MODEL-BASED META-LEARNING FOR ALGORITHM DISCOVERY

Theo Wolf[1]* **Jarek Liesen**[1] **Alex Goldie**[1,2] **Uljad Berdica**[1] **Mattie Fellows**[1] **Jakob Foerster**[1]

[1]FLAIR, University of Oxford, [2]WhiRL, University of Oxford

ABSTRACT

Reinforcement Learning (RL) algorithms are typically hand-crafted through a slow and iterative scientific process. While meta-RL promises to automate algorithm discovery, research into meta-RL has been held back by the large computational requirements of simulating environments for meta-training. 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. We show that MBML matches the performance of traditional online meta-RL at a fraction of the time and compute, without needing perfect world models. By substantially reducing the computational cost of meta-training, MBML lowers the barrier to entry for meta-RL research and enables algorithm discovery at a larger scale.

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" Schmidhuber (1987); Finn et al. (2017); Clune (2020); Beck et al. (2023). It aligns with the dominant paradigm in machine learning: scaling computation and data ultimately outperforms methods informed by human knowledge Sutton (2019). The approach of using meta-learning to automatically discover new algorithms shows great promise for the wider field of machine learning Chen et al. (2023); Metz et al. (2022) and more specifically for reinforcement learning (RL) Houthooft et al. (2018); Lu et al. (2022); Oh et al. (2021; 2025). Many components of typical RL pipelines can be replaced by small, meta-learned neural networks, including the loss function (Houthooft et al., 2018; Lu et al., 2022; Jackson et al., 2024), optimizer (Goldie et al., 2025a), or the entire gradient update mechanism (Oh et al., 2021; 2025).

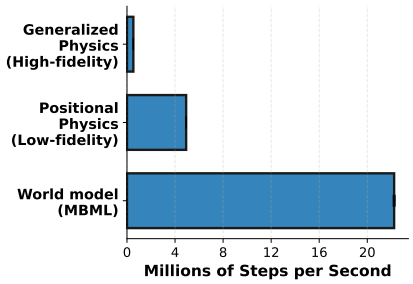


Figure 1: Our new meta-learning method, MBML, introduces a major shift for the field by leveraging offline data. Our method yields novel algorithms up to $42\times$ faster than current online methods while matching their performance—breaking a fundamental barrier of algorithmic discovery.

However, the computational budget required for meta-RL is typically several orders of magnitude higher than for training a single RL agent, and frequently requires simulating hundreds of environments in parallel (e.g., Oh et al. (2025)). These computational limitations restrict the diversity and complexity of environments, slowing the process of scientific iteration. As a result, prior work evaluates meta-learning algorithms on a restricted set of environments and on a single seed for meta-training (meta-seed), limiting their generalization claims Lu et al. (2022); Jackson et al. (2024).

In this work, we address the computational cost of meta-RL by introducing *Model-Based Meta-Learning* (MBML), a framework that replaces environment simulators with *world models*: neural networks that predict next states and rewards. As modern GPUs are highly optimized for parallel

*Main & corresponding author: theo@robots.ox.ac.uk

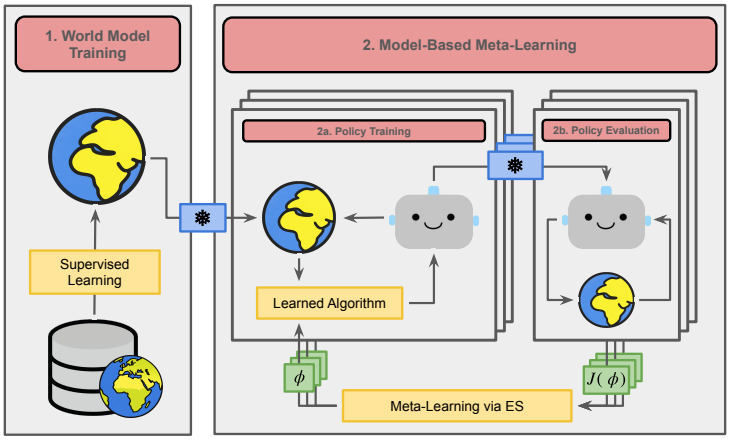


Figure 2: Our Model-Based Meta-Learning framework efficiently discovers novel algorithms. By utilizing world models (left), many policies are trained in parallel (center), then evaluated in the world model (right). The returns are maximized using ES. This approach leads to significant speed-ups and removes the need for handcrafted environments, introducing a new paradigm for algorithm discovery.

neural network inference (Lee et al., 2025), this approach yields substantial speed-ups during meta-training. Training these world models amounts to a negligible fraction of the computational cost of meta-training. We further find that models do not need to perfectly replicate environment dynamics for successful algorithm discovery.

Importantly, MBML is not a meta-RL algorithm, but a general approach that can be applied to existing meta-RL methods. We refer to the component responsible for environment execution, either a true simulator or learned world models, as the *backend*. We demonstrate MBML's flexibility and validity by using world models as a backend for three existing meta-RL algorithms: LPO (Lu et al., 2022), TA-LPO (Jackson et al., 2024), and OPEN (Goldie et al., 2025a). Algorithms discovered with the world model backend are competitive with those discovered using a full-fidelity backend, at a $42\times$ reduction in wall-clock time.

To illustrate the kind of research that MBML unlocks, we develop a new meta-RL algorithm as a small case-study: *Meta-Learning Optimizer and Loss* (MLOL) combines TA-LPO and OPEN (Jackson et al., 2024; Goldie et al., 2025a) to discover algorithms with a meta-learned optimizer and loss function. By meta-training MLOL with an MBML backend trained on common D4RL datasets Fu et al. (2021), we discover two meta-learned RL algorithms from different dataset sizes: *Learned Optimizer & Loss* on 2 and 6 million transitions (LOL-2M and LOL-6M). These learned algorithms transfer well to environments that are semantically similar to the meta-training data distribution (continuous control) and semantically different (procedurally generated games).

Our contributions are the following:

1. We propose MBML, **a novel, offline meta-learning framework** that achieves up to a $42\times$ speed-up in wall-clock meta-time without sacrificing performance (Section 6.1).
2. We demonstrate that **accurate world models are not needed** for MBML to discover performant algorithms (Section 6.2).
3. We use MBML to **design a new meta-learning algorithm** that combines and extends on prior work in meta-RL (Section 7).
4. We publish an open-source codebase that provides building blocks for researchers to build meta-RL pipelines, including MBML[1].

[1] `https://github.com/TheodoreWolf/MetaRL`

2 BACKGROUND

2.1 REINFORCEMENT LEARNING

We consider a *Markov Decision Process* (MDP) $\mathcal{M} = (\mathcal{S}_0, \mathcal{S}, \mathcal{A}, T, R, H)$. $\mathcal{S}_0$ is the initial state distribution. $\mathcal{S}, \mathcal{A}$ 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 $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \to \mathbb{R}$. H is the trajectory length or horizon. In RL, we are interested in finding a policy π that maximizes the expected return

$$J(\pi) = \mathbb{E}_{a_{0:H} \sim \pi, s_0 \sim \mathcal{S}_0, s_{1:H} \sim H}[\sum_{t=0}^{H} R(s_t, a_t)]. \tag{1}$$

We define an *algorithm* as a function $\mathcal{F} : \mathcal{M} \mapsto \pi$ that maps from an MDP to a policy π, aiming to maximize $J(\pi)$. The algorithm may be parameterized with parameters 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 parametrized by θ. This is typically trained with supervised learning and is commonly referred to as a *world model* (Ha & Schmidhuber, 2018). World models are often imperfect representations of the true environment, and policies trained in world models transfer poorly (Jackson et al., 2025). This problem is widely known as the *Sim-to-Real Gap* (Da et al., 2025).

Online and Offline We use the terminology set out in Levine et al. (2020). We consider online RL to be whenever the algorithm is interacting with the true environment of interest to learn a policy. Offline RL is whenever a dataset of static data is used to learn the policy.

2.2 EVOLUTIONARY STRATEGIES (ES)

Evolution Strategies are black-box optimization methods that use batches of function evaluations to estimate a gradient or search direction (Li et al., 2020). OpenES (Salimans et al., 2017) is commonly used for meta-optimization in meta-reinforcement learning (Houthooft et al., 2018; Lu et al., 2022; Jackson et al., 2024; Goldie et al., 2025a). Gradient estimates improve with larger batch sizes, and can be fed to standard machine learning optimizers such as Adam (Kingma & Ba, 2017).

2.3 PRIOR META-LEARNING 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. To reduce meta-training steps, the drift function of LPO is initialized as that of Proximal Policy Optimization (Schulman et al., 2017, PPO) drift function.

TA-LPO Jackson et al. (2024) builds on this for *Temporally-Aware LPO* (TA-LPO), which adds information about the training progress into the feature augmented vector while still obeying the constraints of drift functions. They find that conditioning the policy loss on information about the training progress enables the agent to change its behavior throughout training such as lowering the entropy of the policy over time.

OPEN Goldie et al. (2025a) meta-learns an optimization algorithm for RL that replaces conventional gradient-based optimizers like SGD or Adam (Kingma & Ba, 2017). OPEN uses conditioning features and an augmented output expression to overcome certain pathologies of RL training, such as different moving average timescales (Kingma & Ba, 2017), neuron dormancy (Sokar et al., 2023), and, similarly to TA-LPO, how far training is through its total horizon.

3 PROBLEM SETTING

In meta-RL, we are generally interested in maximizing the average return over a set of MDPs. One means to do this is to learn an RL algorithm $\mathcal{F}$, which outputs a policy π given access to an MDP (real or world model); $\mathcal{F}(\mathcal{M}) = \pi$. Formally for N MDPs $\mathcal{M}_1, \ldots, \mathcal{M}_N$,

$$\mathcal{F}^* = \arg\max_{\mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} J_z(\mathcal{F}(\mathcal{M}_i)), \tag{2}$$

where J_z is a normalized return that allows for aggregation across different MDPs. This is different from classical RL, where the objective is to maximize the return of a policy in a single MDP,

$$\pi^* = \arg\max_{\pi \in \Pi} J(\pi). \tag{3}$$

In the black-box meta-learning setting, we can parametrize the algorithm from Equation (2) with meta-parameters ϕ; we therefore get

$$\phi^* = \arg\max_{\phi} \frac{1}{N} \sum_{i}^{N} J_z(\mathcal{F}_\phi(\mathcal{M}^i)). \tag{4}$$

Previous attempts to optimize this objective use either evolution (Houthooft et al., 2018; Lu et al., 2022; Jackson et al., 2024; Goldie et al., 2025a), or meta-gradients (Oh et al., 2021; Jackson et al., 2023; Oh et al., 2025). We make a slight modification to this setting to be more practical: assuming no access to any online environment at meta-training time. For this, we approximate the MDP with a world model: $\widehat{\mathcal{M}}_\theta = \{\mathcal{S}_0, \mathcal{S}, \mathcal{A}, T_\theta, R_\theta, H\}$. Where the transition function T and the reward function R are parametrized by θ. We therefore are optimizing the following objective

$$\phi^* = \arg\max_{\phi} \frac{1}{N} \sum_{i}^{N} J_z(\mathcal{F}_\phi(\widehat{\mathcal{M}}^i_\theta)). \tag{5}$$

This is a more practical case of the objective Equation (4) as it allows us to leverage available offline RL data, and not rely on hand-crafted hardware-accelerated simulators. Despite this, a distributional mismatch between the world model and the true MDP may lead to pathological behavior in the meta-learned algorithm.

4 METHOD

Our method (Figure 2) has three main steps:

1. **Train a world model** (Figure 2, left) 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 world model, using perturbed meta-parameters.

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

4.1 TRAINING WORLD MODELS

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

$$f^i_\theta(s_t, a_t) = [\mu^i_{s_{t+1}}, \sigma^i_{s_{t+1}}, \mu^i_{R_t} \sigma^i_{R_t}],$$

where each network has a different parameter initialization. This ensemble enables the world model to estimate both epistemic and aleatoric 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

$$s_{t+1} \sim \mathcal{N}\Big(\mu^i_{s_{t+1}}, (\sigma^i_{s_{t+1}})^2\Big)$$
$$R_t \sim \mathcal{N}\Big(\mu^i_{R_t}, (\sigma^i_{R_t})^2\Big).$$

We note that while this parameterization is specific to continuous states and actions, MBML allows using any implementation of world models.

As is common in implementations of these ensemble models, we use the same initial state distribution and termination function as the ground-truth environment (Sun, 2023; Jackson et al., 2025). In addition, we adopt the ensemble-disagreement termination criterion of Kidambi et al. (2021) (details in Section B.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.

4.2 Meta-learning RL algorithms

Policy training Given a world model, we can train agents using standard RL methods, including PPO or, in our case, the parameterized algorithm we are trying to optimize. Before policy training, create a batch of perturbations for the meta-parameters ϕ by adding Gaussian noise $\phi + \epsilon\sigma$ with $\epsilon \in \mathbb{R}$ and $\sigma \sim \mathcal{N}(0, I)$. This allows for meta-gradient estimation in the next step. We then train a policy on each instance of the algorithm. We give additional details about training world models in Section B.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[2]. The gradient $\nabla_\phi J(\phi)$ is then estimated using

$$\nabla_\phi J(\phi) = \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)} \left[\frac{\epsilon}{\sigma} J(\phi + \sigma\epsilon) \right], \tag{6}$$

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.

4.3 Evaluating the meta-learned RL algorithm

For evaluation, we freeze the meta-parameters of the meta-learned algorithm. We then train a policy in an environment of interest using the same methodology as was used to train the policies in the world model.

5 Experimental Setup

Datasets We use the D4RL datasets (Fu et al., 2021), which are commonly used by the offline RL community (Jackson et al., 2025). We do this to ensure reproducibility and to avoid introducing additional confounding factors associated with data collection and tuning. We use three locomotion datasets: *halfcheetah-medium-expert*, *hopper-medium-expert*, and *walker2d-medium-expert*, as these are collected in environments that are available in Brax (Freeman et al., 2021). We use Brax as it is hardware-accelerated, and we can thus directly compare MBML performance to online meta-training. We use the *medium-expert* datasets, which combine data collected from two behavioral policies and therefore provide broader coverage of the state space.

Comparing Environment Implementations Brax has multiple physics backends; we compare our world model backend with two of them:. First, the *generalized* backend is a reimplementation of the original MuJoCo simulator, and provides high fidelity at slow simulation speeds. Secondly, the *positional* backend improves performance by compromising on fidelity. It was previously used

[2]For convenience, we write $J(\phi) = J(\mathcal{F}_\phi(\mathcal{M}))$.

Table 1: Summary of meta-training and evaluation 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)
Evaluation backend	Generalized (Brax, high-fidelity)
# Seeds	Meta-training runs: 5 Policy training runs: 8 Policy evaluations per run: 128
Stop criterion	100 generations **OR** 24h wall-clock time

for meta-training by Lu et al. (2022); Jackson et al. (2024); Goldie et al. (2025a). More information about Brax backends can be found in Section C.1.

Meta-learning Algorithms We meta-train LPO Lu et al. (2022), TA-LPO Jackson et al. (2024), OPEN Goldie et al. (2025a) on three different environments, using both two different 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 detail is in Section C.3.2.

6 Results of MBML

We demonstrate two core results:

1. MBML serves as a computationally efficient alternative to traditional meta-RL, without sacrificing performance (Section 6.1).
2. MBML remains effective even when using imperfect world models (Section 6.2).

6.1 MBML is efficient and effective

Meta-training results Figure 3 shows that meta-learning curves of meta-learning algorithms using different backends. We evaluate every learned algorithm using the generalized backend in the environment, probing transferability from the positional or world model backend. For each meta-learning algorithm (rows), transfer from the positional (orange) and world model backend (green) leads to similar performance across environments (columns). Unsurprisingly, meta-training on the generalized backend (blue) achieves the highest performance.

Baseline We compare to a PPO baseline with hyperparameters from cleanRL (Huang et al., 2021, see Section C.3.2), that has been trained for the same number of steps we use for one training run of a learned algorithm. We highlight that within 100 generations, methods meta-trained on the positional or world model backend struggle to consistently outperform PPO. We note that this does not contradict prior literature's claims of substantial improvements over PPO, since authors usually meta-train and test on the positional backend only. Analogously, meta-training and testing on the generalized backend does lead to slight improvements in our benchmarks. We therefore posit that the meta-RL algorithms we investigate have limited capacity to generalize.

Runtime efficiency Figure 4 demonstrates that MBML obtains between $25\times$ and $42\times$ speed-up when compared to meta-training on the generalized (high-fidelity) environment directly. Training the world model is approximately two orders of magnitude faster in wall-clock time than meta-training. For example, training a world model for halfcheetah on an H200 takes around 642 seconds, while the subsequent meta-training LPO with MBML takes around 2:01h.

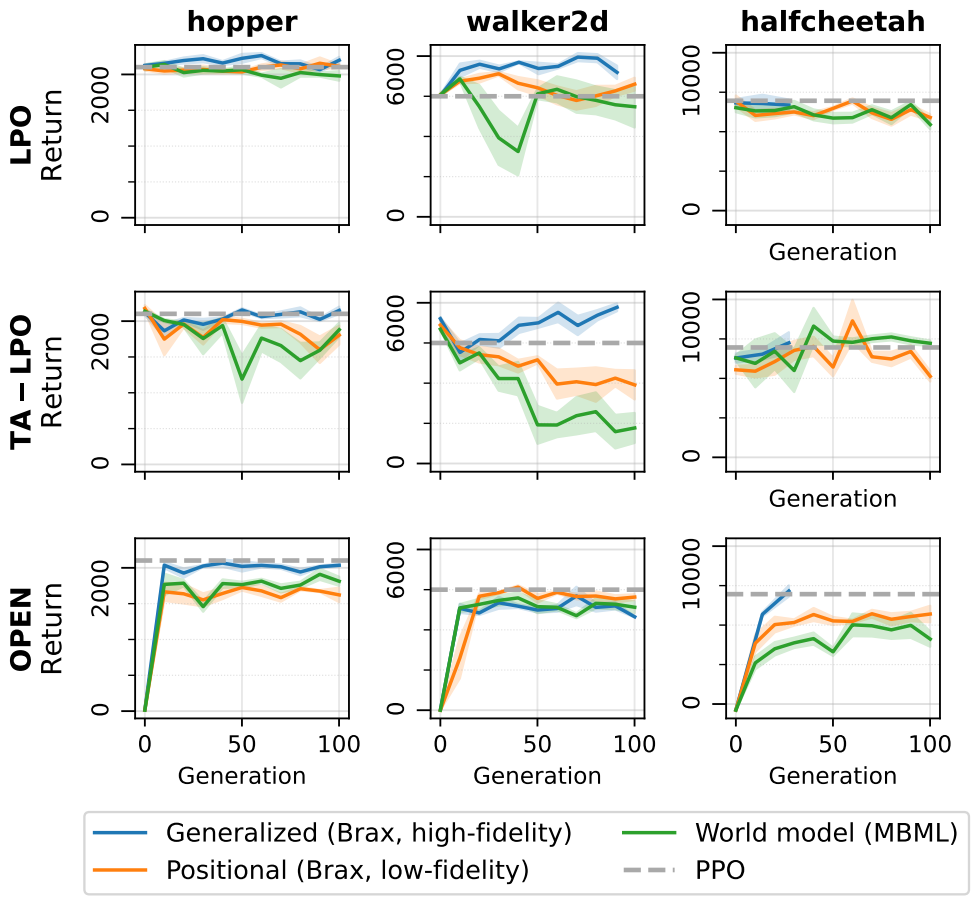


Figure 3: Our offline meta-training method (green) is competitive with online meta-training performance. MBML almost matches the final return in the high-fidelity generalized environments during meta-training. This holds across three environments and meta-training algorithms. Note that *Halfcheetah* and *Walker2d* (generalized) terminated after only 25 and 90 generations, due to the 24 hours of wall-clock time limit for our experiments.

Additionally, we were able to reuse the same three world models for all meta-training, rendering world-model training as nothing but a negligible upfront cost. In combination with the performance of MBML-trained algorithms, these results suggest that MBML is a scalable alternative to meta-training with online simulators. Additionally, we show that similar speed-ups are possible on consumer-grade hardware in **??**.

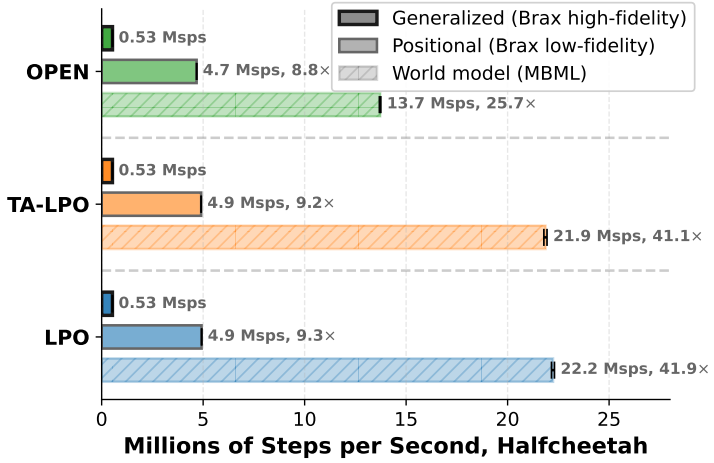


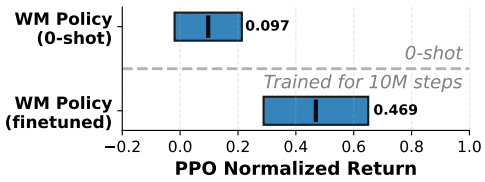
Figure 4: MBML is *up to 42 times faster* than comparable online meta-learning methods, achieving up to 22.3 million steps per second during meta-training on *halfcheetah*.

6.2 MBML WORKS WITH IMPERFECT WORLD MODELS

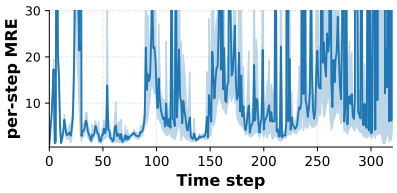
We have made design decisions for our world models that, while lending themselves well to parallel meta-training, are not optimized for accuracy. Figure 5a shows that policies trained by PPO in the world model do not transfer to the generalized backend. Even after significant fine-tuning, they do not reach even 50% of the performance of policies trained by PPO in the generalized backend directly. This is not surprising: our world model training is inspired by common offline RL methods (namely Yu et al., 2020; Kidambi et al., 2021), that use world model rollouts as data augmentation instead of an environment replacement.

Additionally, the world model produces transitions that are significantly different from the true physics backend. For instance, Figure 5b shows that the mean relative error between transitions and world model predictions diverges.

Furthermore, in Figure 14 of the appendix, we use a world model with significantly higher validation loss, and show that the meta-learned algorithm achieves similar generalization performance.



(a) 0-shot transfer of policies trained in the world models for 50M time steps to the full-fidelity environment. Training these policies for 10M more steps still yields policies that are worse than training for 10M steps from random.



(b) 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.

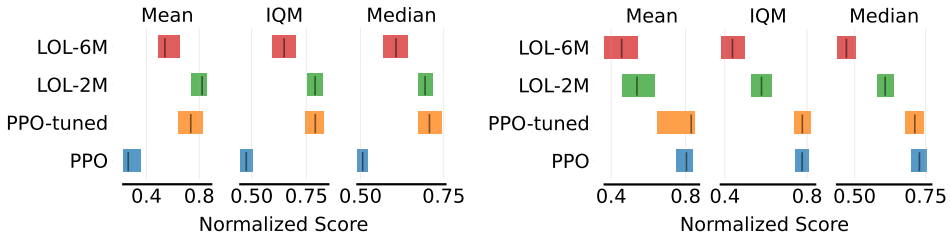
7 CASE STUDY: UTILIZING MBML

Designing a new meta-learning method To highlight the kind of research that MBML, as a general approach in combination with our unified software environment, unlocks, we next present a case study that uses these to push the boundaries of Meta-RL. We investigate *Meta-Learning Optimizer & Loss* (MLOL), a novel meta-RL algorithm that merges contributions of prior methods and combines them with MBML. Concretely, we extend PPO by simultaneously meta-learning a temporally-aware loss function (Jackson et al., 2024), an RL-specific optimizer (Goldie et al., 2025a) bootstrapped from `adamW` (Loshchilov & Hutter, 2019), and hyperparameter values (for more detail see Section C.3.2). Note that while we could use any backend, MBML accelerates meta-training with MLOL by more than $25\times$. We additionally experiment with meta-training using *multiple world models*. We describe our new meta-learning algorithm in detail in Section B.2.

Using MLOL, we meta-train an algorithm on a world model of *halfcheetah*, which we call LOL-2M. Additionally, we use MLOL with all three world model environments, *hopper*, *walker2d*, and *halfcheetah*, to produce LOL-6M. Experimental details can be found in Section C.

Baseline We baseline against Proximal Policy Optimization (Schulman et al., 2017, PPO) with hyperparameters adapted from Huang et al. (2021); Lu et al. (2022). An additional baseline is given by tuned PPO, for which we tune hyperparameters in the halfcheetah world model using the same budget as we have for meta-training. We give the full list of PPO hyperparameters we optimize in Section C.3.2.

Evaluating meta-learned algorithms We evaluate the algorithms on the full Brax suite with the exception of *halfcheetah*, which we use as a meta-validation environment to choose the best learned algorithm out of 5 meta-seeds. Additionally, we also evaluate in a semantically diverse set of environments that are outside the meta-training data distribution: gridworlds in Navix (Pignatelli et al., 2024), procedurally generated long-context games in Craftax (Matthews et al., 2024), 2d



(a) Brax (b) Non-Brax

Figure 6: LOL-2M is competitive with a PPO tuned in the world model for Brax environments, but is outperformed by PPO tuned and untuned in other environments. 95% bootstrapped confidence intervals over 10 seeds. Full per-environment returns in Section D.6.

physics simulation in Kinetix (Matthews et al., 2025), and diverse puzzles and games in Jumanji (Bonnet et al., 2024). We provide the full list of environments in Table 7.

We see in Figure 6a that LOL-2M and LOL-6M outperform PPO with base hyperparameters when evaluated on the full brax suite. However, PPO with tuned hyperparameters (PPO-tuned) outperforms LOL-6M, while showing similar performance to LOL-2M. Outside of Brax, Figure 6b shows that LOL algorithms lag behind PPO and PPO-tuned, showcasing that these algorithms do not generalize as well outside their meta-training distribution. Notably, the learned algorithms outperform baselines in some gridworld and game environments, but perform much worse than baselines in *Craftax* environments. The full distribution of returns is in Section D.6.

8 Discussion: Recommendations for meta-RL practitioners

We offer some recommendations for practitioners who wish to develop new methods for algorithm discovery:

- Use meta-seeds to iterate on meta-learning algorithm design, while this was previously infeasible for practitioners (as mentioned in Goldie et al. (2025b)), MBML enables this.
- Use one or more meta-validation environments to ensure the methods transfer to domains of interest, making sure to remove such environments from *meta-test* evaluations.
- Evaluate on environments that are out-of-distribution in final evaluations.
- Attempt to give baselines an equivalent number of steps for hyperparameter tuning as used for meta-training.

9 Conclusion

In this work, we introduced Model-Based Meta-Learning (MBML), a new paradigm for discovering RL algorithms using world models that is up to $42\times$ faster than meta-training on a full-fidelity physics simulator, without sacrificing on performance. Notably, we find that there is no requirement for the world models to be perfect to obtain performant algorithms. Using MBML, we develop a novel meta-learning algorithm, Meta Learning Optimizer & Loss (MLOL). Meta-training MLOL with MBML enables the discovery of high performing algorithms that can generalize far outside their meta-training distribution.

Overall, MBML enables researchers to leverage world models trained on available offline data and not rely on slow, handcrafted environments for meta-training. This allows lowers the entry barrier for meta-RL research, and enables algorithm discovery at a larger scale.

CONTRIBUTIONS

TW led the project; wrote the code, ran the experiments, and helped in writing the paper. **JL** wrote the majority of the paper and helped with designing experiments. **AG** provided close supervision, and helped write the paper. **UB** and **MF** provided supervision and assistance in writing. **JF** provided supervision and computational resources.

ACKNOWLEDGMENTS

TW and **JL** are funded by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems EP/Y035070/1. **AG** is funded by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems EP/S024050/1. **JL** is funded by Sony Interactive Entertainment Europe Ltd. **UB** is supported by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems and the Rhodes Scholarship. **MF** is funded by a generous grant from the UKRI Engineering and Physical Sciences Research Council EP/Y028481/1. **JF** is partially funded by the UKRI grant EP/Y028481/1 (originally selected for funding by the ERC). This project received compute resources from a gracious grant provided by the Isambard-AI National AI Research Resource. We additionally thank the other FLAIRies for stimulating discussions, with special mention to Matthew Jackson, Chris Lu, Alistair Letcher and Eltayeb Ahmed who all gave invaluable advice during the project.

REFERENCES

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.

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I. Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in jax, 2024. URL https://arxiv.org/abs/2306.09884.

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.

Takuya Hiraoka, Takahisa Imagawa, Voot Tangkaratt, Takayuki Osa, Takashi Onishi, and Yoshimasa Tsuruoka. Meta-model-based meta-policy optimization, 2021. URL https://arxiv.org/abs/2006.02608.

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.

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.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020. URL https://arxiv.org/abs/2005.01643.

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.

Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. Regularization matters in policy optimization, 2021. URL https://arxiv.org/abs/1910.09191.

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.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning, 2024. URL https://arxiv.org/abs/2402.16801.

Michael Matthews, Michael Beukman, Chris Lu, and Jakob Foerster. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks, 2025. URL https://arxiv.org/abs/2410.23208.

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.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning, March 2018. tex.archiveprefix: arXiv tex.eprintclass: cs.LG.

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.

Eduardo Pignatelli, Jarek Liesen, Robert Tjarko Lange, Chris Lu, Pablo Samuel Castro, and Laura Toni. Navix: Scaling minigrid environments with jax, 2024. URL https://arxiv.org/abs/2407.19396.

Zohar Rimon, Tom Jurgenson, Orr Krupnik, Gilad Adler, and Aviv Tamar. MAMBA: An effective world model approach for meta-reinforcement learning, March 2024. tex.archiveprefix: arXiv tex.eprintclass: cs.LG.

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.

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.

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.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: A very good method for Bayes-Adaptive Deep RL via meta-learning, October 2019. tex.archiveprefix: arXiv tex.eprintclass: cs.LG.

A Related Works

A.1 Model-Based RL

An RL method is called model-based if the agent acquires a predictive model of the true environment (Sutton & Barto, 2018), traditionally used for planning and to improve the online policy in a more sample-efficient manner. Ha & Schmidhuber (2018) extend the use of models to learning both the state representations and transition functions in which the agent can fully learn a policy that transfers to the original environment. These models ares commonly referred to as *world models*. The Dreamer line of work (Hafner et al., 2023; 2025) further refines the concept by using recurrent state-space models that achieve competitive results in both continuous control and discrete settings. In model-based offline RL, accuracy is enforced through pessimistic halting states (Kidambi et al., 2021) or uncertainty penalties (Yu et al., 2020). While we take inspiration from these works, we find that our meta-training framework method benefits from the sample efficiency of world models without strict accuracy requirements that are needed for policy transfer.

A.2 Meta-RL

Meta-RL methods aim to enable agents to rapidly adapt to new tasks by leveraging experience across a distribution of environments. Early approaches like RL^2 (Duan et al., 2016) formulate meta-RL as implicitly learning an update via recurrent policies, allowing adaptation through hidden state dynamics rather than explicit parameter updates. Gradient-based methods like MAML (Finn et al., 2017) learn initial parameters that can be efficiently adapted to new tasks using a few gradient steps, and have been successfully applied to continuous-control RL settings. Probabilistic approaches, such as VariBAD (Zintgraf et al., 2019), introduce latent task inference by learning a belief over environment dynamics from past transitions, enabling structured and sample-efficient adaptation in partially observable settings. Despite their different mechanisms, these methods typically rely on meta-training over a fixed task distribution, and their generalization performance can degrade under distributional shift. We explicitly evaluate performance in out-of-distribution environments and find that our method remains relatively robust in these environments.

A.3 Model-Based Meta-RL

Nagabandi et al. (2018) successfully combines meta-RL with model-based methods, modeling the dynamics of the environment and quickly adapting to future behavior using past behavior. Hiraoka et al. (2021) extends performance guarantee theorems from model-based RL to the meta-RL setting. They achieve high sample efficiency and performance on continuous tasks, as well as stronger adaptation to unseen tasks. More recently, Rimon et al. (2024) combines the Dreamer architecture (Hafner et al., 2023) with meta-RL, improving generalization and sample efficiency in unconventional environments. Unlike these works, we are interested in discovering new *algorithms* using models of environments.

A.4 Meta-RL for Algorithm Discovery

Houthooft et al. (2018) introduces a loss function that is meta-learned using Evolutionary Strategies (ES) (Salimans et al., 2017). This is then expanded in Lu et al. (2022), by enforcing theoretical constraints on the learned loss function. Jackson et al. (2024) augment this learned loss with information about the training progression, which improves the learned algorithm's performance. This insight is then leveraged in Goldie et al. (2025a), where instead of a loss function, the authors design a training-aware bespoke RL optimizer, meta-learned with ES. We use the last three works for our experiments and expand upon them in Section 2.3.

Another line of work, Oh et al. (2021) uses meta-gradients instead of ES for meta-training *Learned Policy Gradient* (LPG). Follow-up work has since scaled meta-training to a large distribution of environments (Oh et al., 2025). In this work, we perform meta-optimization entirely within the learned world models, significantly reducing step sample complexity and obtaining considerable speedups.

B Extended Methods

B.1 World Models

B.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. We observe that meta-learned algorithms learn to avoid this penalty throughout meta-training in Figure 12.

B.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. 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 C.3.1.

B.2 Meta-Learning Optimizer & Loss

We design a new meta-learning method that is a generalization of prior works: *Meta-Learning Optimizer & Loss* (MLOL).

B.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 MLOL's drift function still obeys the constraints of Mirror Learning is in Section B.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, this includes the learning rate lr but also $\beta_1, \beta_2, \epsilon_{\text{adam}}$ and weight regularizer λ (Loshchilov & Hutter, 2019). 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.

B.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. We also do not evolve **network sizes**, **activation functions**, **number of total environment steps**, and **reward and observation normalization toggling**. 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 C.3.2.

B.2.3 Proof of validity of Drift Function

Kuba et al. (2024) describes 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 MLOL'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:

$$y(r) = h_{\phi_{\mathfrak{D}}}(\boldsymbol{x}_{r,A,t}) - \xi + \phi_\alpha \,\text{ReLU}\left((r - \text{clip}(r, 1 \pm \phi_\epsilon)) \cdot A\right) \tag{7}$$

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

$$\frac{d}{dr}D(r) = \frac{d}{dy}\text{ReLU}(y(r)) \cdot \frac{d}{dr}y(r), \tag{8}$$

$$= (\mathbb{I}(y(r) > 0) + a\mathbb{I}(y(r) = 0)) \cdot \frac{d}{dr}y(r), \tag{9}$$

(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 (9):

$$\frac{d}{dr}D(r = 1) = 0 \cdot \frac{d}{dr}y(r) = 0 \tag{10}$$

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

B.3 Meta-learning across MDPs

The efficiency gains of MBML allow us to scale meta-training by jointly training over all three world models in a single meta-training run, with an aim to leverage more data, following Oh et al. (2025). To aggregate fitnesses from different world models, we use the normalization values and equation provided by D4RL (Fu et al., 2021):

$$J_z(\pi, \mathcal{M}) = \frac{J(\pi, \mathcal{M}) - J^{\mathcal{M}}_{\text{min}}}{J^{\mathcal{M}}_{\text{max}} - J^{\mathcal{M}}_{\text{min}}}. \tag{11}$$

C Extended Experimental Details

C.1 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 (Lu et al., 2022; Jackson et al., 2024; Goldie et al., 2025a). The reason for this is that it is computationally expensive to run meta-training with *generalized* due to its computational cost 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 backend more recent backend is the *MJX* backend. We found early on in our experiments that MJX backend runs almost twice as slow 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 7)

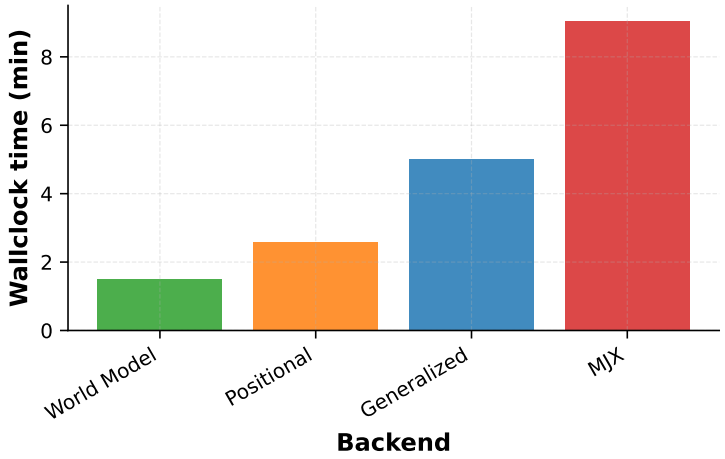


Figure 7: Wallclock time comparison for training a single PPO agent on the hopper environment with 4 different backends and 50M steps. MJX is much slower despite being advertised as "more performant".

C.2 Computational Resources

We run our experiments on a cluster of Nvidia H200 GPUs. For meta-training (online and MBML), we use four H200's in parallel. For meta-testing, we use one H200 per RL training run. Additionally, we use a single RTX 3080Ti for experiments to show that MBML works on consumer-grade hardware.

C.3 Hyperparameters

C.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)

C.3.2 META-TRAINING HYPERPARAMETERS

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

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
MLOL-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

C.4 META-TRAINING

C.4.1 META-TRAINING WITH ONE ENVIRONMENT

Table 6: Meta-Training combinations

Parameter	**Value**
algorithm	[MLOL, 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
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

C.4.2 META-TRAINING LOL-6M

We use the same exact hyperparameters for meta-training with all three world models, except for the number of generations, which we set to 300 to reflect the fact there are three times as many world models over which to meta-train. We show the results of this in Figure 15

C.5 Evaluation Environments

Table 7: Environments used for evaluation, 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

D Extended Results

D.1 World Model Validation

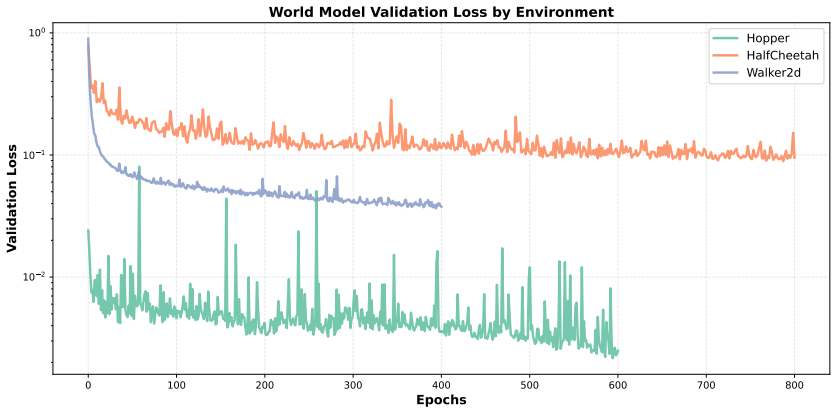


Figure 8: World model validation loss per dataset for final hyperparameter set.

D.2 Policy Transfer

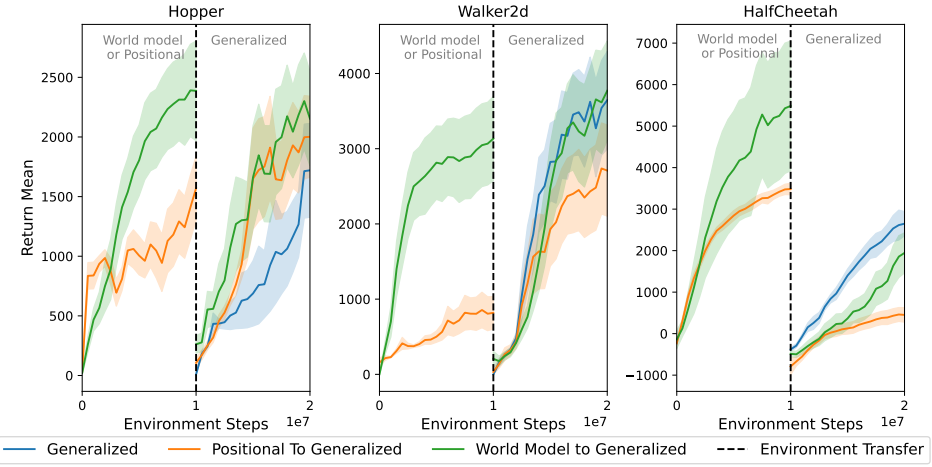


Figure 9: Transferring a policy between backends causes a severe collapse in performance. Training the policy further is only clearly better than training directly from scratch in the *hopper* environment. Average and SEM over 5 seeds.

D.3 META-TRAINING

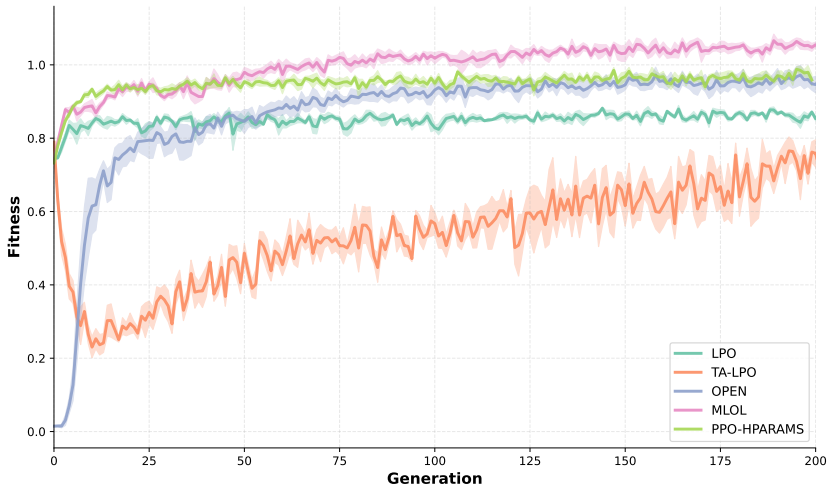


Figure 10: MLOL obtains the highest fitness in the *halfcheetah* world model when meta-training with MBML. Tuning PPO hyperparameters using ES in the world model obtains the second-highest fitness. Average and SEM over 5 meta-seeds.

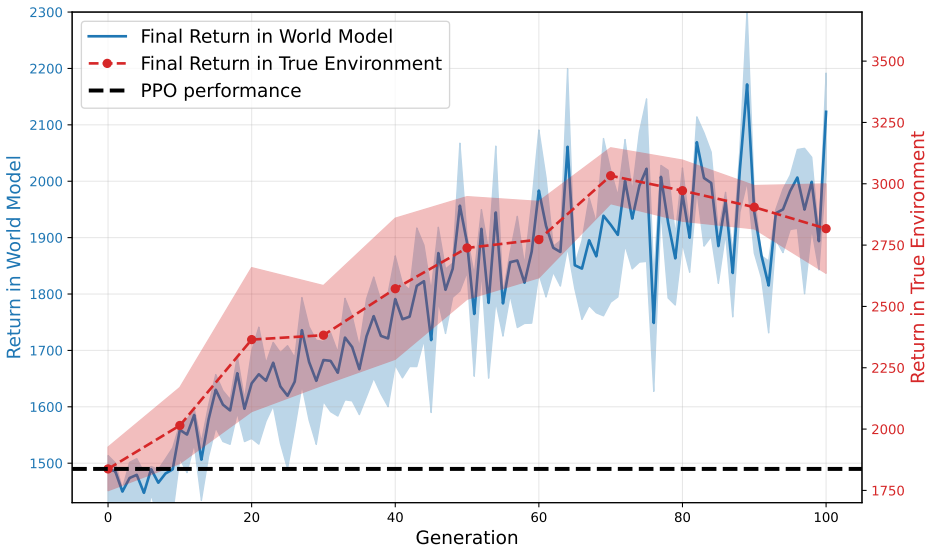


Figure 11: Comparison of final return in the world model and final return in the true environment during meta-training with MLOL on the *hopper* world model. Average and SEM of 5 meta-training seeds. Increasing return in the world model correlates directly to increasing returns in the true environment.

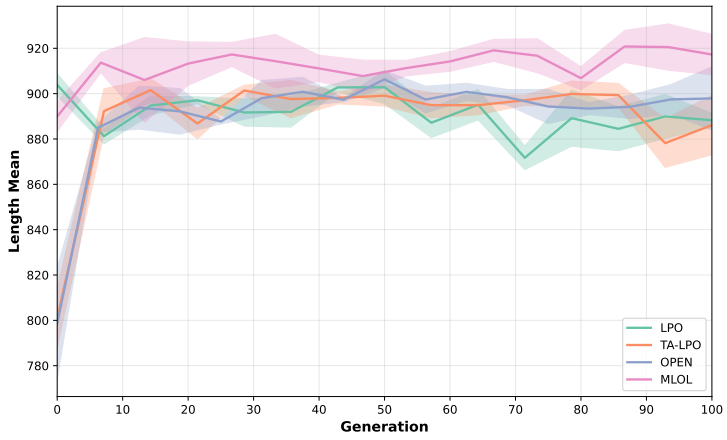


Figure 12: We can observe how early termination changes during meta-training by observing the average episode length for the *halfcheetah* world model. Since the *halfcheetah* environment does not have a termination function by design, any episode shorter than 1000 steps implies the pessimistic MDP termination penalty was applied. The shaded error is the standard error of the mean for 3 meta-training seeds.

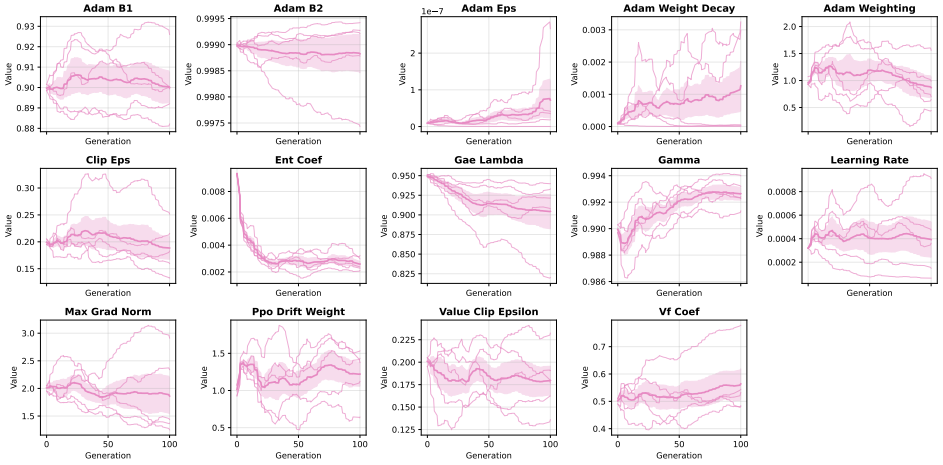


Figure 13: Evolution of hyperparameters during meta-training MLOL on the *halfcheetah* world model. Mean across five meta-training seeds, shaded area is the standard error of the mean. Different seeds show wide variations in hyperparameter values. The meta-optimization learns to decrease entropy during meta-training.

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

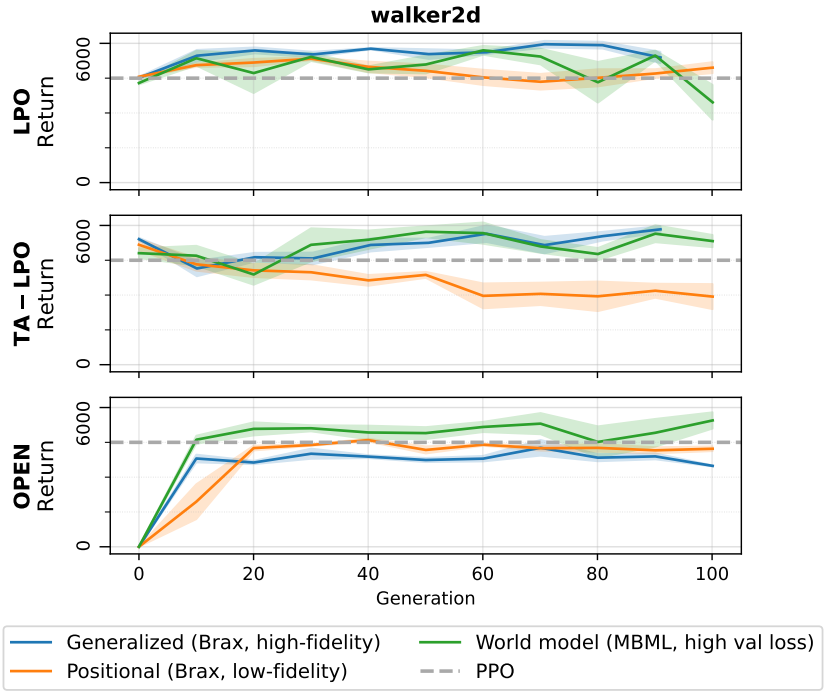


Figure 14: We use a world model from *walker2d-medium-expert* with a validation loss that is higher than the one used previously (0.155 versus 0.032) for meta-training, this suggests that world model accuracy is not the most relevant factor for meta-training good algorithms.

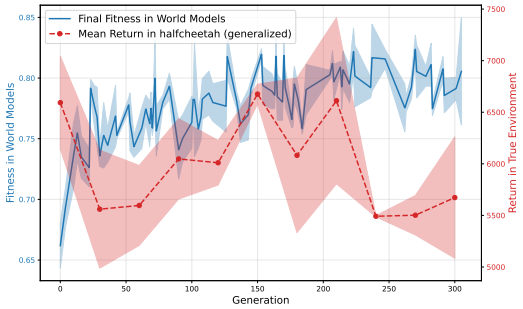


Figure 15: Meta-training LOL-6M does not yield consistently increasing returns in the meta-validation environment. Mean across three meta-seeds.

D.4 MLOL Ablations

(a) Hyperparameter evolution

(b) Population size

Figure 16: Enabling hyperparameter evolution or increasing population size during meta-training leads to qualitatively worse returns in the meta-validation environment after 200 generations of meta-training. Average and SEM over 3 meta-seeds.

D.5 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.

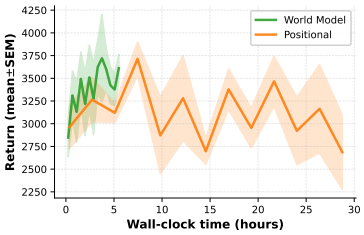


Figure 17: Meta-validation performance on halfcheetah vs. wall-clock time, when meta-training with the world model or positional backend on a singular RTX 3080Ti, using a population size of 32. Each agent is meta-trained for 10M steps rather than 50M.

D.6 ALGORITHM EVALUATION

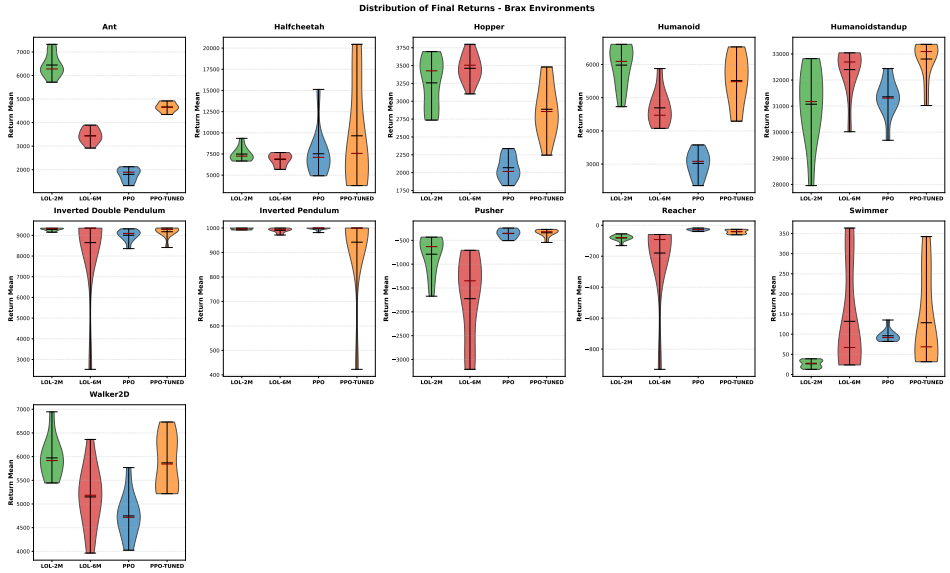


Figure 18: Violin plot showing the distribution of final returns for Brax environments. 10 seeds.

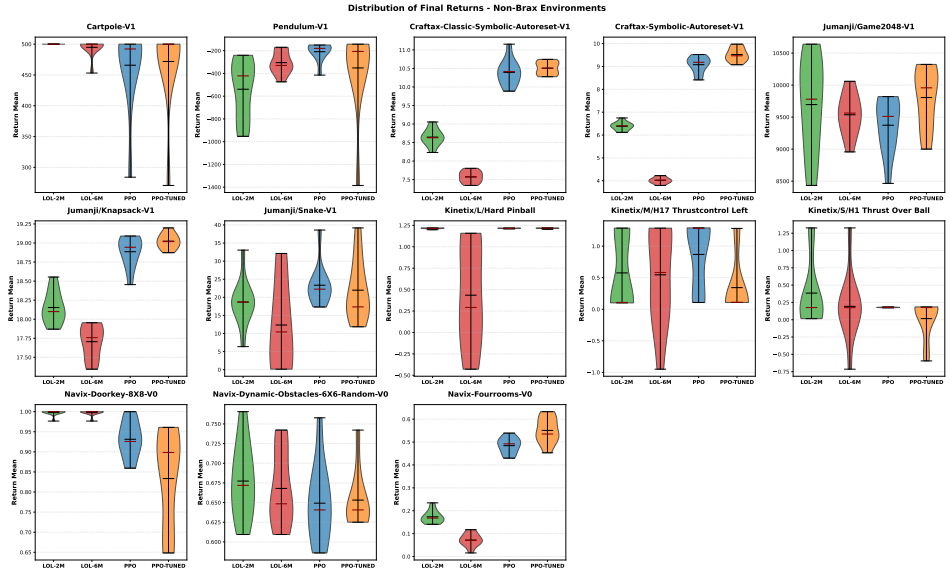


Figure 19: Violin plot showing the distribution of final returns for other environments. 10 seeds.

D.7 Analysis of learned algorithms

Entropy In Figure 20, we repeat an experiment conducted by Lu et al. (2022). We look at the entropy of the policy throughout policy training of PPO and the meta-learned algorithms on the *brax/ant* environment. We observe that PPO's entropy decreases faster than the other algorithms, but plateaus out halfway through training. The learned algorithms, on the other hand, keep the entropy high and then decrease it throughout training. They then have lower entropy than PPO at the end of training. We interpret this as an exploration-exploitation trade-off: the algorithm has learned to explore at the start of training, then to exploit as the training nears its end. This is a finding that was also found in Jackson et al. (2024). Notably, we find that LOL-6M, which was meta-trained on more world models, prefers higher entropy than LOL-2M. This is sensible, as it has previously been shown that MuJoCo tasks with PPO tend not to require much entropy regularization Liu et al. (2021). This also explains why LOL learned algorithms struggle more in other environments where exploration is important throughout training, such as *Craftax* as seen in Figure 19.

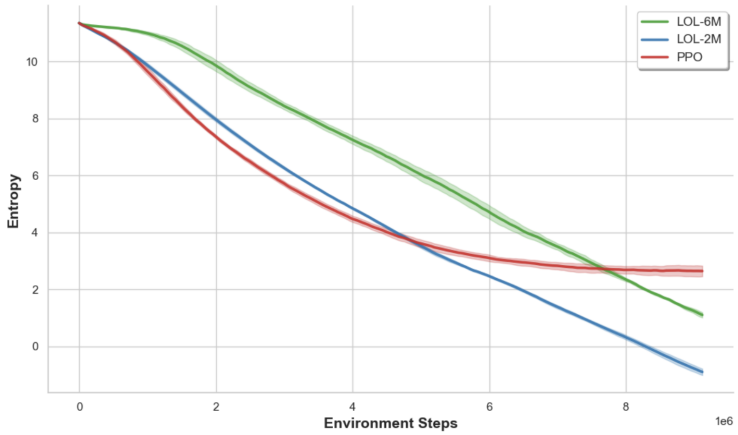


Figure 20: Entropy of PPO and LOL policies throughout training on the *brax/ant* environment. Average and standard error of the mean over 5 seeds.

E ASSETS USED

Here we outline the assets and artifacts used in this work. Everything was generated through our method and open-source libraries.

Table 9: Assets Models used

Asset	License	URL
D4RL Dataset	Apache 2.0	github.com/Farama-Foundation/D4RL
ReJAX	Apache 2.0	github.com/keraJLi/rejax
Navix	Apache 2.0	github.com/epignatelli/navix/
Brax	Apache 2.0	https://github.com/google/brax
Gymnax	Apache 2.0	github.com/RobertTLange/gymnax/
JAX	Apache 2.0	github.com/jax-ml/jax/
Craftax	MIT License	github.com/MichaelTMatthews/Craftax