MetaWorld10

Figure8 shows the per-task success rates and sampling probabilities on all Meta-World 10 tasks for DynATS (trained on the GridWorld distribution), DRATS and Uniform Sampling. The outcomes

are extremely similar to the ones from the MuJoCo6 experiment, see Appendix C.2. DynATS and DRATS are indistinguishable, sampling the harder tasks (e.g. Pick-place) more often.

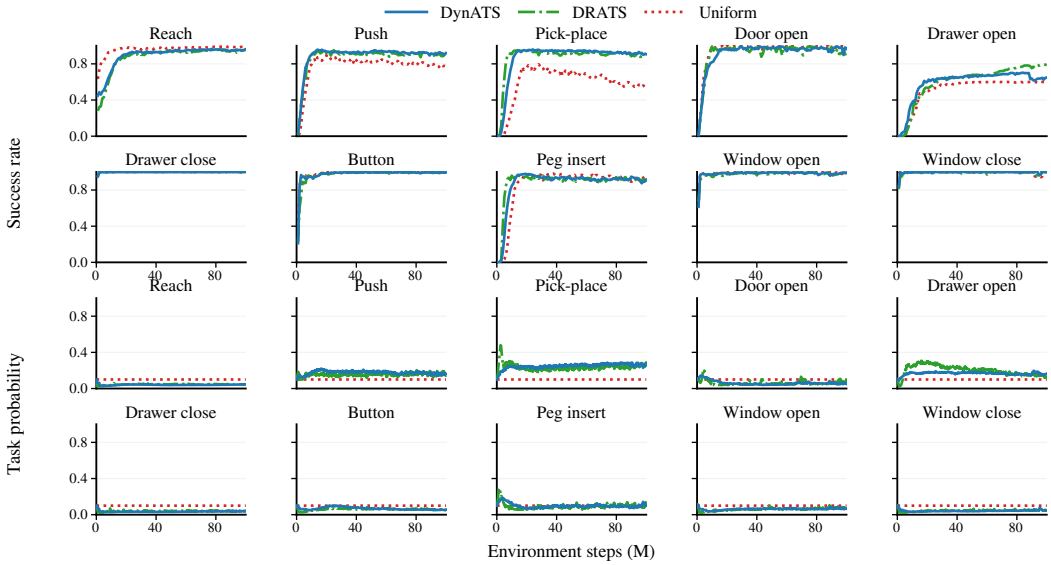


Figure 8: Per-task success rates and sampling probabilities for the Meta-World 10 benchmark. Curves represent the mean over 20 independent seeds

C.4 METAWORLD50

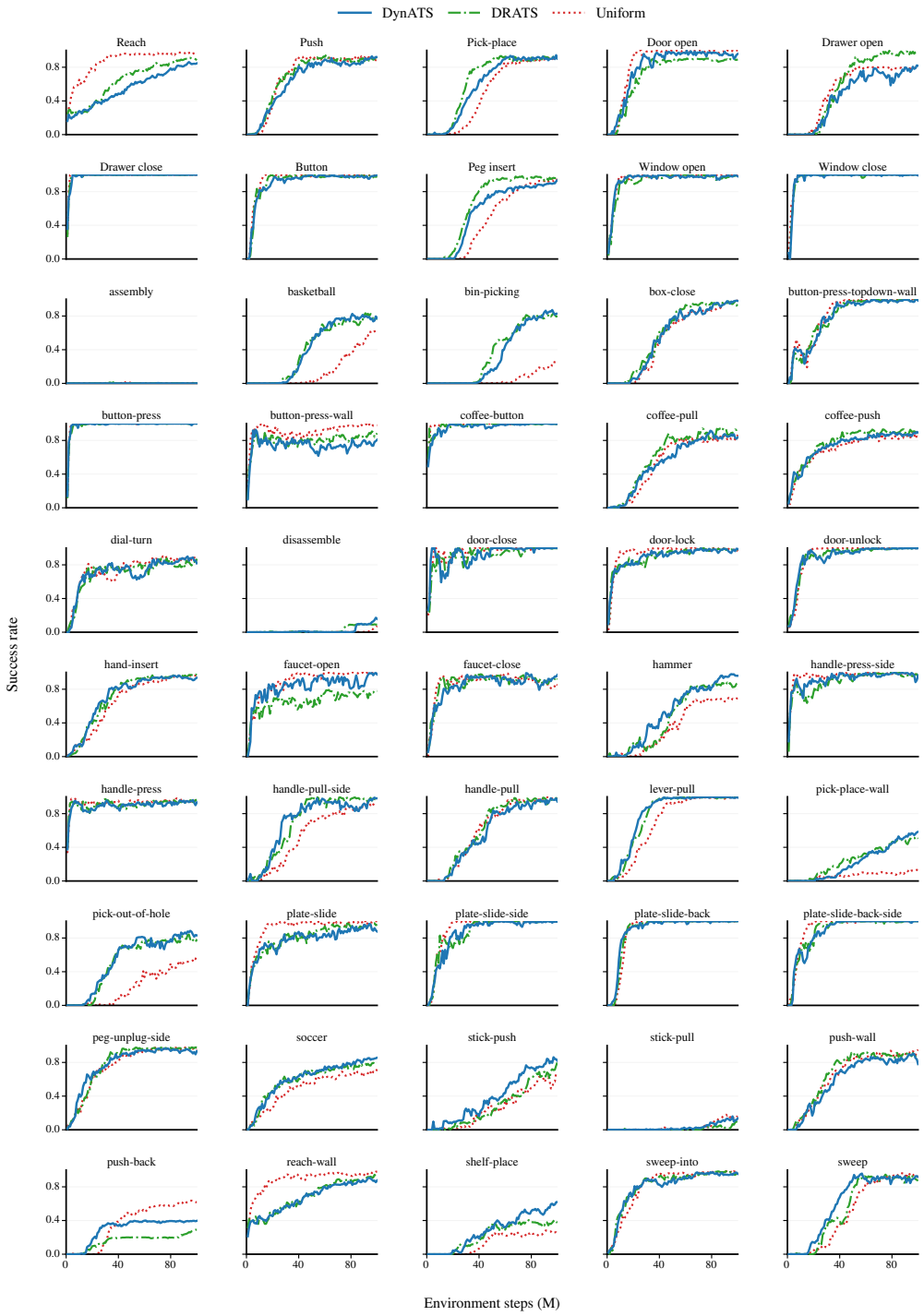


Figure 9: Per-task success rates for the Meta-World 50 benchmark. Curves represent the mean over 10 independent seeds

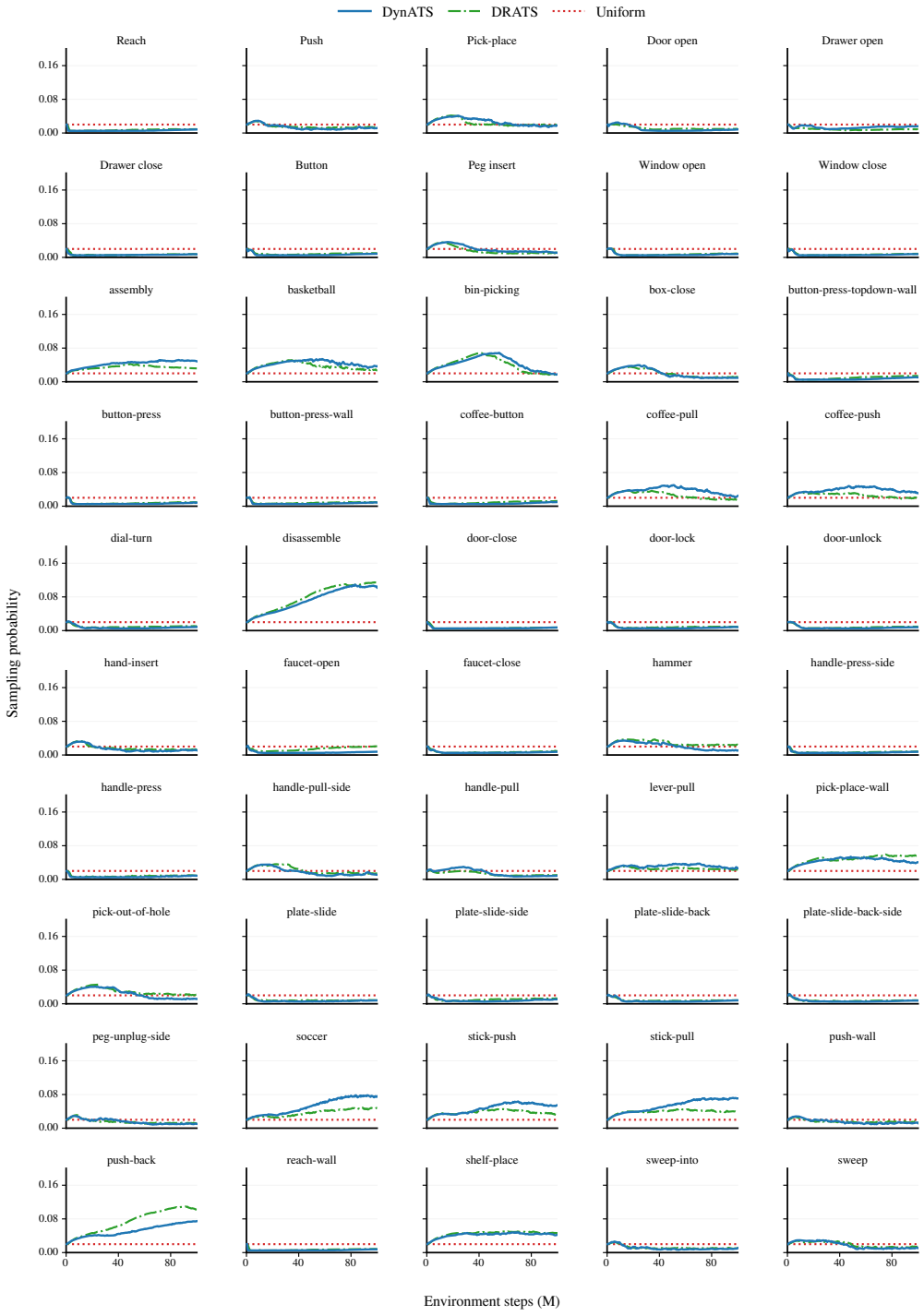


Figure 10: Per-task sampling probabilities for the Meta-World 50 benchmark. Curves represent the mean over 10 independent seeds

Figures 9 and 10 show the per-task success rates and sampling probabilities on all Meta-World 50 tasks for DynATS (trained on the GridWorld distribution), DRATS and Uniform Sampling. Once again, the outcomes are similar to the ones from the MuJoCo6 and Meta-World 10 experiments, see Appendix C.2 and C.3. DynATS and DRATS demonstrate similar sampling strategies, but here the

differences are more noticeable. The general strategy of sampling the harder tasks more often is still present in both methods and they over-sample and under-sample the same tasks. However, there is a larger relative difference in the sampling probabilities assigned to the tasks. For example, DRATS samples experiences from push-back consistently more often than DynATS. Curiously, however, DRATS is the worst performing method on that particular task and uniform sampling is even the best method, despite sampling it the least out of the three methods. This demonstrates that, in MTRL, due to complex cross-task dynamics throughout training, sampling a particular task more often does not necessarily correlate with a better performance on said task.

Figure 11 shows the mean success rate on all Meta-World 50 tasks. DynATS and DRATS outperform uniform sampling, achieving a success rate after $100M$ environment steps roughly 3% higher.

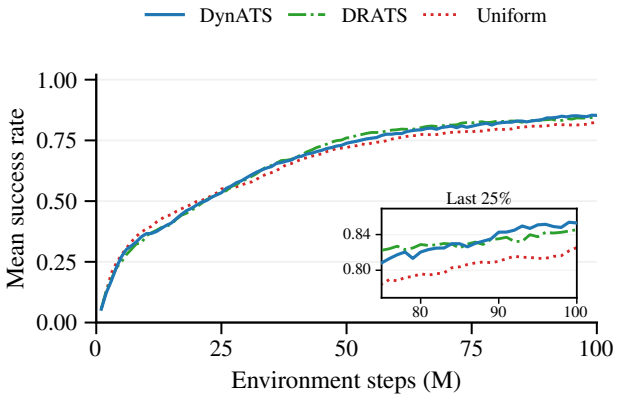


Figure 11: Mean success rate over the 50 Meta-World 50 tasks. The inset zooms in on the final 25% environment steps. All curves are obtained by calculating the mean over 10 seeds.

C.5 Worst-Task Performance

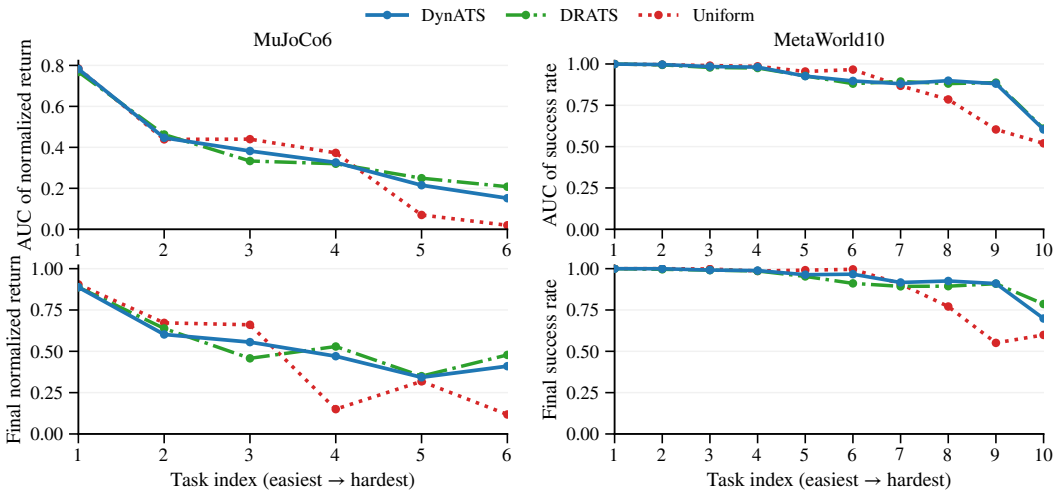


Figure 12: AUC of performance (top row) and final performance (bottom row) for all Mujoco6 and Meta-World10 tasks, averaged over 10 and 20 seeds, respectively.

Figure 12 shows the AUC of performance (top row) and final performance (bottom row) for all MuJoCo6 and Meta-World10 tasks. For MuJoCo6, performance is measured as the mean normalized return, and for Meta-World10 as the mean success rate. In both benchmarks, DRATS and DynATS

sacrifice a small amount of performance on the easier tasks to achieve a higher performance on the harder tasks, similar to the findings of Corrado et al. (2026).

C.6 DynATS training and deployment: computational effort

C.6.1 ES training

Training DynATS is considerably more expensive than deploying it. During training, ES evaluates each candidate sampler through a complete MTRL training run, resulting in a large number of environment interactions. However, the ES population and inner MTPPO runs are vectorized, such that many candidate samplers are evaluated in parallel on a single GPU.

Table 2 reports the computational cost of representative end-to-end ES runs for the GridWorld and XLand domains. We report the complete wall-clock time, including compilation and validation, together with the number of environment transitions consumed by ES training and checkpoint validation. All training hyperparameters are reported in Appendix E.

Table 2: Computational effort of DynATS ES training runs. Environment interactions include all candidate MTPPO runs executed during ES training; validation interactions correspond to periodic evaluation of the ES center on held-out task sets. These experiments were performed on an Nvidia H200 GPU.

	GridWorld	XLand
ES generations	250	200
ES population	512	256
Training interactions	19.20B	51.20B
Validation interactions	6.24B	13.44B
Total interactions	25.44B	64.64B
Wall-clock time	1.30 h	11.54 h

Despite using fewer candidate evaluations, XLand is substantially more expensive because each candidate requires a longer and more computationally demanding MTPPO run. Accordingly, the representative XLand ES run takes approximately 11.5 hours, compared with 1.3 hours for GridWorld.

These measurements correspond to the complete configured training procedure, including all ES generations and periodic evaluations of the ES center on the validation set. They should therefore be viewed as conservative estimates of the actual computational cost of training DynATS. In particular, two factors could substantially reduce the required wall-clock time. First, the selected sampler may reach its best validation performance before the final ES generation, in which case training could in principle be terminated earlier. Second, reducing the frequency of center evaluations or the size of the validation set would lower the overall runtime. We retain the full protocol here to report the computational cost of the experiments as actually conducted.

C.6.2 Deployment overhead

Once trained, DynATS adds little computational overhead to MTRL training. Its input features are constructed from statistics already available from the collected PPO data, so deployment requires no additional environment interactions. The sampler overhead consists primarily of feature construction and a forward pass through the frozen Transformer.

To quantify this cost independently of environment and PPO runtime, we measure the latency of the task-sampling update using the Meta-World10 deployment configuration. We use 20 seeds and 10 tasks, matching the tensor shapes used in the full experiment. We exclude JAX compilation through warm-up and report median latency on an NVIDIA H200 on Table 3.

Even though a DynATS update is more computationally expensive than a DRATS update, both costs are negligible relative to environment collection and PPO optimization. For example, our Meta-World10 experiments (20 parallel seeds on the same H200 GPU) took anywhere from $15 - 18$ hours, depending on concurrent CPU and GPU usage.

Table 3: Isolated task-sampler latency for the Meta-World10 deployment configuration. Values are median latency with the interquartile range in brackets. The final column projects the direct sampler computation over the complete 100M-step evaluation protocol of Meta-World10.

Sampler	Latency / decision	Projected total
Uniform	–	0 s
DRATS	1.16 ms [0.84, 1.20]	1.16 s
DynATS	7.14 ms [6.13, 7.33]	7.14 s

D Details on Experimental Results

D.1 Capturing Inter-Task Dynamics

Synthetic Benchmark. We consider $T = 5$ tasks over $N = 100$ sampling decisions. At epoch t, the sampler selects a sampling distribution $q_t \in \Delta^{T-1}$. We define the cumulative normalized allocation to task j as

$$x_j(t) = \frac{T}{N} \sum_{\tau=1}^{t} q_{\tau,j}, \tag{32}$$

such that uniform sampling gives $x_j(N) = 1$ for every task. The return of task i is

$$J_i(t) = \text{clip}\left[1 - \exp\left(-\sum_{j=1}^{T} W_{ij} x_j(t)^{p_j}\right), 0, 1\right]. \tag{33}$$

Here, W_{ij} controls how samples allocated to task j affect task i: $W_{ij} > 0$ represents positive transfer and $W_{ij} < 0$ represents interference. The exponent p_j controls how quickly the effect of sampling task j develops. Consequently, the performance of every task depends on the complete allocation history across the task set.

Each task set corresponds to a different pair (W, p). We sample $W_{ii} \sim U[0.5, 1.2]$ and $p_i \sim U[0.6, 1.8]$. Each structure contains one or two randomly located positive-transfer edges with magnitude $U[0.6, 1.4]$. With probability 0.6, a receiving task is made primarily dependent on its donor by replacing its diagonal entry with a value sampled from $U[0.05, 0.30]$. With probability 0.7, we additionally add a negative-transfer edge with magnitude $-U[0.2, 0.6]$. Finally, the complete matrix is calibrated such that all tasks can jointly reach return 0.75 after 75% of the available allocation budget, a threshold used to ensure that all tasks are learnable.

We generate three fixed datasets from this distribution: 256 task sets for ES training, 32 for validation and checkpoint selection, and 256 for final testing.

Oracle. Because Equation 33 is differentiable with respect to the allocation, we train an oracle that directly optimizes the complete sequence of sampling distributions for each test task set. The oracle has privileged access to (W, p) and maximizes the same AUC objective used to train DynATS, subject to the same minimum task-probability constraint. Its optimization hyperparameters are reported in Appendix E.

Learned Sampler Architectures. We compare three permutation-equivariant learned samplers. All use the input features defined in Appendix B.1 and produce one scalar logit per task.

For the **Linear** sampler, given the task representation h_i,

$$\ell_i = w^\top h_i + u^\top \left(\frac{1}{T} \sum_{j=1}^{T} h_j\right) + b. \tag{34}$$

The **DeepSets** sampler replaces this linear mapping with a nonlinear permutation-equivariant block,

$$z_i = \text{GELU}\left(A h_i + B \frac{1}{T} \sum_{j=1}^{T} h_j\right), \tag{35}$$

followed by layer normalization and a shared scalar output head.

Linear and DeepSets receive the current inputs together with exponential moving averages using decay factors $\{0.5, 0.9, 0.99\}$, providing them with temporal information without attention. The **Transformer** instead receives the complete $H = 100$ history and uses the architecture described in Appendix B.3.

D.2 Multi-Task Sampler

Chain Benchmark. *Chain* consists of four deterministic tasks with four discrete actions. We refer to the two simpler tasks as *Donors* and the two more difficult tasks as *Recipients*. The environment observation consists of a one-hot encoding of the current cue and the normalized position in the chain. The shared PPO policy is additionally conditioned on a one-hot task identifier. The cue-to-action mapping is

$$0 \to 0, \quad 1 \to 1, \quad 2 \to 2,$$
$$3 \to 3, \quad 4 \to 3, \quad 5 \to 1.$$

The two donor tasks have length eight. Donor A alternates cues $(0, 1)$ and Donor B alternates $(2, 3)$. An incorrect action moves the agent back by one position. The recipient tasks have length twelve. Recipient A starts with the same eight-step prefix as Donor A and appends $(4, 2, 4, 2)$, while Recipient B starts with the Donor B prefix and appends $(5, 0, 5, 0)$. Any incorrect action in a recipient resets the chain to its beginning. The shared prefixes therefore create two explicit transfers,

$$\text{Donor A} \to \text{Recipient A}, \qquad \text{Donor B} \to \text{Recipient B}.$$

Rewards are designed such that the recipients are initially difficult to learn directly and normalized to one. Completing a task gives terminal reward 0.75, while the remaining 0.25 is distributed for single-step successes. Every donor position is rewarded (giving $0.25/8$), while for the recipients only the final four positions are rewarded (giving $0.25/4$). Therefore, a randomly initialized policy must execute the complete prefix before receiving any recipient-specific learning signal (which incentivizes learning the donor tasks first).

Reference Curriculum. For diagnosis, we additionally include a hand-specified curriculum. It uses sampling probabilities $(0.48, 0.48, 0.02, 0.02)$ for the two donors and two recipients, respectively, until both donors reach return 0.9. It then switches to $(0.02, 0.02, 0.48, 0.48)$. This curriculum is not used to train DynATS and is included only as a structured reference for this benchmark.

D.3 Same-Domain Generalization

For both domains, DynATS is trained on task sets containing $T = 5$ tasks and evaluated zero-shot on previously unseen task sets with $T \in \{5, 10\}$. ES training draws task-set seeds from $[10^6, 2 \cdot 10^6)$, checkpoint selection uses 32 validation task sets with seeds $0, \ldots, 31$, and final evaluation uses 32 held-out task sets with seeds $10^7, \ldots, 10^7 + 31$.

GridWorld Domain. We adapt a slightly modified version of the procedural *all* GridWorld distribution used by Jackson et al. (2024); Goldie et al. (2024). Each task is independently sampled from the distribution summarized in Table 4.

For each task, 15 wall locations are sampled without replacement from the maximum 10×10 grid. When $g < 10$, only walls falling within the active grid are present. The agent and the K active objects are initialized in different non-wall cells. Collecting an object produces its associated reward and removes it from the grid. A previously collected object may subsequently respawn according to its respawn probability. When object i is collected, the episode may terminate with its sampled termination probability. Otherwise, it continues until the horizon H is reached.

For every sampled task, we compute the exact finite-horizon minimum and maximum expected returns by dynamic programming. These provide the task-specific values $J_i^{\min}$ and J_i^{ref} used for performance normalization. During ES training each task set contains five independent tasks from this distribution. The same frozen sampler is additionally evaluated on task sets containing ten independently sampled tasks.

Table 4: Procedural distribution used to sample GridWorld tasks. U denotes uniform sampling (discrete for integer-valued parameters) and $U_{\log}$ uniform sampling in log-space.

Parameter	Distribution
Grid side length g	$U[4, 10]$
Number of active objects K	$U[1, 5]$
Episode horizon H	$U[20, 199]$
Reward of object 1	$U[0, 1]$
Reward of remaining objects	$U[-1, 1]$
Termination probability	$U_{\log}[10^{-2}, 1]$
Respawn probability	$U_{\log}[10^{-3}, 10^{-1}]$

XLand Domain. We construct a task distribution from XLand-MiniGrid, using `XLand-MiniGrid-R1-9x9`. Rather than generating new rulesets during ES training, we first generate a fixed dataset of 1,000 unique rulesets from which we sample MTRL task sets. Rulesets are generated using the XLand ruleset generator. For every ruleset, the dependency-chain depth is sampled uniformly from $\{0, \ldots, 4\}$. Since our shared MTPPO policy is feed-forward and the inner PPO runs must remain relatively short to make ES training tractable, we exclude tile-relative operations, which were particularly difficult to learn under this setting during preliminary experiments. Consequently, goals require the agent either to hold an object or to satisfy an agent–object spatial relation.

The resulting 1,000 rulesets are split into 700 training, 150 validation, and 150 test rulesets using a fixed seed. Consequently, no ruleset can occur in more than one split. ES training samples task sets only from the training set, checkpoint selection and baseline validation use the validation set, and final evaluation uses only the held-out test set. For a task-set, T rulesets are sampled independently, with replacement, from the corresponding set and instantiated in `XLand-MiniGrid-R1-9x9`. We use $T = 5$ during ES training and evaluate the frozen sampler with $T \in \{5, 10\}$. Episodes have a maximum length of 200 environment steps and use the native sparse XLand reward.

Evaluation Protocol. DynATS checkpoints are selected only using the validation task sets and are then frozen. For the adaptive heuristic baselines, the results reported in the main comparison use task-set-specific hyperparameters: for each held-out task set, baseline hyperparameters are optimized using a gridsearch. The searched hyperparameters are given in Appendix E.

D.4 Cross-Domain Generalization

For the cross-domain experiments, we deploy the DynATS sampler trained on the GridWorld distribution directly on two substantially different continuous-control MTRL benchmarks: MuJoCo6 and Meta-World10. The DynATS parameters remain frozen throughout these experiments. No ES optimization or finetuning is performed on either target benchmark. We compare against uniform sampling and DRATS.

MuJoCo6. MuJoCo6 consists of six continuous-control locomotion tasks: Swimmer, Hopper, HalfCheetah, Walker2d, Ant, and Humanoid. The tasks differ in both observation and action-spaces. We therefore use task-specific actor and critic input layers and task-specific output heads, while sharing the intermediate hidden layers across all six tasks. Each episode has a maximum length of 1,000 environment steps.

For task-sampling normalization, the lower reference J_i^{rand} is estimated from random-policy episodes at the beginning of each run. The upper reference J_i^{ref} is the maximum single-episode return observed for task i so far. Both DRATS and DynATS use these references to construct their normalized performance signals. DynATS then feeds the resulting generic task-level features through the frozen GridWorld-trained Transformer, whereas DRATS applies its normalized-gap prioritization rule.

Meta-World10 and Meta-World50. Meta-World10 and Meta-World50 contain ten and fifty robotic manipulation tasks with common observation and action spaces. We use a task-conditioned

shared actor and critic with task-specific output heads. Episodes have a fixed length of 500 environment steps, and aggregate performance is measured using success rate.

Following DRATS, the upper return reference for every task is initialized to 5000. Once the observed success rate of task i exceeds 0.5, its reference is permanently switched to the maximum single-episode return observed for that task and is subsequently updated as a running maximum. The lower reference J_i^{rand} is estimated from random-policy episodes. As in MuJoCo6, DynATS receives the corresponding normalized generic features and directly predicts the task distribution without target-domain optimization.

D.5 Input Feature Ablations

The input-feature experiments use the same task distributions, Transformer architecture, PPO protocol, ES optimization settings, and training/validation splits as the corresponding full GridWorld and XLand experiments. Every ablated sampler is trained from scratch and differs from the full model only in the features supplied to the Transformer.

We group the seven inputs from Table 1 into: *allocation* (Relative Allocation and Relative Cumulative Allocation), *normalized performance* (Performance and Performance Change), *relative performance* (Relative Performance and Relative Performance Change), and *training proportion*. Each grouped ablation retains only the features in one group. GridWorld results are averaged over three independently trained ES samplers, whereas XLand uses one ES run due to its substantially greater computational cost. Checkpoint selection uses the same held-out validation sets as for the corresponding full model.

D.6 Baseline Sampling Strategies

Uniform sampling keeps

$$q_{n,i} = \frac{1}{T} \tag{36}$$

for every task throughout training. The three adaptive baselines considered in this work (DRATS, LPR and LP) instead differ only in the task-level priority signal $g_{n,i}$ they construct. Given this signal, all three update the sampling distribution using the same exponentiated-gradient mirror-ascent rule.

Specifically, let $u_i = 1/T$ denote the uniform distribution. At sampling step n, the adaptive baselines optimize the regularized objective

$$\mathcal{L}(q_n) = \sum_{i=1}^{T} q_{n,i} g_{n,i} - \frac{1}{\eta} D_{\text{KL}}(q_n \,\|\, u), \tag{37}$$

where $\eta > 0$ controls the strength of the regularization toward uniform sampling.

We perform one exponentiated-gradient step on this objective,

$$\tilde{q}_{n+1,i} \propto q_{n,i} \exp\left[\alpha \left(g_{n,i} - \frac{1}{\eta} \log \frac{q_{n,i}}{u_i}\right)\right], \tag{38}$$

followed by normalization across tasks. Here, α is the mirror-ascent step size. In all experiments we set $\alpha = \eta/2$ and tune η using the search spaces reported in Appendix E.

Finally, we enforce a minimum sampling probability m by mixing the resulting distribution with the uniform distribution,

$$q_{n+1,i} = m + (1 - Tm)\tilde{q}_{n+1,i}. \tag{39}$$

This guarantees $q_{n+1,i} \geq m$ for every task and prevents permanent task starvation.

The adaptive baselines differ only in their priority signals. DRATS uses the normalized performance gap

$$g_{n,i}^{\text{DRATS}} = \max\left(\frac{J_i^{\text{ref}} - \hat{J}_{n,i}}{\left|J_i^{\text{ref}} - J_i^{\text{rand}}\right| + \epsilon}, 0\right), \tag{40}$$

and therefore assigns larger priority to tasks whose current performance is far from its reference value. LPR uses

$$g_{n,i}^{\text{LPR}} = \left|\hat{J}_{n,i} - \hat{J}_{n-1,i}\right|, \tag{41}$$

prioritizing tasks whose performance is currently changing most rapidly. Learning Potential instead uses

$$g_{n,i}^{\mathrm{LP}} = \mathbb{E}\left[\left|\hat{A}_t\right| \mid i\right], \tag{42}$$

the mean absolute GAE magnitude for transitions from task i, as a proxy for the remaining policy-learning signal.

E HYPERPARAMETERS

Unless otherwise stated, all learned samplers use the seven features in Table 1. The Transformer uses one task-attention block followed by one temporal-attention block. Each block uses pre-layer normalization and a feed-forward layer of width $2d$ with GELU activation. Temporal positions use fixed sinusoidal encodings. The output includes a learned scalar temperature initialized to one and constrained to $[0.1, 10]$.

E.1 DYNATS AND ES TRAINING

Table 5 summarizes the sampler architecture and Evolution Strategies hyperparameters used in each experiment.

Table 5: DynATS architecture and ES hyperparameters. "Task" denotes antithetic ES in which each perturbation pair receives an independently sampled task set; "Seed" denotes a fixed task set with independently sampled PPO randomness for each perturbation pair.

	Synthetic	Chain	GridWorld	XLand
Sampler architecture				
Number of tasks during training	5	4	5	5
History length H	100	60	50	100
Embedding dimension d	16	16	16	16
Attention heads	4	4	4	4
Attention order	Task→Temp.	Task→Temp.	Task→Temp.	Task→Temp.
Minimum task probability	0.05	0.02	0.05	0.05
ES optimization				
Antithetic randomization	Task	Seed	Task	Task
ES generations	1000	250	250	200
ES population	512	512	512	256
Perturbation std. σ	0.03	0.03	0.03	0.03
σ decay	0.999	0.999	0.999	0.999
ES optimizer	Adam	Adam	Adam	Adam
ES learning rate	0.01	0.01	0.01	0.01
ES learning-rate decay	1.0	0.999	0.999	0.999
Validation interval	5	5	10	10
Independent ES runs	3	3	3	1

For each antithetic pair, the positive and negative perturbations are evaluated under the same task-set and PPO randomness, and the pairwise winner/loser fitness is used to construct the ES update. Sampler fitness is the AUC of mean normalized task performance.

Synthetic validation uses a fixed set of 32 validation task sets. Chain validation evaluates the ES center using 64 PPO random seeds. GridWorld uses 32 held-out validation task sets with 50 PPO seeds per set, while XLand uses 32 held-out validation task sets with 20 PPO seeds per set.

E.2 INNER-LOOP MTPPO

Table 6 reports the MTPPO configuration used when evaluating candidate samplers during ES training.

E.3 EVALUATION AND BASELINE TUNING

Table 7 summarizes the dataset splits and hyperparameter search spaces used for the adaptive sampling baselines DRATS, LP, and LPR.

For Chain, baseline hyperparameters are selected using 20 PPO seeds, while final results use 100 held-out seeds. For GridWorld and XLand, final evaluation uses 100 PPO seeds.

For all adaptive baselines (DRATS, LP, and LPR), we set $\alpha = \eta/2$.

Table 6: Inner-loop MTPPO hyperparameters used during DynATS training.

	Chain	GridWorld	XLand
Training protocol			
MTPPO epochs	60	50	100
Rollout length	24	200	200
Rollouts per PPO epoch	16	15	50
Task conditioning	Yes	Yes	Yes
Policy head	Shared	Shared	Shared
Backbone	MLP	MLP	CNN
Hidden dimensions	$128, 128$	$128, 128$	$128, 128$
CNN channels	–	–	$16, 32$
Activation	ReLU	ReLU	ReLU
PPO optimization			
PPO learning rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
Learning-rate annealing	No	No	No
Discount γ	0.99	0.99	0.99
GAE λ	0.95	0.95	0.95
PPO clip	0.2	0.2	0.2
Entropy coefficient	0.01	0.01	0.01
Value coefficient	0.5	0.5	0.5
Max. gradient norm	0.5	0.5	0.5
PPO update epochs	4	4	4
Minibatches	10	10	10
Advantage normalization	Global	Per-task	Global
Sampler-fitness evaluation			
Fitness evaluation interval	1	1	5
Evaluation episodes / task	1	5	5
Evaluation policy	Greedy	Stochastic	Stochastic
Performance references	$[0, 1]$	Exact DP	$[0, 1]$

Table 7: Task-set splits and hyperparameter tuning of the adaptive sampling baselines DRATS, LP, and LPR.

	Synthetic	Chain	GridWorld	XLand
Training task sets	256	1	Procedural	700-rule dataset
Validation task sets	32	1	32	150-rule dataset / 32 sets
Test task sets	256	1	32	150-rule dataset / 32 sets
Tasks per task set T	5	4	5, 10	5, 10
Baseline η search grid	$\{1, 2, 4, 8, 16, 32\}$	$\{1, 2, 4, 8, 16\}$	$\{2, 4, 8, 16\}$	$\{2, 4, 8\}$

Synthetic Oracle. The dynamic oracle is optimized independently for every synthetic test structure using Adam for 500 optimization steps with learning rate 0.05. Its allocation is subject to the same minimum task probability of 0.05 used by the learned samplers.

E.4 Cross-Domain Evaluation

Table 8 reports the PPO and sampling hyperparameters used when deploying the GridWorld-trained DynATS sampler on MuJoCo6 and Meta-World10/50.

Table 8: PPO and task-sampling hyperparameters for cross-domain evaluation.

	MuJoCo6	Meta-World10	Meta-World50
Training protocol			
Tasks	6	10	50
Environment steps / seed	100M	100M	100M
Episode horizon	1000	500	500
Target transitions / PPO update	384,000	100,000	500,000
Independent seeds	10	20	10
Policy architecture			
Policy hidden layers	3×256	3×400	3×400
Activation	Tanh	ReLU	ReLU
Task-specific output heads	Yes	Yes	Yes
Task-specific input layers	Yes	No	No
Per-task observation normalization	Yes	No	No
Per-task value normalization	Yes	No	No
Per-task advantage normalization	Yes	Yes	Yes
PPO optimization			
PPO learning rate	10^{-4}	5×10^{-4}	5×10^{-4}
Discount γ	0.99	0.99	0.99
GAE λ	0.95	0.95	0.95
PPO clip	0.2	0.2	0.2
Entropy coefficient	0.01	0.01	0.03
Value coefficient	0.5	0.001	0.001
Max. gradient norm	0.5	1.0	1.0
PPO update epochs	16	16	16
Minibatches	32	32	32
Adam ϵ	10^{-5}	10^{-5}	10^{-5}
Task sampling			
DRATS η	4	8	3
DRATS α	2	4	1.5
Minimum task probability	$1/24$	0.025	0.005
Random-policy episodes / task	32	1000	1000