ELECTRICAL ENGINEERING & COMPUTER SCIENCE
UNIVERSITY OF MICHIGAN

# Few-Shot Preference-Based Reinforcement Learning

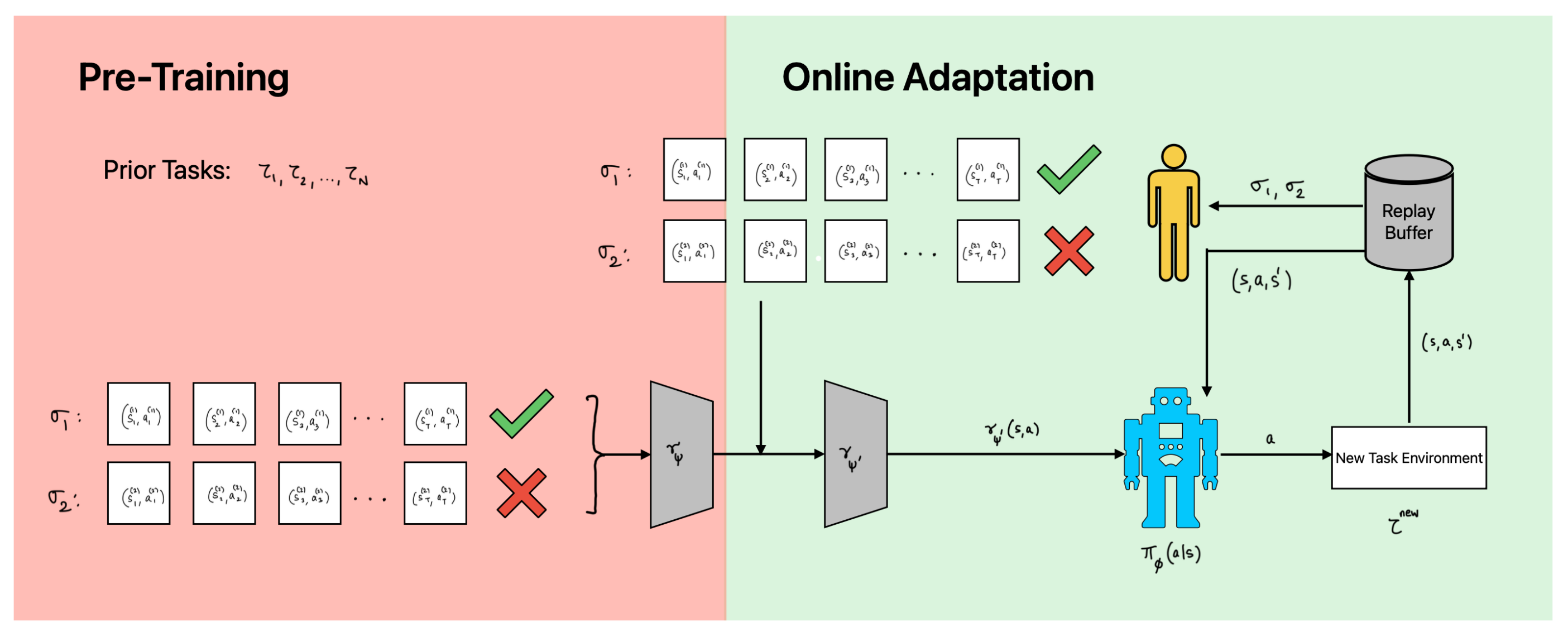
Ishan Kapnadak, Karan Anand, Rishikesh Ksheersagar, Shlok Agarwal, Sudhanshu Agarwal
{kapnadak, karanand, rishiksh, ashlok, sudhagar}@umich.edu
University of Michigan, Ann Arbor

## Introduction

- RL has shown promise in various domains but design of a good reward function remains a critical challenge (especially with large state and action spaces)
- Sparse reward functions do not afford effective learning, and dense reward functions are often susceptible to reward hacking.
- Some recent work has been done on the use of human preferences to align reward functions with human intent.
- These methods require a lot of human-annotated data to perform well – can we do this with minimal burden on humans?

## Set-up

- Given dataset of previous tasks, learn policy for a new task
- Instead of reward regression, we use a preference-based approach with a preference over two partial trajectories $\sigma_1$ and $\sigma_2$
- Use a neural network to predict rewards and then classify segments
- First learn a generalized reward function then adapt to a new task using as few human queries as possible!



## Data Preparation

- We use the MetaWorld environment to generate data
- We use 10 pre-training tasks with 1250 episodes per task
- 39-dimensional observation space, 4-dimensional action space
- Divide each episodes into segments of length 25
- For each task, randomly sample segments and generate comparisons with labels generated according to the preference:

$$y(\sigma_1, \sigma_2) := \begin{cases} 1 & \text{if } \sum_t r(s_t^{(1)}, a_t^{(1)}) > \sum_t r(s_t^{(2)}, a_t^{(2)}), \\ 2 & \text{otherwise.} \end{cases}$$

- For data formatting, the data vector X contains all the segments stacked vertically, and the output vector y contains the cumulative rewards of the corresponding segments in the X vector
- Our neural network takes in a given 43-dimensional state-action pair and tries predicting the reward $\hat{r}_\psi(s,a) \approx r(s,a)$

## Methodology

- Given two partial trajectories $\sigma_1$ and $\sigma_2$, we use Bradley-Terry model for preference prediction:

$$P[\sigma_1 \succ \sigma_2] = \frac{\exp \sum_t \hat{r}_\psi(s_t^1, a_t^1)}{\exp \sum_t \hat{r}_\psi(s_t^1, a_t^1) + \exp \sum_t \hat{r}_\psi(s_t^2, a_t^2)}$$

- With this, the learning task becomes a simple classification task with the following cross-entropy loss function:

$$\mathcal{L}_{\text{pref}}(\psi, \mathcal{D}) = -\mathbb{E}_{(\sigma_1,\sigma_2,y)\sim\mathcal{D}} \left(\mathbb{I}\{y=1\} \log P[\sigma_1 \succ \sigma_2] + \mathbb{I}\{y=2\} \log(1 - P[\sigma_1 \succ \sigma_2])\right)$$

- Our neural network employs a three-linear-layer architecture with an input size of 43 (state + action) and a hidden size of 256
- We use a ReLU activation at the end of each hidden layer, and the final output represents the predicted reward for a given state-action pair

### Model-Agnostic Meta-Learning

- The first model we use is Model-Agnostic Meta-Learning (MAML) which learns a generalized reward function through the following update

$$\psi \leftarrow \psi - \beta \nabla_\psi \sum_{i=1}^{N} \mathcal{L}_{\text{pref}}\left(\psi - \alpha \nabla_\psi \mathcal{L}_{\text{pref}}(\psi, \mathcal{D}_i), \mathcal{D}_i\right)$$

- The generalized reward function is then adapted to a new task using a simple gradient descent update as follows

$$\psi \leftarrow \psi - \alpha \nabla_\psi \mathcal{L}_{\text{pref}}(\psi, \mathcal{D}_{\text{new}}).$$

- We also try out a variant of MAML which we call ***iterated* MAML** where the inner update step is performed multiple times before the outer step
- The final learning framework we try is REPTILE which uses the following update rule for learning the generalized reward function:

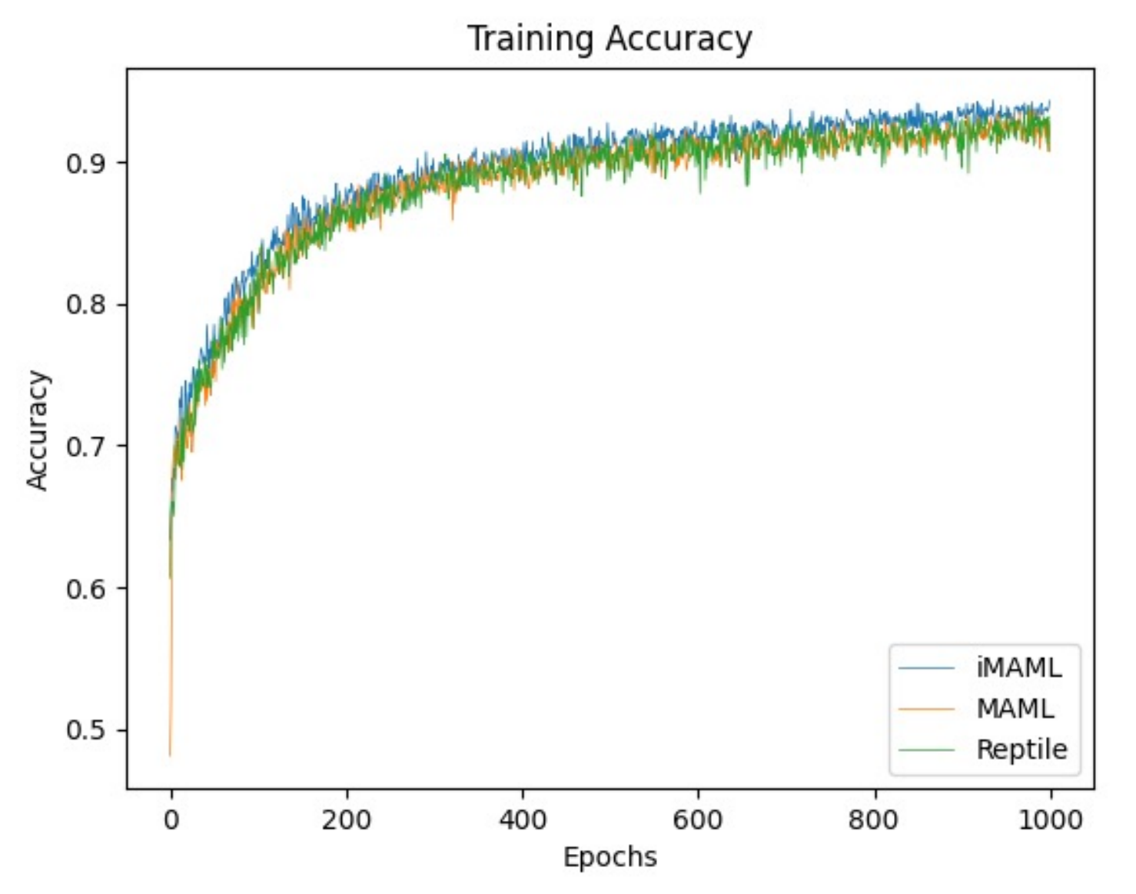
$$\psi \leftarrow \psi - \epsilon \nabla_\psi \sum_{i=1}^{N} \mathcal{L}_{\text{pref}}\left(\psi, \mathcal{D}_i\right)$$

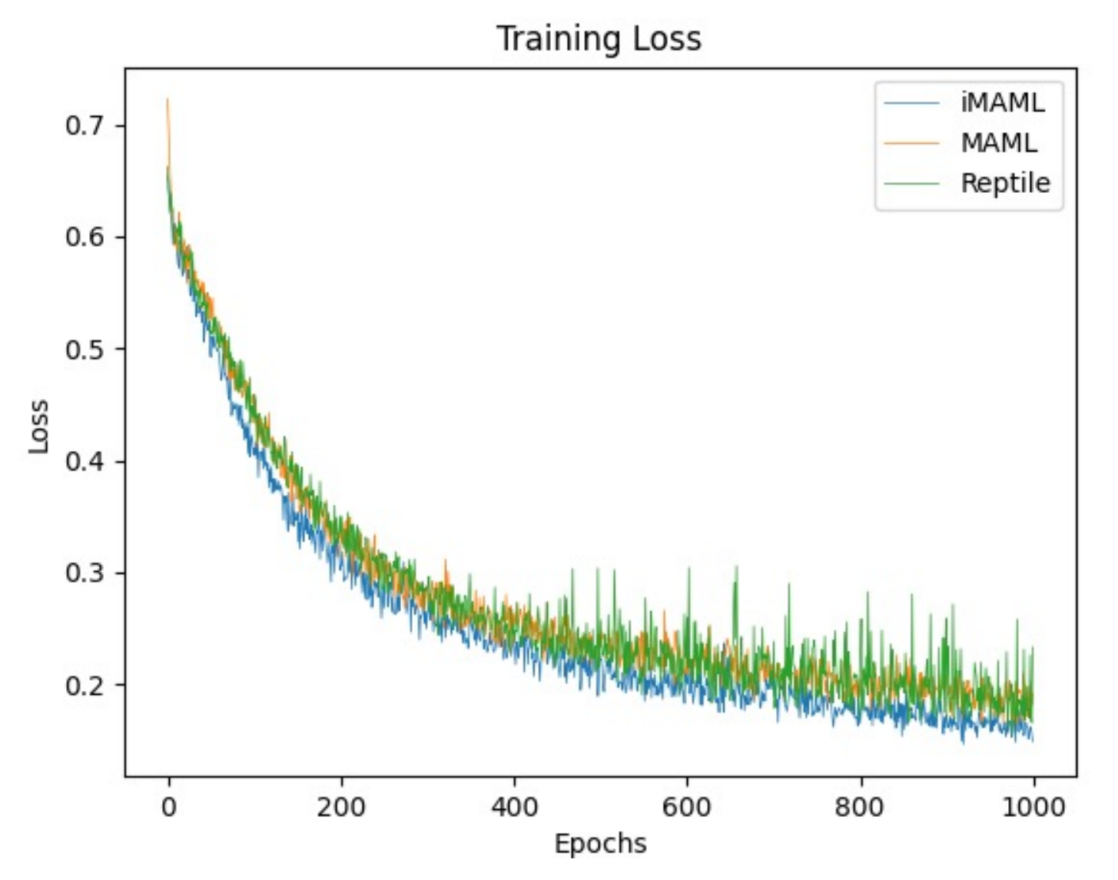
- The adaptation process remains the same and uses gradient descent. In practice, we have used Adam optimizers for faster convergence

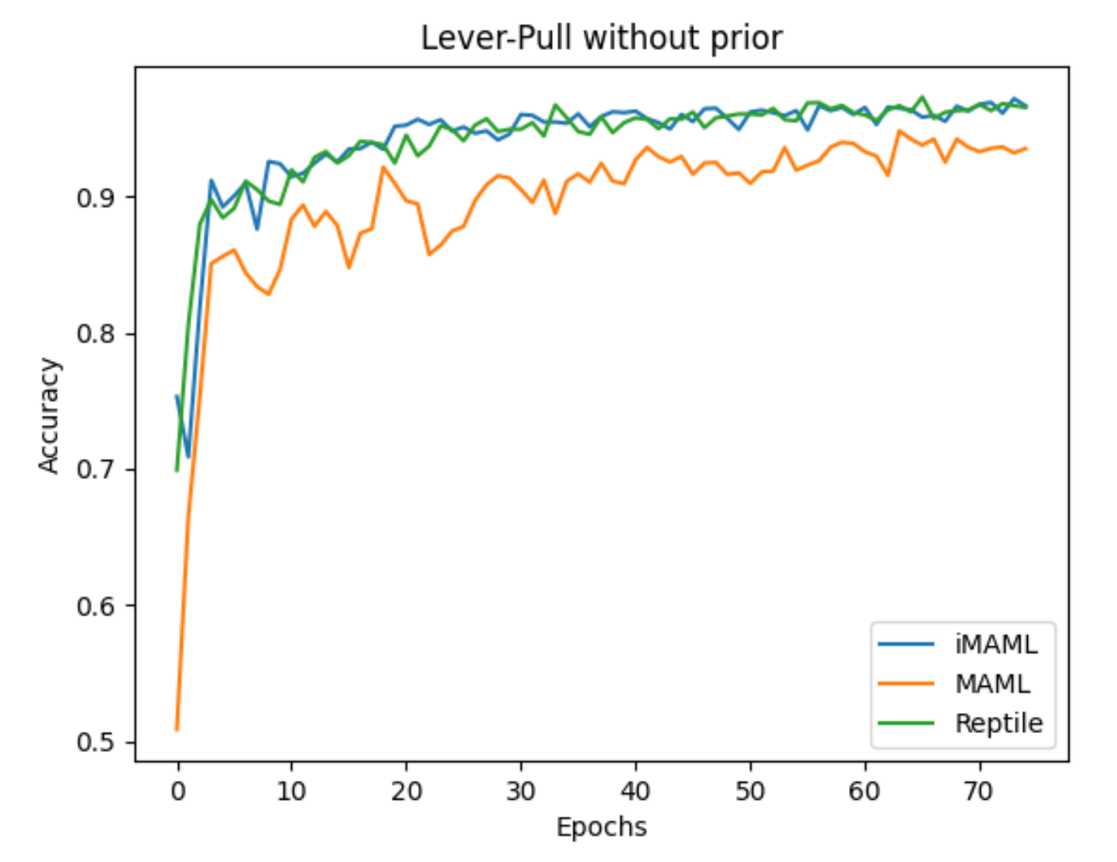
### Prior Policy

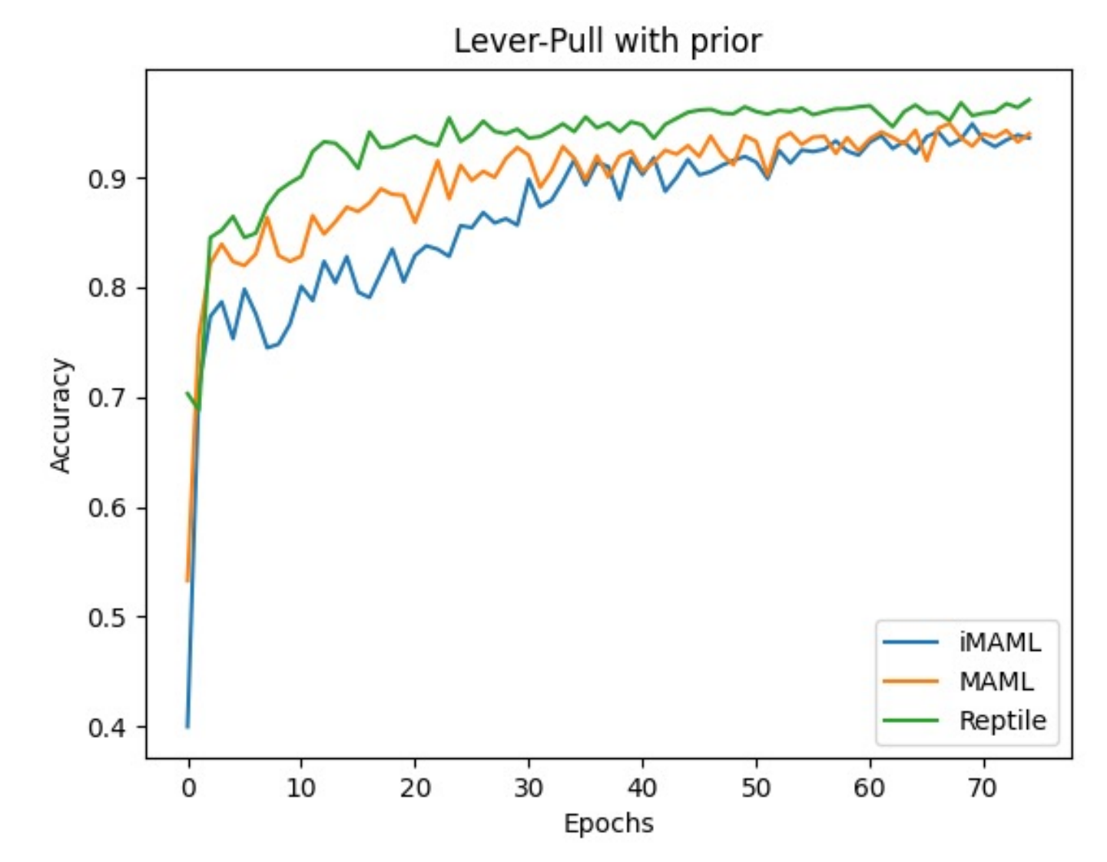
- To improve the adaptation process, we also generated data for the new tasks using a different policy (without noise) than the one already used
- We then use comparisons with segments from the prior policy (noiseless) and segments from the original policies (noisy measurements)
- Our hope is that having a prior policy guiding the learning process will help the model adapt to a new task faster

## Results









## Conclusion & Future Work

- All our models are able to learn a generalized reward function in about 1000 iterations, which adapts to a new task in only about 75 epochs (which validates our few-shot goal!)
- While adding a prior policy slightly improves the performance for Reptile, the same cannot be said for the other two models
- A reweighting of the prior policy might help alleviate this issue
- A better benchmark for our models is the success rate on the new task rather than accuracy of predicting preferred segments – this requires the use of a Soft Actor-Critic algorithm for policy learning
- We plan to learn the policy in the next week, and later, incorporate actual human feedback rather than just oracle feedback
- Should time permit, we also plan to perform a study on how the segment length affects performance and query efficiency

## Significant References

- Joey Hejna and Dorsa Sadigh. *Few-Shot Preference Learning for Human-in-the-Loop RL*. 2022
- Kimin Lee, Laura Smith, and Pieter Abbeel. *PEBBLE: Feedback-Efficient Interactive Reinforcement Learning via Relabeling Experience and Unsupervised Pre-training*. 2021.

COLLEGE OF ENGINEERING
ELECTRICAL ENGINEERING & COMPUTER SCIENCE
UNIVERSITY OF MICHIGAN