MODEL-BASED META-LEARNING FOR ALGORITHM DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement Learning (RL) algorithms are typically hand-crafted through a slow and iterative scientific process. While meta-RL promises to automate RL algorithm discovery, high simulation costs have limited research. In this work, we introduce *Model-Based Meta-Learning* (MBML), a novel approach that uses *learned world models* as an efficient alternative to environment simulation. Policies trained in these world models fail to transfer to the simulator, but algorithms discovered in world models transfer successfully. We show that MBML matches the performance of meta-RL with environment simulators at a fraction of the time and compute. By substantially reducing the computational cost of meta-training, MBML lowers the barrier to entry for meta-RL research while enabling algorithm discovery at a larger scale. We demonstrate this by using MBML to meta-learn a novel algorithm that significantly outperforms prior work on unseen continuous control environments.

1 INTRODUCTION

Manually developing new machine learning algorithms is difficult and time-consuming. Meta-learning has emerged as one potential solution towards automating this process, by "learning to learn" (Beck et al., 2023; Clune, 2020; Finn et al., 2017; Schmidhuber, 1987). This aligns with the dominant paradigm in machine learning: scaling computation and data ultimately outperforms methods informed by human knowledge and intuition (Sutton, 2019). The approach of using meta-learning to automate the discovery of new algorithms has shown great promise for the wider field of machine learning (Chen et al., 2023; Metz et al., 2022) and initial signs of life for reinforcement learning (RL) (Beck et al., 2025; Houthooft et al., 2018; Lu et al., 2022; Oh et al., 2021; 2025).

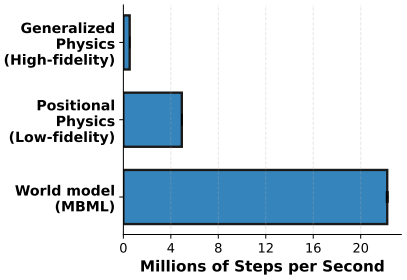


Figure 1: MBML reduces the cost of meta-RL by up to $42\times$ in our experiments, without reducing performance.

In RL, this problem of discovering new algorithms or update rules is called *many-shot meta-RL* (Beck et al., 2025). Prior works have replaced typical RL algorithm components with meta-learned neural networks, including the loss function (Houthooft et al., 2018; Jackson et al., 2024; Lu et al., 2022), optimizer (Goldie et al., 2025a), or the entire gradient update mechanism (Oh et al., 2021; 2025).

However, these meta-RL approaches require training thousands of standard RL policies from scratch. These policies are discarded, and only the learned update rule is transferred and used to train new policies from scratch. Despite efforts to parallelize training to reduce wall-clock meta-training time, simulating thousands of environments remains a significant computational bottleneck (Oh et al., 2025). This limitation slows scientific iteration and constrains the complexity and diversity of environments used in meta-training. As a result, prior work evaluates meta-learning algorithms on a restricted set of simple environments, and on a single seed for the meta-training run (meta-seed, Goldie et al., 2025a), limiting generalization claims (Jackson et al., 2024; Lu et al., 2022). Indeed, computational cost has been identified as the primary hurdle for many-shot meta-RL research (Beck et al., 2025).

In this work, we first address this computational limitation of prior work by introducing *Model-Based Meta-Learning* (MBML), a framework that replaces the environment *backend*—the component responsible for environment execution, such as a physics simulator—with learned *world models*.

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.

As modern GPUs are highly optimized for parallel neural network inference (Lee et al., 2025), this approach yields substantial speed-ups during meta-training. These world models are trained at a negligible fraction of the computational cost of meta-training and do not need to be perfectly accurate. Specifically, we find that while policies trained in these world models do not transfer to a physics simulator, algorithms meta-learned with the world model do.

We first demonstrate MBML's flexibility and validity by using world models as a backend for four existing meta-RL algorithms: LPO (Lu et al., 2022), TA-LPO (Jackson et al., 2024), OPEN (Goldie et al., 2025a), and LPG (Oh et al., 2021). Algorithms discovered with the world model backend achieve similar performance to those discovered using a physics-based backend (Figure 4), but exhibit a substantial speedup in wall-clock time, from $2.9\times$ to $42\times$ in our experiments (Figure 3).

As a case study, we design a new meta-learning algorithm, using a world model backend trained on D4RL datasets (Fu et al., 2021) and building on prior methods (Goldie et al., 2025a; Jackson et al., 2024). The result is a novel algorithm that combines a meta-**L**earned **O**ptimizer and **L**oss function (LOL). Our experiments show that LOL significantly outperforms existing RL algorithm-discovery methods.

Our contributions are:

1. We propose MBML, a novel **meta-learning framework** that substantially reduces wall-clock meta-training time without sacrificing performance on unseen environments (Section 7.1).
2. We demonstrate that **accurate world models are not needed** to discover performant algorithms (Section 7.2).
3. We use MBML to **design a new meta-learning algorithm** that extends prior work in meta-RL, and show that it significantly outperforms previous meta-learning algorithms (Section 8), demonstrating that MBML accelerates research.

2 Related Work

Model-Based RL An RL method is model-based if the agent learns a predictive model of the true environment (Sutton & Barto, 2018), traditionally used for planning to improve the online policy more sample-efficiently. In the alternative paradigm, model-free RL, an agent trains its policy without explicitly learning the environment's dynamics. Ha & Schmidhuber (2018) extend model use to learning both state representations *and* transition functions. These models are commonly called *world models*, and are used to train policies in "imagination" without ever interacting with the original environment. The Dreamer line of work (Hafner et al., 2023) further refines this concept using recurrent state-space models, achieving competitive results across both continuous control and discrete settings. More recently, diffusion models (Ho et al., 2020) have been applied to world modeling (Alonso et al., 2024; Hafner et al., 2025), where model accuracy scales with the number of denoising steps, offering a natural trade-off between fidelity and compute cost. While we take inspiration from these works by training policies for meta-learning entirely in world models, we find that our meta-training framework benefits from world-model sample efficiency without the strict accuracy requirements needed for policy transfer.

Many-shot Meta-RL (Beck et al., 2025) involves learning general RL update rules that apply across a broad set of environments. We contrast this with *Few-shot Meta-RL* such as Duan et al. (2016) or Team et al. (2023), where the meta-learning algorithm adapts a policy online to a new environment. Lu et al. (2022) introduces a theoretically bounded RL loss function that is meta-learned using Evolutionary Strategies (Salimans et al., 2017, ES). Jackson et al. (2024) augment this learned loss with information about the training progression, improving the learned algorithm's performance. This insight is then leveraged in Goldie et al. (2025a), where the authors design a training-aware, bespoke RL optimizer instead of a loss function, meta-learned with ES. Instead of using ES, a separate line of work, Oh et al. (2021), uses meta-gradients for meta-training a *Learned Policy Gradient* (LPG) algorithm; Follow-up work has scaled this meta-training to a large distribution of environments (Oh et al., 2025). We are the first work to directly compare these meta-learning algorithms under the same evaluation protocol. Importantly, we distinguish between *meta-training*, *meta-validation*, and *meta-test* environments, which prior work did not consistently do.

3 BACKGROUND

3.1 REINFORCEMENT LEARNING

We consider a *Markov Decision Process* (MDP) $\mathcal{M} = (\mathcal{S}_0, \mathcal{S}, \mathcal{A}, T, R)$. $\mathcal{S}_0$ is the initial state distribution. $\mathcal{S}$ and $\mathcal{A}$ are the state space and the action space, respectively. The transition function is $T : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ and the reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \to \mathbb{R}$. In RL, we are interested in finding a policy π that maximizes the expected return in the MDP $\mathcal{M}$

$$J(\pi; \mathcal{M}) = \mathbb{E}_{a_t \sim \pi, s_0 \sim \mathcal{S}_0, s_{t+1} \sim T(s_t, a_t)} \left[\sum_{t=0}^{H} R(s_t, a_t) \right], \tag{1}$$

where H is the trajectory length or horizon. We define an *algorithm* as a function $\mathcal{F} : \mathcal{M} \mapsto \Pi$ that maps from an MDP to a distribution of policies Π, with the objective of maximizing the expected $J(\pi; \mathcal{M})$ where $\pi \sim \Pi$. The algorithm may be parameterized by hyper/meta-parameters[1] ϕ, in which case we denote it by $\mathcal{F}_\phi$.

Model-Based RL In model-based RL, the algorithm has access to a parameterized approximation of an MDP $\mathcal{M}$, denoted as $\widehat{\mathcal{M}}_\theta = \{\mathcal{S}_0, \mathcal{S}, \mathcal{A}, \widehat{T}_\theta, \widehat{R}_\theta, H\}$, where $\widehat{T}_\theta$ and $\widehat{R}_\theta$ are parameterized by θ. This is typically trained with supervised learning and is commonly called a *world model* (Ha & Schmidhuber, 2018). World models are often imperfect representations of the true environment, and policies trained in world models generally transfer poorly due to the distribution shift (Jackson et al., 2025), in a problem widely known as the *Sim-to-Real Gap* (Da et al., 2025).

3.2 META-OPTIMIZATION

Meta-optimization can be challenging, as meta-loss landscapes exhibit fractal-like structure and optimization dynamics can be chaotic (Metz, 2021; Sohl-Dickstein, 2024). Many-shot meta-RL uses two common meta-optimization approaches.

Meta-Gradients One approach is to directly backpropagate gradients through the training process using *truncated backpropagation through time* (Bengio et al., 1994; Pascanu et al., 2013; Werbos, 1990, TBPTT). For example, Oh et al. (2021; 2025) backpropagate through the policy and critic gradient-update process. However, others suggest this is unstable and difficult to tune (Jackson et al., 2023; Metz, 2021); our results corroborate this.

Evolution Strategies (ES) An alternative approach uses black-box optimization, such as evolutionary algorithms like ES (Goldie et al., 2025b). These methods evaluate populations of functions to estimate a gradient or search direction (Li et al., 2020). OpenES (Salimans et al., 2017) is commonly used for meta-optimization in many-shot meta-RL (Goldie et al., 2025a; Houthooft et al., 2018; Jackson et al., 2024; Lu et al., 2022), and shows better resilience to local minima in this setting (Metz, 2021). Gradient estimates improve with larger population sizes and can be fed to standard optimizers such as Adam (Kingma & Ba, 2017).

3.3 PRIOR MANY-SHOT META-RL ALGORITHMS

LPO Lu et al. (2022) design *Learned Policy Optimization* (LPO), a meta-learning algorithm that learns a loss function parameterized by a small feedforward network. This network obeys the constraints of *drift functions* laid out by Mirror Learning (Kuba et al., 2024), giving LPO theoretical convergence guarantees. The drift function of LPO is initialized as that of Proximal Policy Optimization (Schulman et al., 2017, PPO); it is meta-optimized with OpenES (Salimans et al., 2017).

TA-LPO Jackson et al. (2024) builds on LPO, introducing *Temporally-Aware LPO* (TA-LPO), which additionally conditions the policy on the training progress. This enables the agent to adapt its behavior, for example, by reducing the policy's entropy over time.

[1] As noted in Beck et al. (2025), many-shot meta-RL often does not distinguish between meta-parameters and hyperparameters.

OPEN Goldie et al. (2025a) meta-learns OPEN, an optimizer for RL with OpenES (Salimans et al., 2017) as an alternative to conventional optimizers like Adam (Kingma & Ba, 2017). OPEN is conditioned on gradients and momentum from different moving-average timescales (Kingma & Ba, 2017), neuron dormancy (Sokar et al., 2023), and other statistics.

LPG Oh et al. (2021) is a meta-learned RL update rule for discrete action spaces, parameterized as a backward-LSTM (Hochreiter & Schmidhuber, 1997) and optimized via meta-gradients (Xu et al., 2018). It takes as input the agent's action probabilities, auxiliary prediction vector, and observed reward, and outputs targets used to update the agent's policy.

4 Terminology and Problem Setting

We adapt terminology from Goldie et al. (2025b), and give a quick overview here:

- *Meta-learned algorithm*: a learned algorithm that implements RL training components.
- *Meta-learning algorithm* (or meta-RL algo.): a method to learn a meta-learned algorithm.
- *Meta-training*: training a meta-learned algorithm using a meta-learning algorithm.
- *Meta-validation*: After meta-training, the evaluation of several meta-learned algorithms on one or more held-out meta-validation environments.
- *Meta-testing*: Evaluation of the meta-learned algorithm that was selected after meta-validation on one or more held-out meta-test environments.

In single-task many-shot meta-RL, the objective is to find a meta-learned algorithm that transfers to different environments (Beck et al., 2025). We differentiate between the *environment* that defines a specific task and its *backend*, which implements the execution of the environment dynamics. For example, the backend could be a specific physics simulator, game engine, or numerical ODE solver. This distinction gives rise to two axes of transfer. *Environment transfer* asks whether an algorithm discovered on one task transfers to another, whereas *backend transfer* asks whether an algorithm discovered using one implementation of the dynamics transfers to another.

With MBML, we investigate if we can discover new RL algorithms when replacing a simulator backend with a learned world model backend. In our Brax experiments, we consider both axes simultaneously. We compare meta-training in the *positional* physics simulator and in the world model (trained on D4RL data), and meta-evaluate them with the *generalized* simulator backend, which has higher fidelity than positional.

5 Method

MBML requires training a world model and using it as the environment backend, which integrates straightforwardly with any meta-RL method. In this work, we run experiments with ES-based meta-RL, and give more detail on this specific instantiation of MBML (alternatively, we apply MBML to a meta-gradient method in Section C.4). Overall, the resulting meta-RL algorithm (Figure 2) has three main steps:

1. **Train a world model** (Figure 2, left; Section 5.1) with supervised learning inspired by model-based offline RL methods (Kidambi et al., 2021; Yu et al., 2020).

2a. **Train a large batch of policies** (Figure 2, center) in the environment with the world model backend (i.e., the world model itself), using perturbed meta-parameters.

2b. **Evaluate policies and update meta-parameters** (Figure 2, right and bottom; Section 5.2) based on ES gradient estimates to improve algorithms towards training policies with a higher final performance. We repeat from 2a.

3. **Evaluate the resulting learned algorithms** in a meta-validation environment to pick a checkpoint, then use this checkpoint on an unseen environment (meta-test).

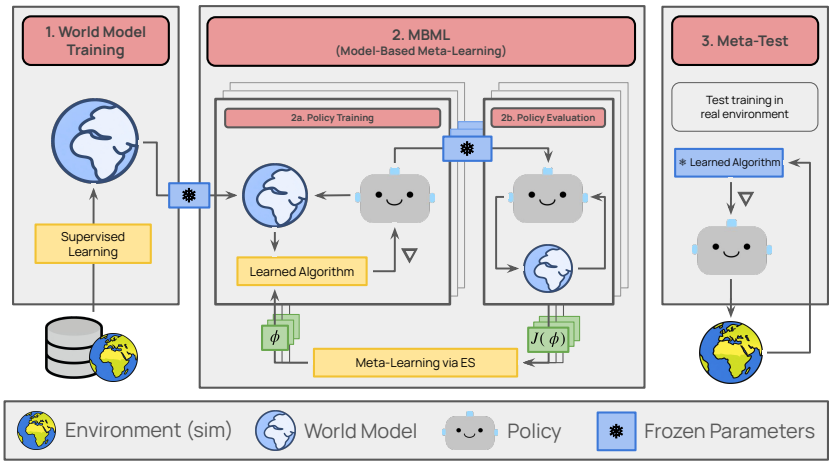


Figure 2: Our Model-Based Meta-Learning framework efficiently discovers novel algorithms. Using world models (1), we train many policies in parallel (2a), then evaluate them in the world model (2b) for meta-learning. By replacing physics simulation with hardware-efficient world model inference, this approach yields significant speedups. We evaluate meta-learning algorithms using an expensive high-fidelity backend in the meta-test setting (3).

5.1 Training World Models

As in Chua et al. (2018); Jackson et al. (2025); Kidambi et al. (2021); Yu et al. (2020), we train an ensemble of N dense networks f_θ^i with $i \in \{1, \ldots, N\}$ that each predict a mean and standard deviation of the next state and reward given the previous state and action,

$$f_\theta^i(s_t, a_t) = [\mu_{s_{t+1}}^i, \sigma_{s_{t+1}}^i, \mu_{R_t}^i \sigma_{R_t}^i].$$

Outputting a distribution enables the world model to estimate aleatoric uncertainty, while ensemble variance estimates epistemic uncertainty (Chua et al., 2018).

To sample from an ensemble-world model, we first uniformly sample a network index $i \sim \mathcal{U}(1, \ldots, N)$, and then sample the next state and reward from a Gaussian via

$$s_{t+1} \sim \mathcal{N}\left(\mu_{s_{t+1}}^i, \left(\sigma_{s_{t+1}}^i\right)^2\right), \quad R_t \sim \mathcal{N}\left(\mu_{R_t}^i, \left(\sigma_{R_t}^i\right)^2\right).$$

Although this parameterization is specific to continuous states and actions, MBML is agnostic to world-model implementations; we demonstrate this by using MBML with pre-trained Atari world models in Section 7.1.

As is common in implementations of these ensemble models, we use the same initial state distribution and termination function as the ground-truth environment (Jackson et al., 2025; Sun, 2023). In addition, we adopt the ensemble-disagreement termination criterion of (Kidambi et al., 2021, details in Section A.1.1). We adopt these design choices because they are well-established and widely validated in prior work (Jackson et al., 2025), allowing us to isolate the effects of meta-learning within world models without introducing confounding factors from alternative model design or rollout heuristics. MBML is robust to ablating these design choices, as shown in Section C.9.

5.2 Meta-learning RL algorithms

Policy training We train policies in the world model using the parameterized algorithm we are meta-learning. Before policy training, we create a batch of perturbations for the meta-parameters ϕ by adding Gaussian noise $\phi + \epsilon\sigma$ with $\sigma \in \mathbb{R}$ and $\epsilon \sim \mathcal{N}(0, I)$. We then train a policy using each parametrized algorithm. We give additional details about training world models in Section A.1.2.

Policy evaluation and meta-parameter update We evaluate each policy's performance $J(\phi + \epsilon\sigma)$ by repeatedly rolling it out in the world model. We write $J(\phi) = J(\mathcal{F}_\phi(\widehat{\mathcal{M}}); \widehat{\mathcal{M}})$. The gradient

$\nabla_\phi J(\phi)$ is then estimated using

$$\nabla_\phi J(\phi) = \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)} \left[\frac{\epsilon}{\sigma} J(\phi + \sigma\epsilon) \right], \tag{2}$$

as in OpenES (Salimans et al., 2017). We feed these gradient estimates to a standard optimizer, `adam` with weight decay (Loshchilov & Hutter, 2019), to update ϕ. We continue alternating between training a batch of policies and updating ϕ for a set number of iterations.

5.3 Testing the meta-learned RL algorithm

After meta-training, we freeze the meta-parameters of the meta-learned algorithms. For meta-validation, we switch the backend while keeping the environment fixed, and for meta-testing, we switch both the backend and the environment. For example, we might meta-train five algorithms on a HalfCheetah world model, select the one that performs best on HalfCheetah with a physics simulator, and then meta-test the selected algorithm on Hopper with a physics simulator.

6 Experimental Setup

Datasets We use datasets from D4RL (Fu et al., 2021), which the offline RL community commonly uses (Jackson et al., 2025). We do this to ensure reproducibility and avoid additional confounding factors from data collection and tuning. We evaluate online training performance in Brax (Freeman et al., 2021), a hardware-accelerated environment that allows us to directly compare MBML performance with meta-training on physics simulators. As such, we focus on three D4RL locomotion datasets collected from environments available in Brax: *halfcheetah-medium-expert*, *hopper-medium-expert*, and *walker2d-medium-expert*. We use the *medium-expert* datasets, which combine data from two behavioral policies and therefore provide broader state-space coverage.

Table 1: Summary of meta-training and meta-test setup.

Component	Description
Environments	Halfcheetah, Walker2d, Hopper
Meta-training algorithms	LPO, TA-LPO, OPEN
Meta-training backend	World model (MBML) Generalized (Brax, high-fidelity) Positional (Brax, low-fidelity)
Meta-test	Full Brax suite (Generalized)
# Seeds	Meta-training runs: 5 Policy training runs: 10 Policy evaluations per run: 128
Stop criterion	100 generations **OR** 24h wall-clock time

Comparing Environment Implementations Brax has multiple physics backends; we compare our world model backend with two of them. First, we evaluate the *generalized* backend, a reimplementation of the original MuJoCo simulator that provides high fidelity at the cost of slower simulation speeds. Second, we test the *positional* backend, which improves simulation throughput at the cost of fidelity. Goldie et al. (2025a); Jackson et al. (2024); Lu et al. (2022) previously used it for meta-training. More information about Brax physics backends can be found in Section B.2.

Meta-learning Algorithms We meta-train LPO (Lu et al., 2022), TA-LPO (Jackson et al., 2024), and OPEN (Goldie et al., 2025a) on three different environments, using both physics backends provided by Brax and a world model backend for MBML. We test all meta-learned algorithms using the generalized backend. The full meta-training experimental setup is summarized in Table 1. The full hyperparameter details are in Section B.3.

7 Results of MBML

We demonstrate two core results:

1. MBML serves as a computationally efficient alternative to traditional meta-RL, without sacrificing performance relative to accelerated simulators (Section 7.1).
2. MBML remains effective even when using imperfect world models (Section 7.2).

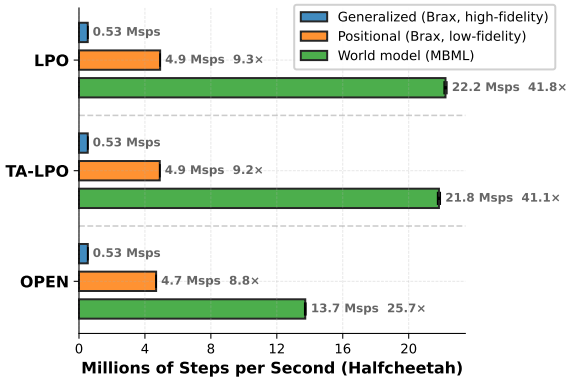


Figure 3: MBML is *up to 42 times faster* than comparable online meta-learning methods, achieving up to 22.3 million steps per second during meta-training on *halfcheetah*. Mean and SEM over 5 seeds. Further details and speedups for Hopper are in Section B.2.

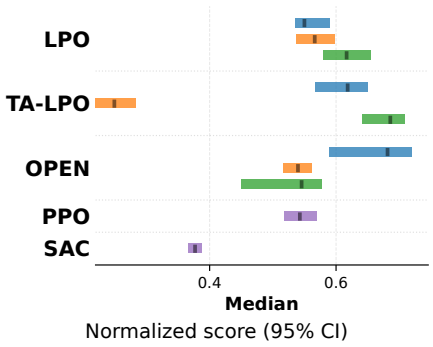


Figure 4: MBML enables discovery of algorithms that perform within error of, or better than, those meta-trained with a physics environment. Each algorithm is evaluated with 10 seeds on the full Brax suite with the generalized backend.

7.1 MBML IS EFFICIENT AND EFFECTIVE

Runtime efficiency Figure 3 demonstrates that, in our experiments, MBML obtains between $25\times$ and $42\times$ speed-up when compared to meta-training on the generalized (high-fidelity) backend directly. We highlight that MBML's speed-up relative to positional (low-fidelity) is $2.9\times$ to $4.5\times$. Training the world model itself is approximately an order of magnitude faster in wall-clock time than meta-training. For example, training a world model for halfcheetah on an H200 takes 11m, while the subsequent meta-training of LPO with MBML takes 2h. Additionally, we reused the same three world models for all meta-training, making world-model training a negligible upfront cost. In combination with the performance of MBML-trained algorithms, these results suggest that MBML is a faster alternative to meta-training with online simulators. We demonstrate this by running meta-training on consumer-grade hardware in Section C.1.

Meta-test Performance We select the best-performing meta-seed for each algorithm and backend for the *halfcheetah* environment (the meta-validation environment) and evaluate these algorithms on the full Brax suite, using the generalized backend. In Figure 4, we see that algorithms meta-trained with MBML (green) are on aggregate within error of meta-training on a generalized backend (blue) or the positional backend (orange). We compare meta-learned algorithms to PPO and SAC baselines with hyperparameters from cleanRL (Huang et al., 2021, see Section B.3.2); both have been trained for the same number of steps we use for one training run of a learned algorithm. We observe that within 100 generations, learned algorithms do not consistently outperform PPO. This does not contradict prior literature's claims of improvements over PPO, where authors usually meta-train and meta-test with the same backend (usually positional), and generally for longer. We therefore posit that the meta-RL algorithms we investigate have limited generalization capacity, as Goldie et al. (2025b) corroborates. Additionally, we note that both TA-LPO meta-trained on positional seem unstable. We show meta-validation and meta-training performance in Section C.2. We analyze the properties of the algorithms in Section C.10.

Meta-training with Atari world models To show broad applicability, we additionally meta-train with diffusion world models from (Alonso et al., 2024), where we demonstrate similar speedups. We also replicate the finding from continuous environments that different backends do not significantly alter meta-validation performance. The results are in Section C.3.

Meta-training with LPG To show that MBML also applies to meta-gradient meta-RL, we meta-train with LPG on both the Atari world models and the continuous control environments. For continuous control, we discretize the action space as LPG does not support continuous action spaces. We obtain up to $49\times$ more meta-gradient steps per second than with a physics simulator. Notably, LPG does not learn a valid update rule within the allocated compute limit for the large majority of environments, neither with MBML nor the environment simulator. The results are in Section C.4.

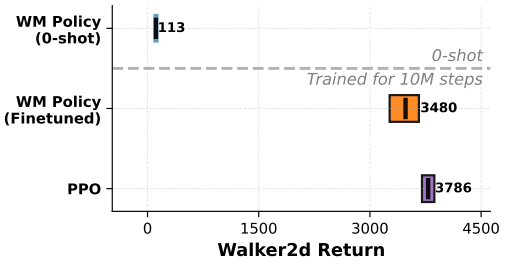


Figure 5: World models do not perform well for policy transfer. Training a policy in the world model and then fine-tuning performs worse than training with PPO from scratch. A more comprehensive version of this plot is available in Section C.7.

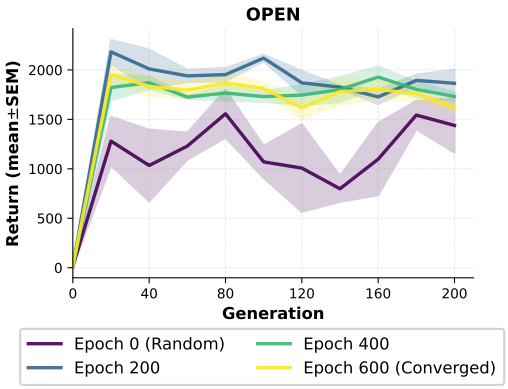


Figure 6: Meta-validation performance for different Hopper world model checkpoints with OPEN. Meta-training on random networks still yields a valid update rule in this case.

7.2 MBML works with imperfect world models

Figure 5 shows that policies trained by PPO in the world model do not transfer to the generalized backend. Even after significant fine-tuning, they underperform compared to a PPO policy trained directly in the generalized backend. This is expected: our world model training is inspired by common offline RL methods (Kidambi et al., 2021; Yu et al., 2020) that use world-model rollouts for data augmentation instead of environment replacement. It is therefore surprising that such world models induce the discovery of strong learning algorithms.

In Figure 6, we meta-train with world model checkpoints at different stages of world model training. We see that increasing the world model training time does not significantly affect meta-validation performance. Surprisingly, meta-training on randomly initialized networks still improves performance on the meta-validation environment for OPEN. We could not replicate this phenomenon with other meta-learning algorithms, suggesting behavior specific to OPEN that is beyond the scope of this work (details in Section C.6).

To be successful, an algorithm learns to exploit the high-level structure of the transition and reward functions to maximize return. Continuous control environments have dense rewards and smoothly changing observations, which is analogous to the structure of our randomly initialized world model with our parameterization. We hypothesize that this structural similarity enables meta-learning transferable algorithms with inaccurate (or even random) transitions, while low-level accuracy is superfluous. Overall, we find that **while policies do not transfer from the world model to the simulator, meta-learned algorithms do**.

8 Case Study: Accelerating Research With MBML

Designing a new meta-learning method So far, we have used MBML to reproduce existing meta-learning methods at lower computational cost. We now ask whether we can reinvest these savings in a richer algorithm search space. Specifically, we jointly meta-learn the policy loss, optimizer, and several hyperparameters, combining and extending components from prior work. Specifically, we use MBML for ablations to iteratively build this meta-learning algorithm. We call the resulting algorithm Learned Optimizer & Loss (LOL). We show that LOL's loss function satisfies the convergence properties from Mirror Learning (Kuba et al., 2024) in Section A.2. We see in Figure 7 that LOL significantly outperforms PPO, and the best algorithm in Figure 4 (TA-LPO meta-trained with MBML). We also compare against a surprisingly strong baseline: PPO-MBML, the same PPO implementation but with hyperparameters tuned in the world model for the same number of meta-train steps as other methods. We analyze LOL features and ablate specific components to show that each component improves performance Section C.10.

Additionally, Figure 8 identifies an emergent training phenomenon due to the particularities of our meta-training setup, which we call "*crunchtime*". Combining a training-aware optimizer and loss with policy evaluation on the last training checkpoint causes policy entropy, which is linked to exploration, to drop rapidly near the end of policy training, making the policy more exploitative right before final evaluation. This is sensible, as overly stochastic policies tend to perform worse in continuous control with deterministic environments (Andrychowicz et al., 2020). We provide more details in Section C.10; this also helps explain LOL's strong performance across continuous control environments.

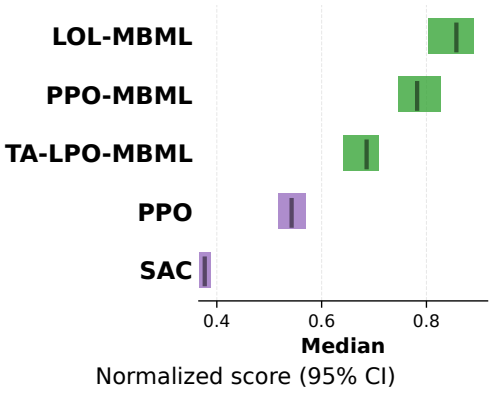


Figure 7: LOL meta-trained with MBML outperforms all other learned algorithms on the Brax suite. PPO-MBML also outperforms the best meta-learned algorithm baseline except LOL.

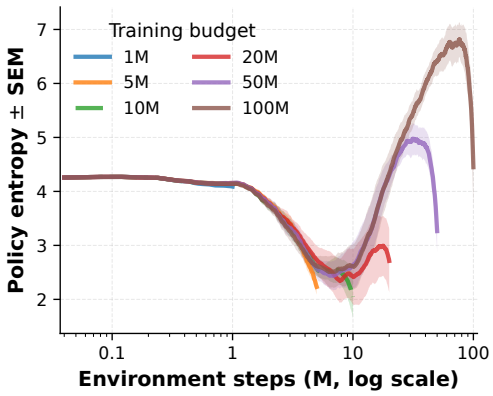


Figure 8: LOL experiences "crunchtime": its policy entropy rapidly decreases as it nears the end of training and prepares for final evaluation. Mean policy entropy over 5 seeds of Hopper (Generalized) policy training with LOL.

9 Limitations

In this work, we focus on meta-learning algorithm evaluation, but the number of axes of variation to explore is very large, including world model architecture, training data, initialization, and termination design, and while we made an effort to ablate the latter, it is as yet unknown which properties matter for the former. This is especially relevant when applying MBML to domains with discrete observation spaces, which may require different world-model architectures and training procedures. Our strongest algorithm-discovery results are within continuous-control environments. As shown in Section C.11, algorithms meta-trained on these environments do not consistently generalize to semantically different domains, indicating that MBML reduces the cost of algorithm discovery but does not by itself solve broad algorithmic generalization.

10 Conclusion

In this work, we introduced Model-Based Meta-Learning (MBML), a new paradigm for discovering RL algorithms using world models. We showed that MBML is up to $42\times$ faster in our experiments than meta-training on a full-fidelity physics simulator, without sacrificing meta-test performance on unseen continuous control environments. Notably, we found that world models need not be perfect to enable discovery of performant algorithms, in contrast to policy transfer requirements. Using MBML, we developed a novel meta-learned algorithm, Learned Optimizer & Loss (LOL), which significantly outperforms prior meta-learning methods. Overall, MBML lets researchers leverage world models trained on offline data rather than relying on slower, handcrafted environments for meta-training. This could significantly accelerate meta-RL research, lower the barrier to entry, and enable algorithm discovery at a much larger scale than previously explored. We outline many directions for future work in Section D.

AI USE STATEMENT

In this work, we used generative AI tools to propose or refine hypotheses, design or provide feedback on research methodology or experiments, interpret results, and implement methods. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, or assist in the writing of proofs. Additionally, we used generative AI tools to create or modify scientific figures or images, suggest experimental parameters, create or edit software code, create artifacts, summarize or analyze existing literature, discover research topics or identify gaps, brainstorm, sourcing/searching for information. We have reviewed all AI-assisted work, including the literature and the algorithm implementations. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and Francois Fleuret. Diffusion for world modeling: Visual details matter in atari, 2024. URL https://arxiv.org/abs/2405.12399.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study. October 2020. URL https://openreview.net/forum?id=nIAxjsniDzg.

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning, January 2023. tex.archiveprefix: arXiv tex.eprintclass: cs.LG.

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A Tutorial on Meta-Reinforcement Learning. *Foundations and Trends in Machine Learning*, 18(2-3):224–384, April 2025. ISSN 1935-8237, 1935-8245. doi: 10.1561/2200000080.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013. ISSN 1076-9757. doi: 10.1613/jair.3912. URL http://dx.doi.org/10.1613/jair.3912.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms, 2023. URL https://arxiv.org/abs/2302.06675.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *CoRR*, abs/1805.12114, 2018. URL http://arxiv.org/abs/1805.12114.

Jeff Clune. AI-GAs: AI-generating algorithms, an alternate paradigm for producing general artificial intelligence, February 2020. URL http://arxiv.org/abs/1905.10985. arXiv:1905.10985 [cs].

Longchao Da, Justin Turnau, Thirulogasankar Pranav Kutralingam, Alvaro Velasquez, Paulo Shakarian, and Hua Wei. A survey of sim-to-real methods in rl: Progress, prospects and challenges with foundation models, 2025. URL https://arxiv.org/abs/2502.13187.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning, 2016. URL https://arxiv.org/abs/1611.02779.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017. URL https://arxiv.org/abs/1703.03400.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax – a differentiable physics engine for large scale rigid body simulation, 2021. URL https://arxiv.org/abs/2106.13281.

Erik Frey and Brax authors. Brax ¡3 MuJoCo = MJX. GitHub Discussion #409, google/brax, October 2023. URL https://github.com/google/brax/discussions/409. Announcement of MuJoCo XLA (MJX) as a JAX-based physics pipeline, released alongside MuJoCo 3.0.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021. URL https://arxiv.org/abs/2004.07219.

Alexander David Goldie, Chris Lu, Matthew Thomas Jackson, Shimon Whiteson, and Jakob Nicolaus Foerster. Can learned optimization make reinforcement learning less difficult?, 2025a. URL https://arxiv.org/abs/2407.07082.

Alexander David Goldie, Zilin Wang, Jakob Nicolaus Foerster, and Shimon Whiteson. How Should We Meta-Learn Reinforcement Learning Algorithms?, July 2025b. URL http://arxiv.org/abs/2507.17668. arXiv:2507.17668 [cs].

David Ha and Jürgen Schmidhuber. World models, 2018. URL https://zenodo.org/record/1207631.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, January 2023. tex.archiveprefix: arXiv tex.eprintclass: cs.AI.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, pp. 1–7, 2025.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020. URL https://arxiv.org/abs/2006.11239.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Rein Houthooft, Richard Y. Chen, Phillip Isola, Bradly C. Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved Policy Gradients, April 2018. URL http://arxiv.org/abs/1802.04821. arXiv:1802.04821 [cs].

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, and Jeff Braga. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms, 2021. URL https://arxiv.org/abs/2111.08819.

Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

Matthew Thomas Jackson, Minqi Jiang, Jack Parker-Holder, Risto Vuorio, Chris Lu, Gregory Farquhar, Shimon Whiteson, and Jakob Nicolaus Foerster. Discovering general reinforcement learning algorithms with adversarial environment design, 2023. URL https://arxiv.org/abs/2310.02782.

Matthew Thomas Jackson, Chris Lu, Louis Kirsch, Robert Tjarko Lange, Shimon Whiteson, and Jakob Nicolaus Foerster. Discovering temporally-aware reinforcement learning algorithms, February 2024. tex.archiveprefix: arXiv tex.eprintclass: cs.LG.

Matthew Thomas Jackson, Uljad Berdica, Jarek Liesen, Shimon Whiteson, and Jakob Nicolaus Foerster. A clean slate for offline reinforcement learning, 2025. URL https://arxiv.org/abs/2504.11453.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel : Model-based offline reinforcement learning, 2021. URL https://arxiv.org/abs/2005.05951.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks, 2017. URL https://arxiv.org/abs/1706.02515.

Jakub Grudzien Kuba, Christian Schroeder de Witt, and Jakob Foerster. Mirror learning: A unifying framework of policy optimisation, 2024. URL https://arxiv.org/abs/2201.02373.

Seonho Lee, Amar Phanishayee, and Divya Mahajan. Forecasting gpu performance for deep learning training and inference. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, pp. 493–508. ACM, March 2025. doi: 10.1145/3669940.3707265. URL http://dx.doi.org/10.1145/3669940.3707265.

Zhenhua Li, Xi Lin, Qingfu Zhang, and Hailin Liu. Evolution strategies for continuous optimization: A survey of the state-of-the-art. *Swarm and Evolutionary Computation*, 56:100694, August 2020. ISSN 2210-6502. doi: 10.1016/j.swevo.2020.100694.

Jarek Liesen, Chris Lu, and Robert Lange. rejax, 2024. URL https://github.com/keraJLi/rejax.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018. URL https://arxiv.org/abs/1708.02002.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Chris Lu, Jakub Grudzien Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation, 2022. URL https://arxiv.org/abs/2210.05639.

Luke Metz. Exploring hyperparameter meta-loss landscapes with JAX. https://lukemetz.com/exploring-hyperparameter-meta-loss-landscapes-with-jax/, February 2021. Blog post.

Luke Metz, James Harrison, C. Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, and Jascha Sohl-Dickstein. Velo: Training versatile learned optimizers by scaling up, 2022. URL https://arxiv.org/abs/2211.09760.

Junhyuk Oh, Matteo Hessel, Wojciech M. Czarnecki, Zhongwen Xu, Hado van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms, 2021. URL https://arxiv.org/abs/2007.08794.

Junhyuk Oh, Gregory Farquhar, Iurii Kemaev, Dan A. Calian, Matteo Hessel, Luisa Zintgraf, Satinder Singh, Hado van Hasselt, and David Silver. Discovering state-of-the-art reinforcement learning algorithms. *Nature*, 648(8093):312–319, December 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09761-x.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013. URL https://arxiv.org/abs/1211.5063.

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.

Jürgen Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-. hook. 1987. URL https://api.semanticscholar.org/CorpusID:264351059.

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

Jascha Sohl-Dickstein. The boundary of neural network trainability is fractal, 2024. URL https://arxiv.org/abs/2402.06184.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The Dormant Neuron Phenomenon in Deep Reinforcement Learning, June 2023. URL http://arxiv.org/abs/2302.12902.

Yihao Sun. Offlinerl-kit: An elegant pytorch offline reinforcement learning library. https://github.com/yihaosun1124/OfflineRL-Kit, 2023.

Richard Sutton. The Bitter Lesson. http://www.incompleteideas.net/IncIdeas/BitterLesson.html, 2019.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, Vibhavari Dasagi, Lucy Gonzalez, Karol Gregor, Edward Hughes, Sheleem Kashem, Maria Loks-Thompson, Hannah Openshaw, Jack Parker-Holder, Shreya Pathak, Nicolas Perez-Nieves, Nemanja Rakicevic, Tim Rocktäschel, Yannick Schroecker, Jakub Sygnowski, Karl Tuyls, Sarah York, Alexander Zacherl, and Lei Zhang. Human-Timescale Adaptation in an Open-Ended Task Space, January 2023. URL http://arxiv.org/abs/2301.07608. arXiv:2301.07608 [cs.LG].

Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, Zhongwen Xu, and Shuicheng Yan. EnvPool: A highly parallel reinforcement learning environment execution engine. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 22409–22421. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8caaf08e49ddbad6694fae067442ee21-Paper-Datasets_and_Benchmarks.pdf.

P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. doi: 10.1109/5.58337.

Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-gradient reinforcement learning, 2018. URL https://arxiv.org/abs/1805.09801.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization, 2020. URL https://arxiv.org/abs/2005.13239.

APPENDIX

Our Appendix is structured as follows:

- Section A provides further details of our world model training, the termination functions used and a more fine-grained description of where our method fits in the landscape of prior methods. We also provide a description of the meta-learning hyperparameters and a validity proof for the drift function.
- Section B contains the complete experimental details to reproduce the results in this paper, such as the hardware used, information about Brax backends, an exhaustive list of hyperparameters, and the characteristics of the evaluation environments.
- Section C provides additional results that further support the value of our contributions, such as meta-training on consumer hardware, Atari results using diffusion world models, LPG results, results from using randomly initialized world models, more ablations, and meta-training across multiple environments.
- Section D lists potential future research threads.
- Section E describes our work's impact and suggests measures to ensure safe and responsible deployment of meta-learned algorithms.

A Extended Methods

A.1 World Models

A.1.1 Termination

We use the world model implementation from (Jackson et al., 2025) that combines the step-wise uncertainty penalty from MOPO (Yu et al., 2020) and the pessimistic MDP termination when the ensemble disagreement reaches a pre-defined threshold, as done in MOREL (Kidambi et al., 2021). The termination threshold is determined by the largest distance observed in the world model training datasets. Once the threshold is exceeded during training, the episode terminates, and the agent incurs a penalty equal to -200 plus the minimum reward found in the training dataset.

A.1.2 World Model training

For each world model, we train an ensemble of seven MLPs using a Negative Log Likelihood (NLL) Loss and form the final ensemble with the best five based on their validation loss at the end of training (Jackson et al., 2025). We hold out 20% of each dataset for validation. Since we find that the validation loss performance is highly sensitive to hyperparameter choice, we tune the hyperparameters using random search within the ranges specified in Appendix B.3.1.

A.2 Learned Optimizer & Loss

We design a new meta-learning method that is a generalization of prior works, we call the resultant algorithm *Learned Optimizer & Loss* (LOL).

A.2.1 Construction

We highlight in red modifications from prior methods. We take a scaffold of PPO and replace the drift function with TA-LPO (Jackson et al., 2024). To obtain a true generalization over algorithms, we multiply the PPO drift term by a learnable meta-parameter ϕ_α, we additionally replace the clipping boundary ϵ with a learnable ϕ_ϵ, such that we have

$$\mathfrak{D}_{\phi_\mathfrak{D},\phi_H}(\boldsymbol{x}_{r,A,t}) = \text{ReLU}\left(h_{\phi_\mathfrak{D}}(\boldsymbol{x}_{r,A,t}) - \xi + \phi_\alpha \,\text{ReLU}\left((r - \text{clip}(r, 1 \pm \phi_\epsilon)) \cdot A\right)\right).$$

This allows the meta-optimization to interpolate between the learned drift and PPO's drift by changing ϕ_α. The proof that LOL's drift function still obeys the constraints of Mirror Learning is in Section A.2.3.

Similarly, we replace the optimizer with OPEN (Goldie et al., 2025a), but modify the update for each policy and critic parameter p^i to obtain a generalization over optimizers, and bootstrap the learning process. We initialize OPEN updates $\bar{u}^i_{\text{OPEN}}$ with an `adamW` optimizer (Loshchilov & Hutter, 2019). (Huang et al., 2022) observe that one of `adam`'s hyperparameters' default value, while generally optimal for supervised learning, is potentially suboptimal for RL. Motivated by this, we include the `adamW` hyperparameters in the meta-optimization, these include the learning rate lr but also $\beta_1, \beta_2, \epsilon_{\text{adam}}$ and weight regularizer λ. The update then becomes

$$p^i_{\text{new}} = p^i_{\text{old}} + \bar{u}^i_{\text{OPEN}} + \phi_\omega u^i_{\texttt{adamW}},$$
$$\text{where } u^i_{\texttt{adamW}} = \texttt{adamW}(g^i, p^i_{old}, \phi_{lr}, \phi_{\beta_1}, \phi_{\beta_2} \phi_{\epsilon_{\text{adam}}}, \phi_\lambda),$$

where g^i is the gradient for the parameter p^i and ϕ_ω is the learnable weight for the optimizer. $\phi_{lr}, \phi_{\beta_1}, \phi_{\beta_2}, \phi_{\epsilon_{\text{adam}}}, \phi_\lambda$ are the learnable `adamW` hyperparameters. This enables the meta-optimization to recover a standard optimizer by setting all the updates from OPEN to 0 and setting ϕ_ω to 1.

A.2.2 Meta-learning hyperparameters

Unlike prior works, we treat many hyperparameters from PPO as learnable meta-parameters; we label them ϕ_H. However, certain hyperparameters are not included, as they cause the training loop to be recompiled at each iteration and significantly slow down meta-training. The hyperparameters we hold

fixed are the number of **parallel actors**, the number of **minibatches**, the number of **epochs**, and the number of **environment steps per actor** before updating. To prevent hyperparameters from taking values that are not desirable (such as $\gamma > 1$), we scale them through a sigmoid before passing them to the algorithm. The sigmoid is defined by a lower and upper bound. For certain hyperparameters, we use a log-linear scaling to enable the hyperparameter to take values spanning multiple orders of magnitude. We define all of these in Section B.3.2.

A.2.3 Proof of validity of Drift Function

Kuba et al. (2024) describe that a valid drift function is non-negative everywhere and is equal to 0 when the ratio of policies $r = \frac{\pi_{new}(s,a)}{\pi_{\text{old}}(s,a)}$ is equal to 1. Additionally, the derivative of the drift function, evaluated at $r = 1$ must also be 0. In LOL's drift function,

$$\mathfrak{D}_{\phi_{\mathfrak{D}},\phi_H}(\boldsymbol{x}_{r,A,t}) = \text{ReLU}\left(h_{\phi_{\mathfrak{D}}}(\boldsymbol{x}_{r,A,t}) - \xi + \phi_\alpha \,\text{ReLU}\left((r - \text{clip}(r, 1 \pm \phi_\epsilon)) \cdot A\right)\right),$$

where ξ is a small positive value (e.g. 10^{-6}, (Lu et al., 2022)), we define

$$\begin{aligned}\boldsymbol{x}_{r,A,t} = [&(1-r), (1-r)^2, (1-r)A, (1-r)^2A, \log(r), \log(r)^2,\\ &\log(r)A, \log(r)^2A, t/T(1-r), t/T(1-r)^2, t/T(1-r)A, t/T(1-r)^2].\end{aligned}$$

It can be trivially seen that $\mathfrak{D}_{\phi_{\mathfrak{D}},\phi_H}(\boldsymbol{x}_{r,A,t})$ is positive everywhere because of the ReLU, and that when $r = 1$, the drift evaluates to 0. This is because $h_{\phi_{\mathfrak{D}}}$ is an MLP with no biases. For the derivative, we define the function $D(r) = \text{ReLU}(y(r))$ where:

$$y(r) = h_{\phi_{\mathfrak{D}}}(\boldsymbol{x}_{r,A,t}) - \xi + \phi_\alpha \,\text{ReLU}\left((r - \text{clip}(r, 1 \pm \phi_\epsilon)) \cdot A\right) \tag{3}$$

$h_{\phi_{\mathfrak{D}}}(\boldsymbol{x}_{r,A,t}) = 0$ for $r = 1$ by design so $y(r = 1) < 0$. Given this, taking derivatives:

$$\frac{d}{dr}D(r) = \frac{d}{dy}\text{ReLU}(y(r)) \cdot \frac{d}{dr}y(r), \tag{4}$$

$$= (\mathbb{I}(y(r) > 0) + a\mathbb{I}(y(r) = 0)) \cdot \frac{d}{dr}y(r), \tag{5}$$

(using the subgradient $\frac{d}{dy}\text{ReLU}(y = 0) = a$ for $a \in [0, 1]$). Now, we assume $|\frac{d}{dr}y(r)| < \infty$, and because $y(r = 1) < 0$, substituting into Equation (5):

$$\frac{d}{dr}D(r = 1) = 0 \cdot \frac{d}{dr}y(r) = 0 \tag{6}$$

We have therefore shown that our modified drift function still obeys the constraints of Mirror Learning.

B Experimental Details

B.1 Computational Resources

We run our experiments on a cluster of Nvidia H200 GPUs with `aarch64` CPU architecture. For meta-training (online and MBML), we use four H200's in parallel. To run the experiments in Section 7.1, we calculate an upper bound of 4320 GPU hours ($24h \times 4$ GPUs $\times$ 5 seeds $\times$ 3 algorithms $\times$ 3 envs $= 4320$). For meta-testing, we use one H200 per RL training run. We estimate that meta-testing accounts for approximately 200 GPU-hours and world model training is negligible. Additionally, we use a single RTX 3080Ti for experiments to show that MBML enables Meta-RL on consumer-grade hardware.

B.2 Brax Backends

The Brax suite of environments has different physics backends that can be chosen. We opt to use the *generalized* backend for *all* evaluations, as it is supported by all environments in the suite, and closely resembles the MuJoCo engine. Despite this, prior works have used the *positional* backend for meta-training and evaluation, a less accurate, but faster backend (Goldie et al., 2025a; Jackson et al., 2024; Lu et al., 2022). The reason for this is that meta-training with generalized is computationally expensive in wall-clock time (Figure 1). We note that the D4RL datasets Fu et al. (2021) were collected in the true MuJoCo and therefore resemble the *generalized* backend the most, while still not being perfectly equivalent Freeman et al. (2021). Another more recent backend is the *MJX* backend. We found early in our experiments that the MJX backend runs approximately twice as slowly as the generalized backend and therefore opted to use the generalized backend. Nevertheless, the Brax team officially announced the generalized backend as deprecated in favor of MJX Frey & Brax authors (2023), citing it as "more performant". We opted to use the generalized backend despite this due to its faster speed (Figure 9)

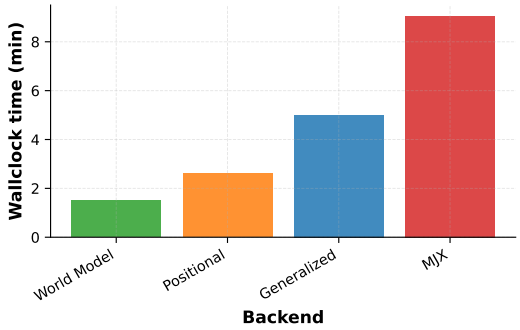


Figure 9: Wallclock time comparison for training a PPO agent on the hopper environment with 4 different backends and 50M steps. MJX is much slower despite being advertised as "more performant.".

B.3 HYPERPARAMETERS

B.3.1 WORLD MODEL TRAINING

Table 2: Training Hyperparameters for the different world models

Environment	**halfcheetah**	**hopper**	**walker2d**
dataset	medium-expert-v2	medium-expert-v2	medium-expert-v2
learning rate	0.001	0.0006	0.0002
batch size	256	256	1048
num epochs	800	600	400
n_layers	8	4	6
layer size	200	200	400
num ensemble	7	7	7
num elites	5	5	5
weight decay	0.00002	0.0001	0.0001
logvar diff coef	0.01	0.1	0.1
validation split	0.2	0.2	0.2

Table 3: Hyperparameter sweep configuration for training ensemble dynamics world models.

Category	**Parameter**	**Search Space / Value**
Architecture	n_layers	{2, 4, 6, 8}
	layer_size	{100, 200, 400}
	num_ensemble	7 (fixed)
	num_elites	5 (fixed)
Training	dataset	{walker2d, hopper, halfcheetah}-medium-expert-v2
	batch_size	$\mathrm{LogUniform}_{q=2}[128, 2048]$
	num_epochs	{200, 400, 600, 800}
	validation_split	0.2 (fixed)
Optimization	learning_rate	$\mathrm{LogUniform}[10^{-5}, 10^{-2}]$
	weight_decay	$\mathrm{LogUniform}[10^{-5}, 10^{-2}]$
	logvar_diff_coef	{0.001, 0.01, 0.1}
Reproducibility	seed	0 (fixed)

B.3.2 META-TRAINING HYPERPARAMETERS

Table 4: Initial configuration of PPO. Inspired by common PPO implementations (Huang et al., 2021; Liesen et al., 2024; Lu et al., 2022).

Category	Parameter	Initial Configuration
Held Constant	total_timesteps	50_000_000
	num_envs	2000
	num_steps	25
	num_epochs	8
	num_minibatches	25
	activation	tanh
	hidden_layers	[64, 64]
	normalize_observations	true
	normalize_rewards	true
	optimizer	adamW
Evolved with ES	learning_rate	3e-4
	gamma	0.99
	gae_lambda	0.95
	clip_eps	0.2
	ent_coef	0.01
	vf_coef	0.5
	max_grad_norm	2
	value_clip_eps	0.2
	adam_b1	0.9
	adam_b2	0.999
	adam_eps	1e-8
	adam_weight_decay	0.0001
LOL-specific	ppo_drift_weight	1.0
	adam_weighting	1.0

Table 5: Initial hyperparameter values and corresponding bounds for sigmoidal parameterization for hyperparameter evolution. Hyperparameters marked as log-scaled are optimized in log-space before sigmoid mapping.

Hyperparameter	Initial Value	Lower Bound	Upper Bound	Log-Scaled
learning_rate	3×10^{-4}	10^{-6}	10^{-2}	Yes
max_grad_norm	2	0.01	10.0	No
gamma	0.99	0.6	0.999999	No
gae_lambda	0.95	0.01	1.0	No
clip_eps	0.2	0.01	2.0	No
ent_coef	0.01	10^{-4}	1.0	Yes
vf_coef	0.5	10^{-4}	1.0	Yes
adam_weighting	1.0	-5.0	5.0	No
ppo_drift_weight	1.0	-5.0	5.0	No
value_clip_epsilon	0.2	10^{-6}	1.0	No
adam_b1	0.9	0.8	0.999999	No
adam_b2	0.999	0.8	0.999999	No
adam_eps	10^{-8}	10^{-12}	10^{-3}	Yes
adam_weight_decay	10^{-4}	10^{-10}	10.0	Yes

B.4 META-TRAINING

Table 6: Meta-Training combinations

Parameter	Value
algorithm	[TA-LPO, OPEN, LPO]
environment	[halfcheetah, walker2d, hopper]
train_backend	[positional, generalized, world model]
eval_backend	generalized
train_config_id	50M_steps
eval_config_id	50M_steps
seed	[0,1,2,3,4]
pop_size	32
num_gens	100
n_eval_seeds	8
algo_eval_freq	10
lpo_hidden_size	128
open_hidden_size	16
open_gru_features	8
es_optimizer	adamW
es_learning_rate	0.03
es_lr_decay_rate	0.995
es_std_schedule	0.03
es_std_decay_rate	0.995

B.5 Evaluation Environments

Table 7: Meta-test environments, categorized by state space, action space, and reward structure. *Note that HalfCheetah was used as a meta-validation environment and therefore excluded from the meta-test set.

Environment	State Space	Action Space	Rewards
Brax			
brax/halfcheetah*	Continuous	Continuous	Dense
brax/walker2d	Continuous	Continuous	Dense
brax/hopper	Continuous	Continuous	Dense
brax/humanoid	Continuous	Continuous	Dense
brax/humanoidstandup	Continuous	Continuous	Dense
brax/ant	Continuous	Continuous	Dense
brax/reacher	Continuous	Continuous	Dense
brax/swimmer	Continuous	Continuous	Dense
brax/pusher	Continuous	Continuous	Dense
brax/inverted_pendulum	Continuous	Continuous	Dense
brax/inverted_double_pendulum	Continuous	Continuous	Dense
Classic Control			
Pendulum-v1	Continuous	Continuous	Dense
CartPole-v1	Continuous	Discrete	Dense
Craftax			
craftax/Craftax-Symbolic-AutoReset-v1	Discrete	Discrete	Sparse
craftax/Craftax-Classic-Symbolic-AutoReset-v1	Discrete	Discrete	Sparse
Navix			
navix/Navix-FourRooms-v0	Discrete	Discrete	Sparse
navix/Navix-Dynamic-Obstacles-6x6-Random-v0	Discrete	Discrete	Sparse
navix/Navix-DoorKey-8x8-v0	Discrete	Discrete	Sparse
Kinetix			
kinetix/s/h1_thrust_over_ball	Continuous	Discrete	Dense
kinetix/m/h17_thrustcontrol_left	Continuous	Discrete	Dense
kinetix/l/hard_pinball	Continuous	Discrete	Dense
Jumanji			
jumanji/Snake-v1	Discrete	Discrete	Sparse
jumanji/Game2048-v1	Discrete	Discrete	Sparse
jumanji/Knapsack-v0	Discrete	Discrete	Sparse

C Extended Results

C.1 Meta-training on consumer-grade hardware

To show that MBML makes meta-training more accessible, we demonstrate using it with a single Nvidia RTX 3080Ti in Figure 10.

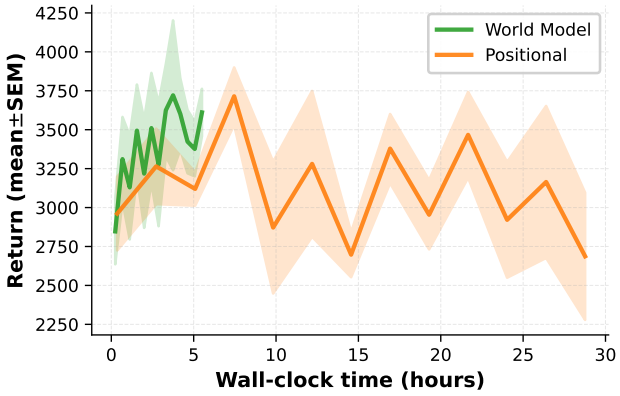


Figure 10: Meta-validation performance on halfcheetah vs. wall-clock time, when meta-training with the world model or positional backend on a single RTX 3080Ti, using a population size of 16. Each agent is meta-trained for 10M steps. Mean and SEM over 5 meta-seeds.

C.2 Meta-training and Meta-validation performance

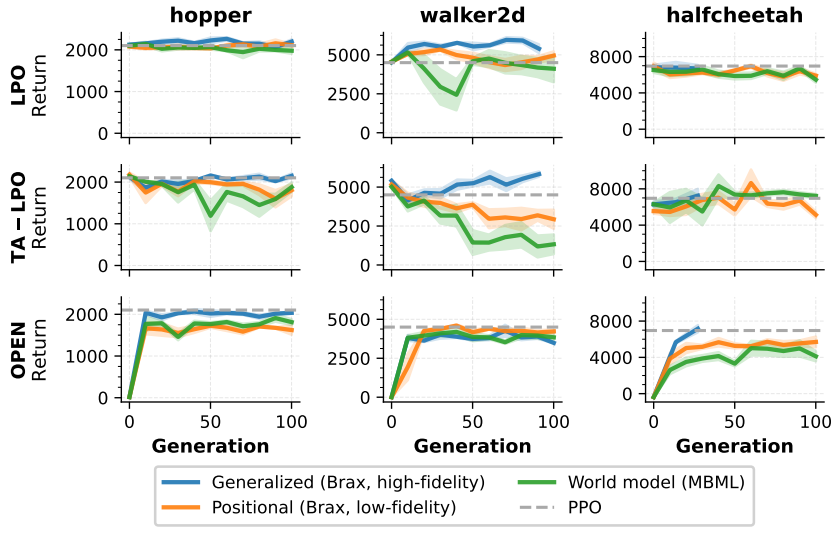


Figure 11: Meta-validation performance during meta-training, measured as final return in the generalized backend (the meta-validation setting). MBML (green) yields results comparable to meta-training with the positional backend (orange). Both fall short of meta-training directly in the meta-validation setting given by the generalized backend (blue). We note that for halfcheetah-generalized, we completed only 20 generations of meta-training due to the 24h stopping condition. Bootstrapped 95% confidence intervals over 5 meta-seeds.

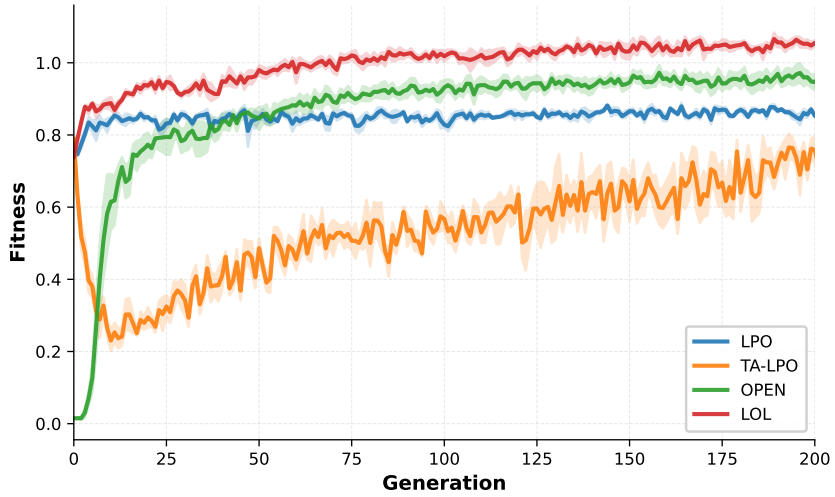


Figure 12: LOL obtains the highest fitness in the *halfcheetah* world model when meta-training with MBML. Mean and SEM over 5 meta-seeds.

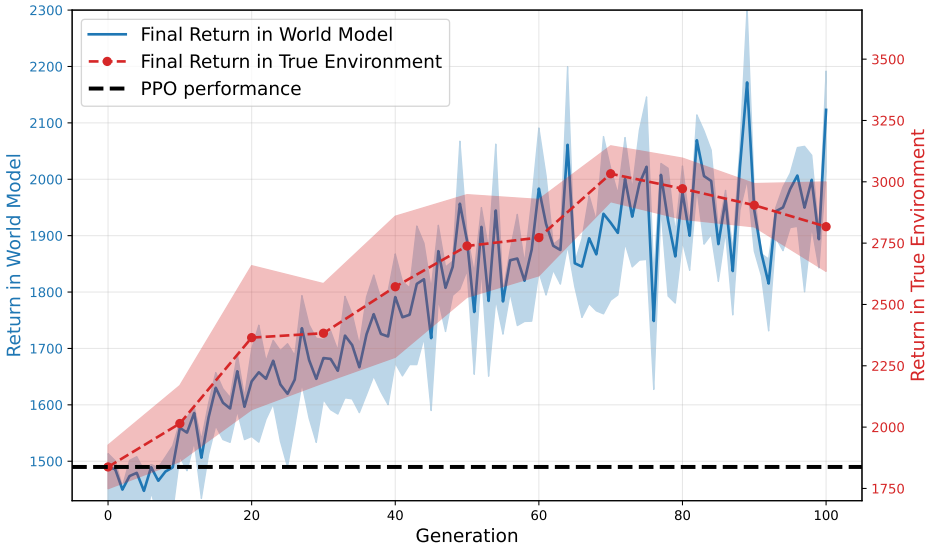


Figure 13: Increasing meta-training performance correlates with increasing meta-validation performance for LOL on the Hopper world model. Mean and SEM of 5 meta-seeds.

C.3 Atari Results

We show in Figure 14 that we obtain a 37x speedup on the original implementation of ALE (Bellemare et al., 2013). We cannot use envpool (Weng et al., 2022) because it does not provide compiled binaries for `aarch64` CPU architectures. Therefore, meta-training directly on the true ALE-py implementation is not computationally achievable within the 24h time limit. In Figure 15 we plot the performance across denoising steps and compare it to PPO. Higher step count corresponds to higher world-model fidelity; (Alonso et al., 2024) use 3 denoising steps for policy training and obtain competitive results on the Atari 100k benchmark.

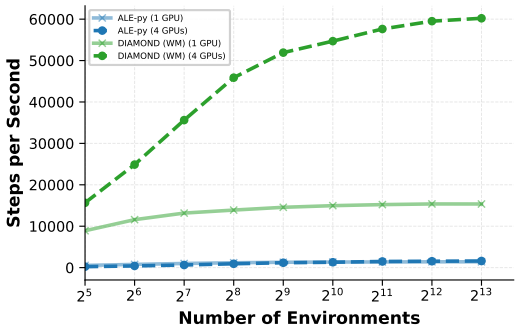


Figure 14: Steps per second for the DIAMOND world model (Alonso et al., 2024) versus the ALE-py, environment implementation (Bellemare et al., 2013). DIAMOND reaches 60k versus 1607 for ALE-py, a 37x increase in throughput. The increase scales linearly with more GPUs. We reimplement DIAMOND in JAX (Bradbury et al., 2018) for compatibility with our codebase.

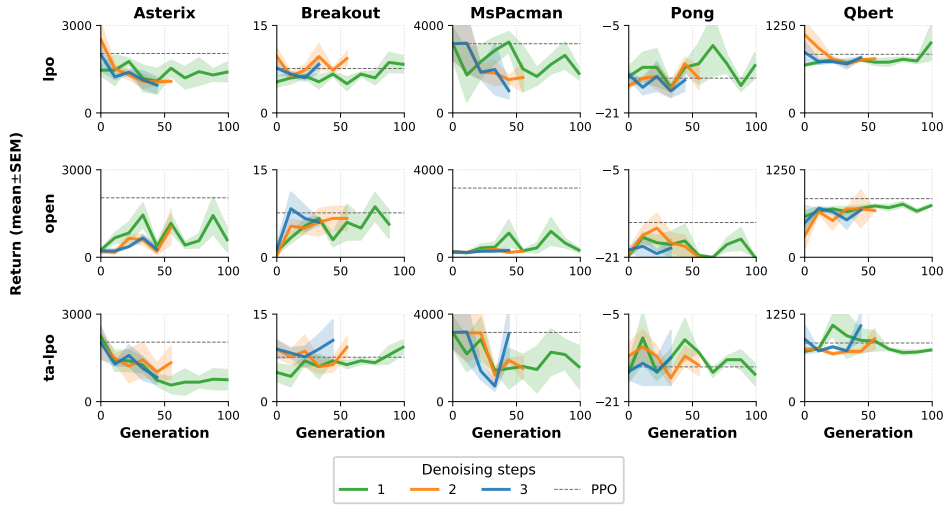


Figure 15: Changing the number of denoising steps on the DIAMOND world models from the original 3 (Alonso et al., 2024) does not noticeably alter meta-validation performance on the respective ALE-py environments. Mean and SEM across 5 meta-seeds.

C.4 LPG Results

Here we show our LPG results. In Figure 16, we show training time for different backends and our framework. In Figure 17 and Figure 18 we evaluate on the real environments.

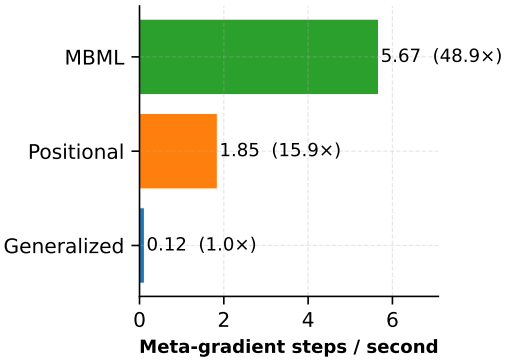


Figure 16: LPG meta-gradient steps per second for the different backends on the halfcheetah environment.

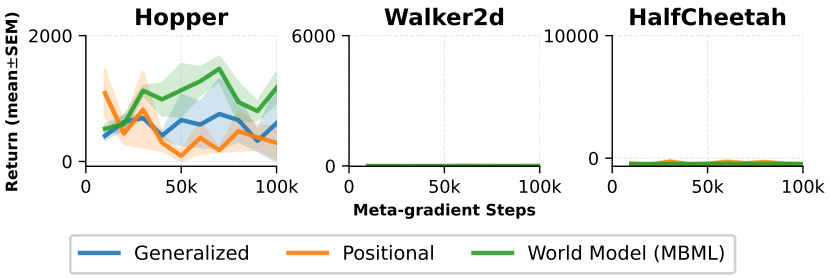


Figure 17: LPG results on continuous environments with action discretization. Action discretization hurts performance the most for walker2d and halfcheetah (64 actions) when compared to hopper (27 actions).

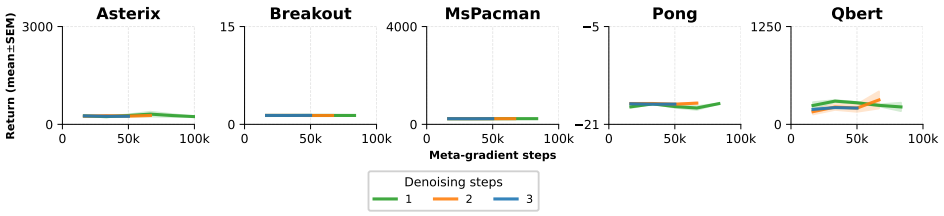


Figure 18: Evaluating LPG throughout meta-training on Atari. LPG struggled to learn a valid update rule within the allocated 100k meta-gradient steps or 24h limit, regardless of the number of denoising steps.

C.5 META-TRAINING FOR 0-SHOT TRANSFER

In our experiments, we use the world model itself to evaluate the trained policies at each generation in meta-training. However, we can modify MBML to use a different backend for policy evaluation. This means we meta-train to transfer *between* backends, learning algorithms that close the world-model-to-simulation gap. We show that this meta-training setup can improve zero-shot generalization to an unseen backend.

We conduct an experiment in which, instead of using the world model for policy evaluation, we use a physics simulator with the *positional backend*. We evaluate backend transfer of this new algorithm by evaluating the world model for *generalized* transfer. In Figure 19, we see that throughout meta-training, 0-shot transfer of the world-model trained policy increases. The meta-learned algorithm is learning policies that generalize better.

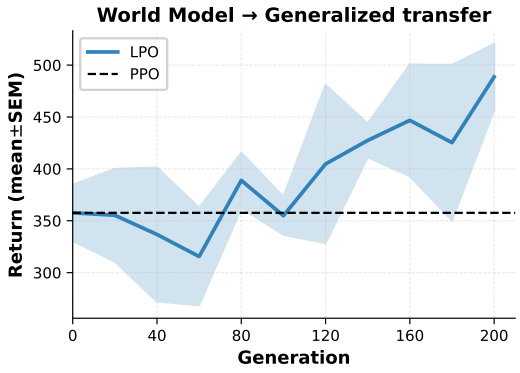


Figure 19: LPO learns policies that generalize better to a new backend on the Hopper environment. Mean and SEM over 5 meta-seeds.

C.6 META-TRAINING ON RANDOMLY INITIALIZED NETWORKS

For this experiment, we use the same ensemble setup, but instead of a trained ensemble of world models, we initialize the networks with LeCun normal noise (Klambauer et al., 2017).

In Figure 20, only OPEN shows consistent improvement throughout meta-training, and most of that improvement is concentrated at the start of meta-training (Figure 21). We posit that OPEN learns a basic policy and critic update mechanism, but does so inefficiently. For other algorithms, random networks provide no learning signal; this degrades LOL and has negligible effects on LPO, TA-LPO. For LPG, no learning occurs in this case. These results suggest that random networks are a useful environment for learning simple algorithm components, but they are insufficient for more complex update rules, such as those needed for strong-performing RL algorithms.

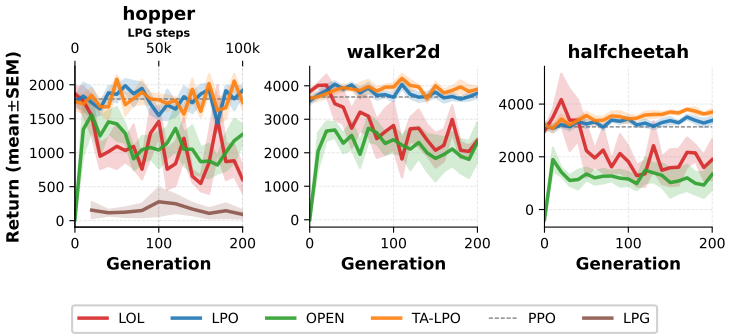


Figure 20: Meta-test performance throughout meta-training of different algorithms on random networks. We do not meta-test LPG on halfcheetah or walker2d given that action discretization stops any learning as shown in Figure 17.

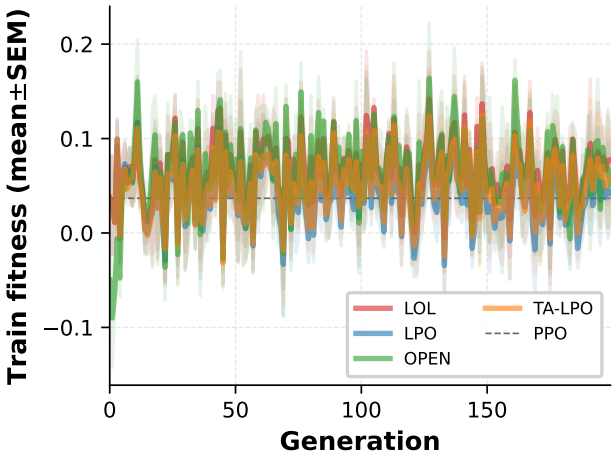


Figure 21: Fitnesses during meta-training on random networks. Fitnesses do not improve throughout meta-training except for OPEN. We observe a strong correlation between algorithms because the random seeds are correlated.

C.7 World Model analysis

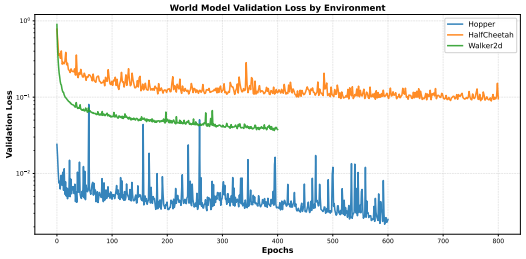


Figure 22: World model validation loss per dataset

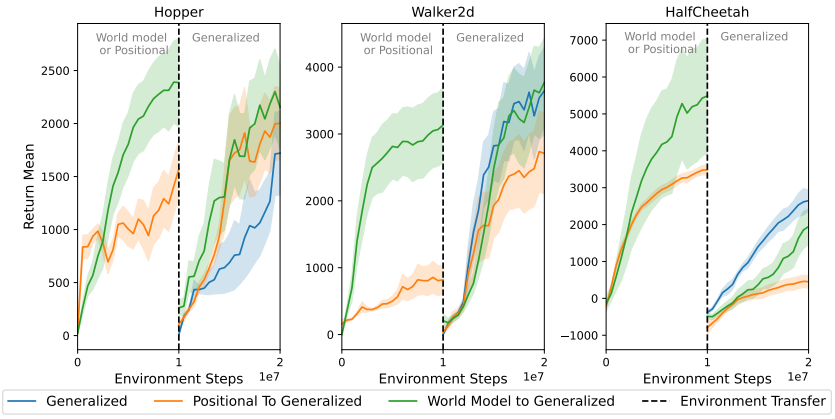


Figure 23: Transferring a policy between backends causes catastrophic collapse. Further training the policy is only clearly better than training from scratch in the *hopper* environment. Average over 5 seeds.

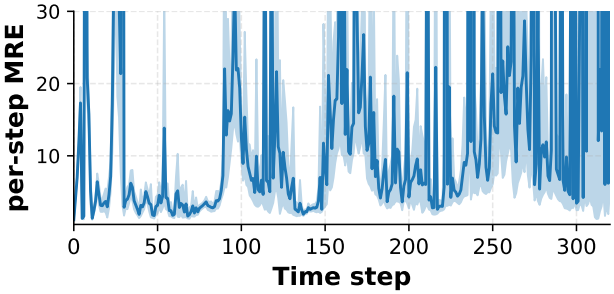


Figure 24: Mean Relative Error in the observation produced in *halfcheetah* and a corresponding world model. We roll out an expert policy in the generalized backend, and replay its actions to the world model, from the same starting state. Bootstrapped averages and 95% confidence intervals over 10 runs.

C.8 META-TRAINING DETAIL

Table 8: Hyperparameters of PPO-tuned, discovered by using ES in the *halfcheetah* world model.

Category	Parameter	Configuration
Held Constant	total_timesteps	50,000,000
	num_envs	2000
	num_steps	25
	num_epochs	8
	num_minibatches	25
	activation	tanh
	hidden_layer_sizes	[64, 64]
	normalize_observations	true
	normalize_rewards	true
Optimized Hyperparameters	learning_rate	3.5007e-4
	gamma	0.99244
	gae_lambda	0.92313
	clip_eps	0.16961
	ent_coef	0.0029018
	vf_coef	0.58394
	max_grad_norm	2.12551
	value_clip_epsilon	0.22877
	adam_b1	0.90853
	adam_b2	0.99928
	adam_eps	1.9572e-7
	adam_weight_decay	0.00030622

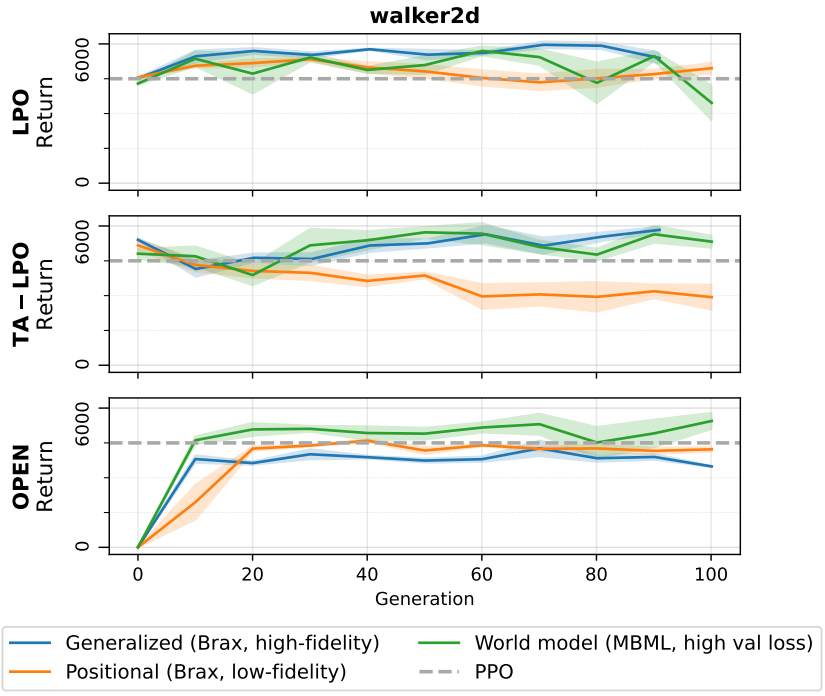


Figure 25: We use a world model from *walker2d-medium-expert* with a validation loss that is higher than the one used previously (0.155 versus 0.032) for meta-training; this suggests that world model accuracy is not the most relevant factor for meta-training good algorithms. Hopper generalized (Meta-validation) performance throughout meta-training. Mean and SEM of 5 meta-seeds.

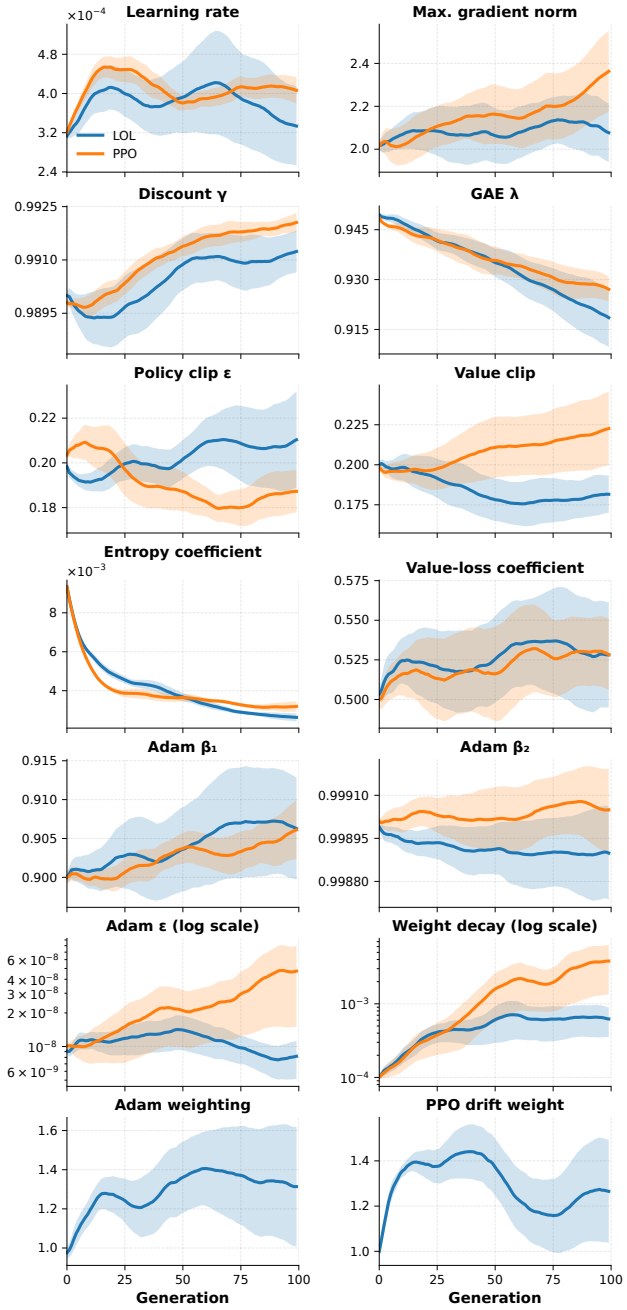


Figure 26: Evolution of hyperparameters during meta-training LOL and PPO on the *halfcheetah* world model. Mean and SEM across 5 meta-seeds. The algorithm learns to decrease entropy during meta-training.

C.9 ABLATIONS

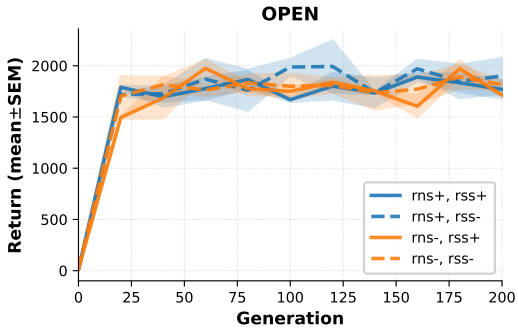
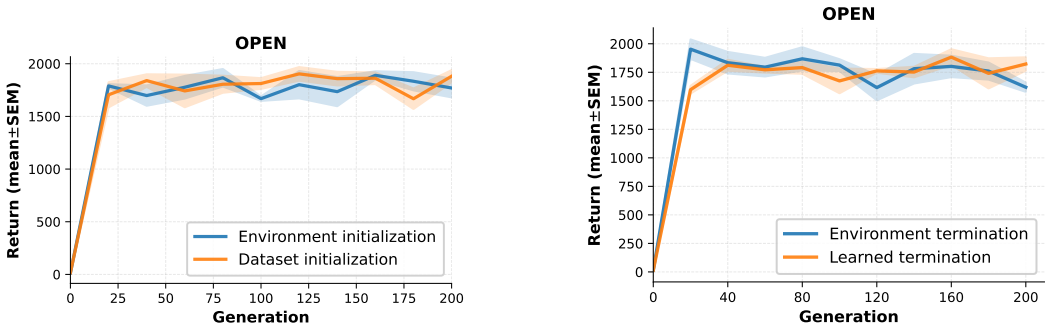


Figure 27: Ablating the specific designs that guard against epistemic uncertainty (random network sampling: **rns**) and aleatoric uncertainty (random state sampling: **rss**) in the models (Chua et al., 2018). We observe that MBML is robust to these world-model design choices.



(a) Initialization ablation

(b) Termination ablation

Figure 28: MBML is agnostic to the use of the original environment termination and environment initialization. For **Dataset initialization**, we reset using a random state drawn from the dataset. For **Learned termination**, we learn a termination function along with the transition function using a focal loss (Lin et al., 2018, $\gamma = 2$, $\alpha = 0.75$).

C.10 Analysis of Meta-learned algorithms

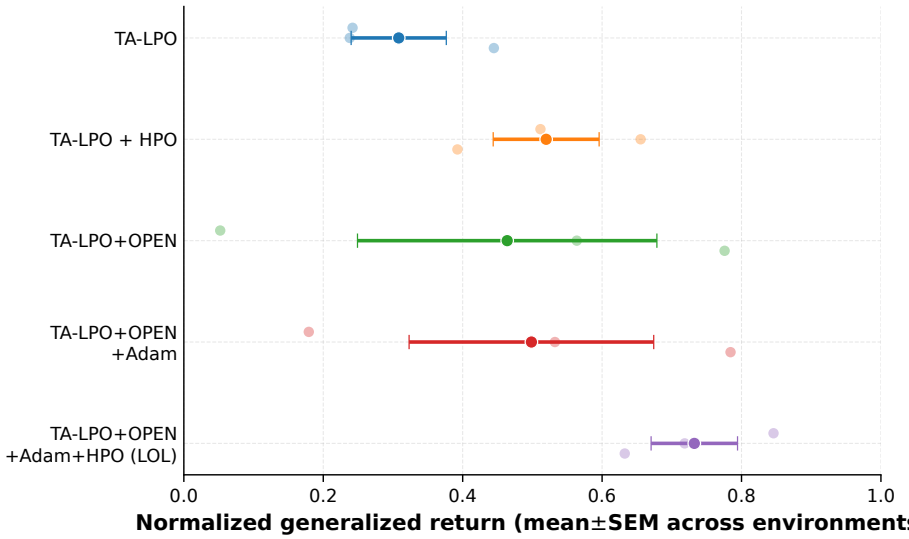


Figure 29: LOL ablation: each component independently helps performance. We train each ablation in each world model with 3 meta-seeds and then measure performance in the respective environment with the generalized backend. We normalize between 0 and 1 using a random policy and the best policy found. Each shaded datapoint corresponds to the mean performance on one of Hopper, HalfCheetah, or Walker2d.

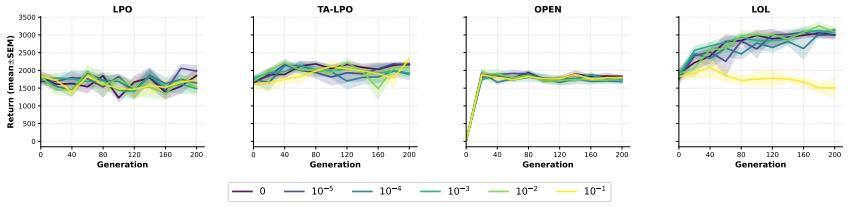


Figure 30: Outer optimizer Regularization does not make a statistical difference in meta-validation. Meta-training on the Hopper world model backend and evaluating on the Hopper generalized backend. We vary the weight decay of the outer `adamW` optimizer over 5 meta-seeds. We note that strong regularization only affects LOL, which jointly optimizes hyperparameters as well.

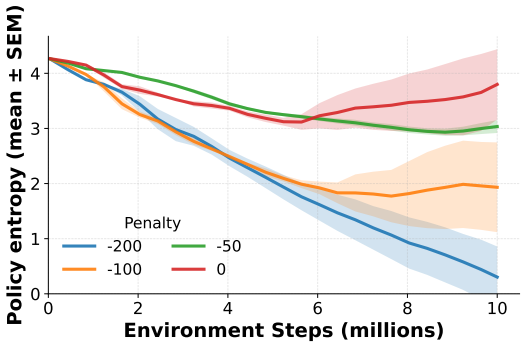
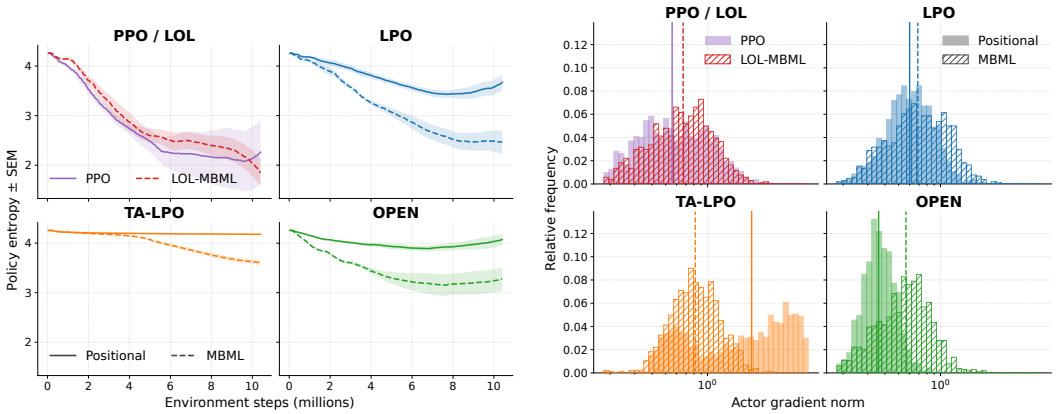


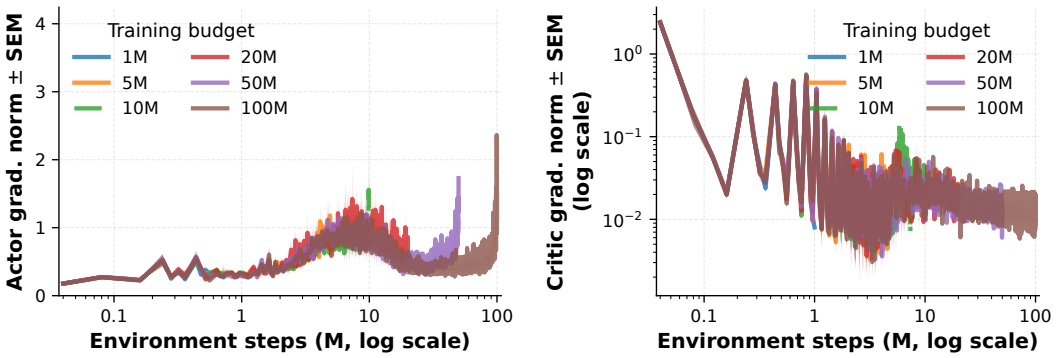
Figure 31: Properties of the world model shape learned behavior. Decreasing the halting penalty from Kidambi et al. (2021) 's original -200 for the Hopper world model results in meta-learned LPO that learns lower-entropy policies when then training on Hopper (generalized).



(a) Policy entropy during training.

(b) Actor gradient-norm distributions.

Figure 32: Policy-update dynamics on Hopper (Generalized), comparing update rules meta-trained using Positional simulation or MBML, with PPO as a baseline. Entropy curves show means across five policy-training seeds, with SEM. We note that MBML seems to induce lower-entropy policies, which we associate with the halting penalty as shown in Figure 31. Histograms are gradient norms across seeds and training updates, normalized within the displayed range; vertical lines mark medians. No particular pattern emerges from the gradient norms.



(a) Actor gradient norms

(b) Critic gradient norms (log)

Figure 33: Gradient norms during policy training on Hopper using the fixed LOL-MBML update rule with different training budgets. Mean and SEM across 5 training seeds.

C.11 Algorithm Evaluation

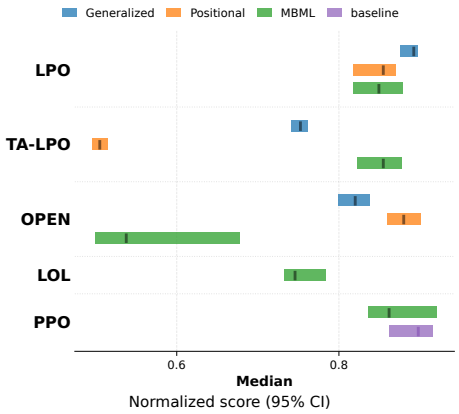


Figure 34: Aggregate scores on environments semantically different from continuous control tasks. PPO significantly outperforms other algorithms meta-trained on continuous control tasks. We explain LPO-Generalized's strong performance by its PPO initialization and its 20-step meta-training.

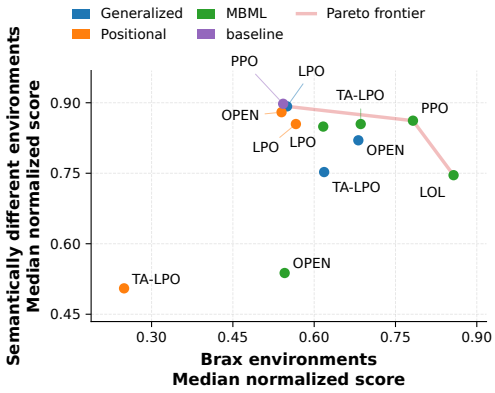


Figure 35: Plotting median Brax returns versus non-Brax returns highlights a Pareto frontier of algorithmic performance, consistent with the No-free lunch theorem. Each data point represents 24 environments $\times$ 10 training seeds $\times$ 128 policy evaluations.

Table 9: Final return on Brax and non-Brax environments (mean $\pm$ SEM over 10 seeds). The highest unrounded mean in each environment is bold. Higher is better.

Environment	PPO	PPO MBML	LOL MBML	LPO Generalized	LPO Positional	LPO MBML	TA-LPO Generalized	TA-LPO Positional	TA-LPO MBML	OPEN Generalized	OPEN Positional	OPEN MBML	SAC
Brax environments													
Ant	1800.64 ± 88.31	4663.90 ± 62.46	**6445.73 ± 157.71**	1883.95 ± 55.54	1938.20 ± 57.59	2856.67 ± 67.03	1823.79 ± 53.17	1.05 ± 0.52	2353.56 ± 93.48	2691.86 ± 95.20	2192.16 ± 74.86	1855.51 ± 71.77	6329.68 ± 104.03
HalfCheetah	7524.21 ± 903.20	**9645.70 ± 2024.61**	7486.50 ± 263.92	6794.78 ± 220.16	6937.04 ± 232.29	7070.31 ± 292.00	4086.88 ± 375.30	368.29 ± 40.01	4048.76 ± 307.18	7170.07 ± 303.23	7376.28 ± 173.64	6246.26 ± 510.97	2549.03 ± 696.78
Hopper	2069.18 ± 53.66	2887.70 ± 123.44	**3257.54 ± 115.13**	2096.56 ± 34.56	2158.15 ± 62.06	2348.82 ± 77.23	1964.37 ± 88.39	960.04 ± 65.31	2072.89 ± 51.93	2001.41 ± 109.78	2057.92 ± 46.68	2079.74 ± 64.73	434.35 ± 24.39
Humanoid	3020.91 ± 124.60	5514.45 ± 250.24	**5981.10 ± 205.05**	3391.57 ± 128.71	3217.08 ± 90.73	3395.81 ± 95.54	4148.32 ± 165.80	4512.48 ± 158.92	4640.55 ± 96.30	4072.14 ± 208.63	2648.58 ± 177.98	4519.29 ± 127.97	1915.57 ± 250.72
Humanoid Standup	31357.68 ± 238.89	32896.21 ± 239.72	31075.15 ± 470.70	34345.51 ± 2261.00	31989.09 ± 223.93	30426.73 ± 1808.09	32896.88 ± 173.30	30948.67 ± 61.05	32896.67 ± 140.50	**44038.02 ± 3859.28**	31548.03 ± 119.36	31520.02 ± 347.94	27279.48 ± 328.72
Inverted Double Pendulum	9005.76 ± 88.33	9175.75 ± 90.06	**9286.21 ± 21.97**	8813.06 ± 231.00	8698.83 ± 436.82	9055.81 ± 102.14	8843.13 ± 144.40	7048.72 ± 525.07	9148.33 ± 77.61	9281.83 ± 27.35	8450.38 ± 583.80	9208.86 ± 105.77	6804.02 ± 1093.15
Inverted Pendulum	997.19 ± 1.99	942.25 ± 57.75	996.79 ± 1.05	978.05 ± 4.43	982.91 ± 1.94	**1000.00 ± 0.00**	985.03 ± 2.72	874.39 ± 16.91	977.98 ± 1.79	990.72 ± 1.95	980.47 ± 2.80	996.08 ± 3.74	998.37 ± 1.01
Pusher	−356.58 ± 29.40	−340.45 ± 26.45	−790.52 ± 130.98	−315.68 ± 23.49	−340.09 ± 13.78	−314.86 ± 12.41	−285.33 ± 18.60	−1305.33 ± 30.75	**−277.45 ± 9.94**	−303.76 ± 23.21	−370.11 ± 39.45	−1201.61 ± 162.45	−396.34 ± 13.92
Reacher	−28.89 ± 2.27	−41.63 ± 4.49	−81.83 ± 6.49	−25.10 ± 2.47	−27.18 ± 2.17	−27.58 ± 2.11	−28.64 ± 3.90	−378.41 ± 69.64	−39.61 ± 14.01	**−21.73 ± 1.49**	−28.31 ± 1.95	−317.43 ± 109.06	−32.02 ± 0.61
Swimmer	96.15 ± 4.97	**128.54 ± 36.60**	26.51 ± 3.08	104.46 ± 6.65	89.10 ± 6.17	96.81 ± 3.07	91.73 ± 3.75	69.42 ± 7.12	108.99 ± 2.63	71.79 ± 6.44	89.26 ± 6.39	48.34 ± 4.34	41.98 ± 5.46
Walker2d	4748.24 ± 153.48	5868.29 ± 185.50	5972.98 ± 140.06	4920.95 ± 156.97	5198.05 ± 157.41	5305.67 ± 208.10	4756.66 ± 177.00	1068.45 ± 217.04	5145.48 ± 183.24	5112.73 ± 237.73	4373.54 ± 80.11	**6402.65 ± 195.06**	576.14 ± 61.75
Non-Brax environments													
CartPole	465.81 ± 21.42	471.89 ± 22.68	**500.00 ± 0.00**	496.01 ± 2.82	499.02 ± 0.65	488.08 ± 7.28	498.78 ± 0.75	495.19 ± 1.81	498.99 ± 0.36	497.95 ± 0.85	498.77 ± 0.78	491.63 ± 6.47	5.84 ± 0.01
Pendulum	−208.86 ± 26.15	−351.67 ± 118.78	−539.14 ± 83.40	−152.22 ± 2.72	**−147.68 ± 1.73**	−194.14 ± 19.03	−148.79 ± 2.30	−207.41 ± 5.54	−153.11 ± 2.45	−148.34 ± 2.62	−170.76 ± 8.80	−229.55 ± 30.20	−149.94 ± 2.25
Craftax Classic	10.39 ± 0.11	**10.51 ± 0.06**	8.65 ± 0.07	10.09 ± 0.04	9.80 ± 0.07	9.98 ± 0.06	8.71 ± 0.06	6.27 ± 0.05	9.87 ± 0.08	9.50 ± 0.14	8.89 ± 0.06	6.37 ± 0.12	−0.90 ± 0.00
Craftax Full	9.08 ± 0.11	**9.52 ± 0.10**	6.38 ± 0.06	9.03 ± 0.07	7.56 ± 0.10	7.85 ± 0.07	7.04 ± 0.04	4.71 ± 0.08	8.41 ± 0.07	8.40 ± 0.10	4.96 ± 0.04	2.17 ± 0.14	−0.90 ± 0.00
Jumanji 2048	9373.42 ± 138.24	**9806.19 ± 149.63**	9695.99 ± 236.57	9031.95 ± 115.22	8633.51 ± 112.67	9506.67 ± 113.20	3212.89 ± 321.55	1295.92 ± 19.23	5329.65 ± 227.88	7107.58 ± 816.80	9493.13 ± 108.02	9243.44 ± 91.83	147.47 ± 4.13
Jumanji Knapsack	18.89 ± 0.06	**19.02 ± 0.03**	18.15 ± 0.07	18.92 ± 0.06	18.44 ± 0.07	18.51 ± 0.04	18.89 ± 0.04	18.88 ± 0.05	18.81 ± 0.04	18.03 ± 0.06	18.28 ± 0.08	18.61 ± 0.06	0.49 ± 0.01
Jumanji Snake	**23.34 ± 1.84**	21.97 ± 2.70	18.68 ± 2.05	21.73 ± 1.70	19.90 ± 1.00	19.32 ± 1.54	16.50 ± 0.14	0.74 ± 0.30	19.26 ± 1.72	7.41 ± 1.01	13.62 ± 1.80	21.11 ± 2.79	0.04 ± 0.01
Kinetix (L) Hard Pinball	1.215 ± 0.001	1.215 ± 0.002	1.213 ± 0.002	1.213 ± 0.002	1.215 ± 0.001	**1.219 ± 0.001**	1.214 ± 0.000	1.194 ± 0.010	1.214 ± 0.000	1.217 ± 0.001	1.205 ± 0.004	−0.782 ± 0.130	0.259 ± 0.161
Kinetix (M) Thrust Control Left	0.868 ± 0.177	0.343 ± 0.156	0.576 ± 0.192	0.810 ± 0.191	0.108 ± 0.000	0.108 ± 0.000	0.155 ± 0.047	0.105 ± 0.001	0.107 ± 0.000	0.808 ± 0.192	**1.285 ± 0.001**	0.049 ± 0.017	0.704 ± 0.194
Kinetix (S) Thrust Over Ball	0.178 ± 0.001	0.017 ± 0.092	**0.385 ± 0.158**	0.071 ± 0.099	0.174 ± 0.001	0.174 ± 0.003	0.171 ± 0.001	0.161 ± 0.002	0.177 ± 0.001	0.291 ± 0.115	0.281 ± 0.116	−0.598 ± 0.214	0.256 ± 0.262
Navix DoorKey 8x8	0.931 ± 0.015	0.834 ± 0.038	0.998 ± 0.002	0.922 ± 0.024	0.890 ± 0.023	0.820 ± 0.015	**0.999 ± 0.001**	0.958 ± 0.017	0.975 ± 0.017	0.846 ± 0.016	0.933 ± 0.010	0.926 ± 0.029	0.000 ± 0.000
Navix Dynamic Obstacles	0.649 ± 0.016	0.653 ± 0.011	**0.677 ± 0.015**	0.654 ± 0.014	0.661 ± 0.015	0.657 ± 0.015	0.655 ± 0.013	0.655 ± 0.013	0.661 ± 0.015	0.598 ± 0.016	0.639 ± 0.015	0.645 ± 0.016	0.000 ± 0.000
Navix FourRooms	0.484 ± 0.011	**0.551 ± 0.018**	0.173 ± 0.009	0.502 ± 0.017	0.415 ± 0.012	0.428 ± 0.018	0.219 ± 0.009	0.020 ± 0.007	0.354 ± 0.012	0.299 ± 0.010	0.361 ± 0.011	0.244 ± 0.012	0.000 ± 0.000

D FUTURE WORK

This work presents several intriguing threads for future work, which we present separately here.

While we have shown that meta-learning works empirically for inaccurate world models, we do not yet have a clear theory explaining why. Developing mathematically grounded theory to understand this further would yield key insights for the field of meta-learning.

Additionally, leveraging the finding that random networks provide a signal for meta-learning update rules could lead to theoretical and empirical findings that significantly advance the field of many-shot meta-RL.

In Section C.5, we demonstrate that meta-learned algorithms can close the world model-to-simulator gap–a subset of the broader sim-to-sim gap, and by extension the sim-to-real gap. This is a promising direction for robotics research, where the sim-to-real gap remains a significant obstacle.

Moreover, while we explore LPG in this work, we do not use the property that world models are *differentiable simulators*. Using this property could help learn better update rules more quickly by letting meta-gradients flow through the world model.

Lastly, methods from Unsupervised Environment Design (UED) as implemented in Jackson et al. (2023) could potentially yield stronger meta-learned algorithms. Additionally, neural networks are easily parallelizable, which enables meta-training across different environments in parallel.

E BROADER IMPACT

This work explores the automated discovery of RL algorithms via meta-learning, with potential applications including the fine-tuning of large language models (LLMs).

On the positive side, discovering more efficient or effective RL algorithms could reduce the computational cost of LLM fine-tuning, lowering the resource barrier for researchers and reducing associated energy consumption.

However, automatically discovered algorithms may be less interpretable than hand-designed ones, potentially compounding existing challenges in understanding and auditing LLM behavior. If such algorithms are adopted in safety-critical fine-tuning pipelines, this opacity could make it harder to identify or correct undesirable model behaviors. We encourage future work to investigate the interpretability of meta-learned optimizers and loss functions alongside their empirical performance.