# First-Explore, then Exploit: Meta-Learning to Solve Hard Exploration-Exploitation Trade-Offs*

**Ben Norman**[1,2]
btnorman@cs.ubc.ca

**Jeff Clune**[1,2,3]
jclune@gmail.com

[1]Department of Computer Science, University of British Columbia
[2]Vector Institute
[3]Canada CIFAR AI Chair

## Abstract

Standard reinforcement learning (RL) agents never intelligently explore like a human (i.e. taking into account complex domain priors and adapting quickly to previous explorations). Across episodes, RL agents struggle to perform even simple exploration strategies, for example, systematic search that avoids exploring the same location multiple times. Meta-RL is a potential solution, as unlike standard-RL, meta-RL can *learn* to explore. We identify a new challenge with meta-RL that aims to maximize the cumulative reward of an episode sequence (cumulative-reward meta-RL). When the optimal behavior is to sacrifice reward in early episodes for better exploration (and thus enable higher later-episode rewards), existing cumulative-reward meta-RL methods become stuck on the local optima of failing to explore. We introduce a new method, First-Explore, which overcomes this limitation by learning two policies: one to solely explore, and one to solely exploit. When exploring and thus forgoing early-episode reward is required, First-Explore significantly outperforms existing cumulative meta-RL methods. By identifying and solving the previously unrecognized problem of forgoing reward in early episodes, First-Explore represents a significant step towards developing meta-RL algorithms capable of more human-like exploration on a broader range of domains. In complex or open-ended environments, this approach could allow the agent to develop sophisticated exploration heuristics that mimic intrinsic motivations (e.g., prioritizing seeking novel observations).

## Shortened Version Note

This paper is a shortened version submitted to the Intrinsically Motivated Open-ended Learning workshop. For more details, please refer to the full and updated version at `https://arxiv.org/abs/2307.02276`.

## 1 Introduction

Meta-RL [1–5] trains an agent to operate over a sequence of episodes $\tau_1, \ldots \tau_n$. In every episode $\tau_i$, the agent is able to make use of the observations, rewards, and actions of the previous episodes in the sequence $\tau_1, \ldots, \tau_{i-1}$. There are several different approaches to meta-RL. This paper focuses on cumulative-reward meta-RL: *cumulative-reward meta-RL* aims to optimize the sequence of episodes so as to maximize the sum of expected episode returns $\mathbb{E}[\sum_{i=1}^{n} G(\tau_i)]$, where the episode return $G(\tau_i)$

*A more comprehensive version of this paper is available at `https://arxiv.org/abs/2307.02276`.

Accepted at the Intrinsically Motivated Open-ended Learning workshop at NeurIPS 2024.

is the cumulative reward of episode $\tau_i$. To maximize this sum, the agent should optimally balance exploration and exploitation across the sequence, e.g., prioritizing exploration in early episodes so as to better exploit in later ones.

Cumulative reward meta-RL has an unrecognized failure mode, where state-of-the-art (SOTA) methods achieve low cumulative-reward regardless of how long they are trained. This dynamic occurs in domains with the following properties: **A**. Maximizing the expected total reward requires exploratory actions that forgo immediate reward, and **B**. The benefit of these exploratory actions *only occurs* when they are *reliably* followed by good exploitation (i.e., if exploitation is too inconsistent, then exploration results in *lower* total reward).

An example is a bandit domain where the first arm always provides a reward that is better than the average arm, but not the highest possible. To maximize cumulative reward over many pulls, the agent must explore the other arms, and then repeatedly exploit the best one. Property **A** holds, as this optimal policy forgoes immediate reward by not sampling the first arm and its above-average reward. Property **B** holds, as exploration (sampling arms other than the first) is *only valuable* when it is followed by sufficiently consistent exploitation (reliably re-sampling the best arm).

The failure occurs as follows: 1. At the start of training, the agent, being randomly initialized, lacks the ability to *reliably exploit* learned information. 2. As a result, the domain properties **A** and **B** cause exploratory actions to lead to lower total reward than the other actions do. 3. This lower reward trains the agent to *actively avoid exploration*. 4. This avoidance then locks the agent into poor performance, as it cannot learn effective exploitation without exploration. This process occurs in the bandit example. Initially, the agent cannot exploit (e.g., when it finds the best arm it does not reliably resample it). The associated negative expectation of exploration then trains the agent to *avoid exploration* by only sampling the first arm, with its above-average, but sub-optimal, arm reward.

Current SOTA meta-RL algorithms such as $RL^2$ [4, 5], VariBAD [2], and HyperX [3] attempt to train a single policy to maximize the cumulative reward of the whole episode sequence. This optimization causes step 3 of the above-described failure process, and thus these methods suffer from the issue of failing to properly learn (e.g., converging rapidly to a policy of not exploring), which we demonstrate on multiple domains (Section 3). The issue is especially insidious because distributions of simple environments (each trivially solved by standard-RL) can stymie these methods. Surprisingly, domains such as bandits can be too hard for SOTA meta-RL.

We introduce a new approach, First-Explore (visualized in Fig. 1), which overcomes this problem associated with directly optimizing for cumulative reward. Rather than training a single policy to maximize cumulative reward, First-Explore learns two policies: an exploit policy, and an explore policy. The exploit policy maximizes episode return, without attempting to explore. In contrast, the explore policy explores to best inform the exploit policy, without attempting to maximize its own episode return. Only *after training* are the two policies combined to achieve high cumulative reward.

Because the explore policy is trained solely to inform the exploit policy, poor current exploitation no longer causes immediate rewards (property **A**) to *actively discourage* exploration. This change eliminates step 3 of the failure process, and enables First-Explore to perform well in domains where SOTA meta-RL methods fail. By identifying and solving this previously unrecognized issue, First-Explore represents a substantial contribution to meta-RL, paving the way for human-like exploration on a broader range of domains.

## 2 First-Explore

First-Explore trains two transformer [6] policies, one policy to explore and one policy to exploit. Individually neither policy can achieve high cumulative-reward, but *after* weight-update training the policies are combined to produce an inference policy that achieves high cumulative-reward. Fig. 1 visualizes this process. By not *directly* training a single policy to maximize the total reward of the whole episode sequence, First-Explore avoids the failure mode of earlier approaches.

The **explore policy** $\pi_{\text{explore}}$ performs successive episodes, at each point (in each episode) having access to a context of all previous actions, observations, and rewards in the exploration-episode sequence. The **exploit policy** $\pi_{\text{exploit}}$ is trained to take the context $c$ from the explore policy that has explored for a number of episodes between 1 and $n$, and to then run one further episode of

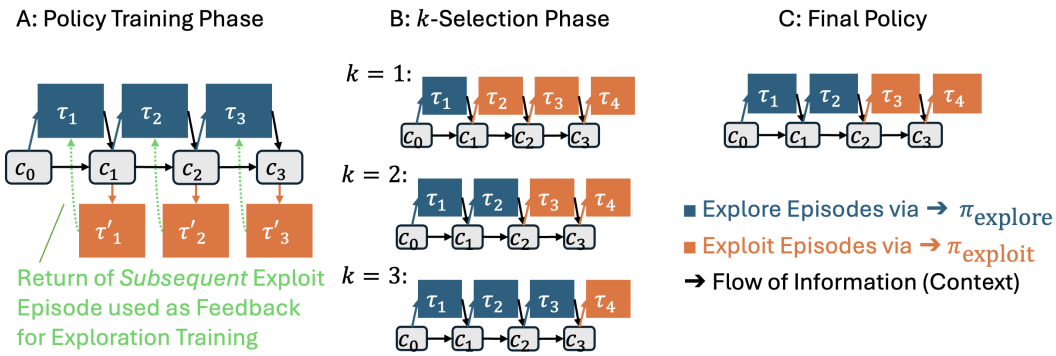



Figure 1: **First-Explore** aims to maximize the cumulative reward of a sequence of $n$ episodes on a target distribution, by first training two separate policies, and then combining them to maximize cumulative reward. **A:** First, two separate policies are trained on the distribution of environments: one to **explore** (produce informative episodes) and one to **exploit** (obtain high reward). The exploit policy $\pi_{\text{exploit}}$ maps $\rightarrow$ the current context $c_i = \tau_1, \ldots, \tau_i$ to an exploit episode $\tau'_i \sim \pi_{\text{exploit}} \mid \tau_1, \ldots \tau_i$. The policy is trained to maximize reward. The explore policy $\pi_{\text{explore}}$ maps $\rightarrow$ the current context $c_i$ to an exploration episode $\tau_{i+1} \sim \pi_{\text{explore}} \mid \tau_1, \ldots, \tau_i$. The exploit policy aims to maximize reward each episode. The explore policy aims to explore so as to maximally increase the rewards of the *subsequent* exploit policy episode. Both policies have access to a context of all previous explorations (visualized with arrows). **B:** After the two policies are trained in the previous phase, the optimal number of explorations is estimated by selecting the number of explorations $k$ that lead to highest mean cumulative-episode reward on a large batch of new (unseen) environments sampled from the target distribution. For each potential value of $k$, the exploration policy performs $k$ sequential rollouts, then the exploitation policy performs the remaining $n - k$ rollouts. **C:** Once $k$ is selected, the two policies are composed to produce the inference policy, which then always performs the first $k$ episodes with the explore policy and the remaining $n - k$ episodes with the exploit policy.

exploitation. During this episode, the exploit policy can also see in context the actions, observations and rewards that have occurred so far in the current episode.

Each policy is incentivized differently. The **explore policy** is incentivized to produce episodes that each, when added to the current context, result in maximally increasing the reward of the *subsequent* exploit policy episode. The **exploit policy** is instead incentivized to produce episodes that each have the highest reward (based on the current context). Training the exploit policy requires context from the explore policy, and training the explore policy requires the rewards of subsequent exploits. This is efficiently achieved by interleaving the policies as depicted in Fig. 1-A (with each rollout from the explore policy followed by a rollout from the exploit policy).

Once the two policies are trained, they are then composed to create the **inference policy** $\pi_{\text{inference}}$. The inference policy first explores with the explore policy for a set number $k$ of episodes. It then switches to exploiting with the exploit policy. $k$ is determined by evaluating each choice of $k$ on batches of new (unseen) environments sampled from the target distribution, selecting the $k$ that leads to highest mean cumulative reward (see Appendix K for an example). $k$ is not a hyperparameter as unlike hyperparameters, the majority of the training compute-expenditure is on policy weight updates that are performed before $k$ is chosen, with the subsequent selection of $k$ only requiring minimal compute.

$$\pi_{\text{inference}} = \begin{cases} \pi_{\text{explore}}, & \text{if episode \#} \leq k \\ \pi_{\text{exploit}}, & \text{otherwise} \end{cases}$$

## 3 Experiments

First-Explore was evaluated (Fig. 2) on three different domains (see Appendix D for details). In each domain, existing SOTA cumulative-reward meta-RL algorithms performed poorly (becoming stuck on a local optimum of not exploring well) when effective exploration required forgoing reward. Even

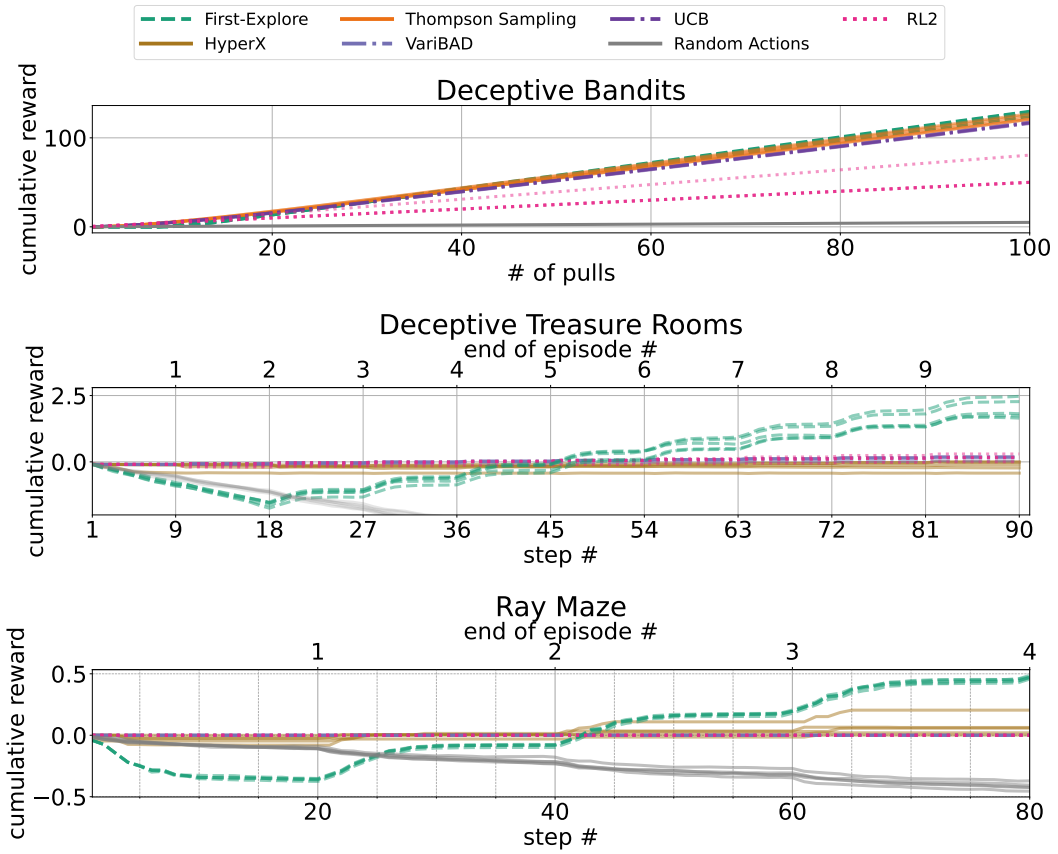



Figure 2: On three different domains, First-Explore significantly outperforms meta-RL controls. For each domain, multiple training runs are plotted superimposed, so as to faithfully represent the variance between runs, and **Random Actions (gray)** is also included to provide an additional baseline for each domain.

trivial seeming environments (bandits) flummox SOTA meta-RL methods. First-Explore escapes these pitfalls and so outperforms all controls, on these challenging deceptive settings.

**Bandit Domain (Fig. 2-Top):** in this bandit [7] domain there is an arm guaranteed to always give reward 0.5, and this reward is greater than the expected reward of an unseen arm. This dynamic stymies **$RL^2$ [4, 5] (fuchsia)** performance, as four of five $RL^2$ training runs (overlaid bold fuchsia) learning to only sample the guaranteed reward (achieving 50 reward after 100 pulls), the remaining run (faint fuchsia) does better but still poorly. The best returns are achieved by methods that consistently explore to find the most rewarding arm, and then exploit that knowledge. **First-Explore (green)** and the non-meta-RL bandit algorithms, **Thompson Sampling [8] (orange)** and **UCB [4] (purple)**, both exhibit this behavior, although impressively, First-Explore achieves marginally greater reward than even the specialized bandit algorithms despite being applicable to more general domains. HyperX and VariBAD are not included in this plot because their performance was overly poor even on non-deceptive versions of this domain. **Deceptive Treasure Room (Fig. 2-Middle):** First-Explore massively outperforms the meta-RL algorithms **$RL^2$ (fuchsia)**, **VariBAD (pale lilac)** and **HyperX (brown)**, achieving significantly more cumulative reward ($>$ 10x) in a ten-episode setting. The meta-RL controls are unable to learn in such a deceptive environment. **Ray Maze (Fig. 2-Bottom):** on the significantly more complex Ray Maze domain, First-Explore similarly achieves significantly more reward than the meta-RL controls ($>$ 7x), with the controls failing to perform well.

## 4 Discussion

What makes the above environments deceptive to standard cumulative-reward meta-RL? To illustrate, we will consider a simplified version of the bandit domain. Each bandit has ten arms. In each episode the agent performs a single action $a \in [1, 10]$. If the first arm is selected then the reward is always $0.5$, and otherwise the reward is an associated arm value $v_a$. The arm means are distributed normally $v_{a\in[2-10]} \sim N(0, 1)$. They are specified when the environment is sampled, and do not change through repeated episodes in the same environment. In the first episode of a newly sampled bandit, expected reward is maximized by pulling the first arm (as other arms give expected reward 0). However, maximizing future reward requires finding a different and more rewarding arm ($v_a > 0.5$) to enable exploiting the found arm repeatedly for higher payoff.

The issue is that, on a newly sampled environment, maximizing the expected cumulative-reward of the pulls requires *consistent* exploitation: exploring in early episodes by sampling arms $2 - 10$ is only useful if followed by reliably resampling the arm that gave the highest rewards. For example, if the agent correctly exploits (chooses the arm with highest expected reward based on past pulls) only half the time then the reduction in reward from early exploration (repeatedly not obtaining arm 1's guaranteed 0.5 reward) may not be worth it, as half the potential exploitation upside is being missed. Thus, the optimal exploration is only worthwhile if the agent is *sufficiently* skilled at exploitation.

The need for consistent exploitation prevents existing methods from learning optimal cumulative-reward behavior. At the start of meta-RL training, the agent is unskilled at exploitation, and thus the optimal exploration is punished (recall that pulling arms 2-10 have a lower expected immediate payoff than arm 1). This punishment trains the agent to minimize exploration. However, once the agent learns to avoid exploration, it can become impossible for the agent to learn good exploitation (as good exploitation often requires having explored). The lack of good exploitation then continues to punish exploration. The resulting dynamic locks the agent into the local optima [7] of exploring and exploiting poorly, regardless of how long the methods are trained (see Appendix H for an example).

The central issue is that cumulative reward is readily deceptive. A reward is *deceptive* if there are local optima such that all small changes to the current policy decrease reward. Because RL and meta-RL train via a sequence of small changes (with changes only made if they increase average reward), these local optima trap the policy and prevent learning the optimal behavior. This paper highlights there are many problems, like these bandits, that are only deceptive (and thus challenging to solve) in the cumulative-reward meta-RL setting, with standard-RL algorithms able to easily solve all environments in the distribution. Returning to the bandit example, in the standard-RL context, each episode and environment is optimized individually, and thus the agent is always in a state where optimal action (i.e., pulling the correct bandit arm) is only a single action away (making finding and reinforcing the global optima trivial).

## 5 Conclusion

We have discovered and shown that seemingly trivial problems such as the Meta-RL-Deceiving Bandits (Fig. 1 & detailed in Appendix D.1) can deceive cumulative-reward meta-RL. To overcome such deception, we introduce a novel meta-RL framework, First-Explore, that learns two separate interleaved policies (one to first explore, another to then exploit). By combining the policies at inference time, First-Explore is able to explore effectively and achieve high cumulative reward on problems that hamstring SOTA methods.

Meta-RL shows the promise of finally fixing the main problem in RL – that it is extremely sample inefficient – even producing *human-level sample efficiency* [9]. However, the promise of this approach is limited, as we have shown, on a large set of important problems. We can only take advantage of this approach if we can harness the benefits of meta-RL even on such problems, and First-Explore enables us to do so, thus offering an important and exciting opportunity for the field.

## References

[1] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

[2] 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. In *International Conference on Learning Representation (ICLR)*, 2020.

[3] Luisa Zintgraf, Leo Feng, Cong Lu, Maximilian Igl, Kristian Hartikainen, Katja Hofmann, and Shimon Whiteson. Exploration in approximate hyper-state space for meta reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2021.

[4] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. $rl^2$: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[5] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[6] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[7] Richard S. Sutton, Francis Bach, and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press Ltd, 2018.

[8] William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933. ISSN 00063444. URL `http://www.jstor.org/stable/2332286`.

[9] Adaptive Agent Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[12] Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, et al. In-context reinforcement learning with algorithm distillation. *arXiv preprint arXiv:2210.14215*, 2022.

[13] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

[14] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[15] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[16] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

[17] Jin Zhang, Jianhao Wang, Hao Hu, Tong Chen, Yingfeng Chen, Changjie Fan, and Chongjie Zhang. Metacure: Meta reinforcement learning with empowerment-driven exploration. In *International Conference on Machine Learning*, pages 12600–12610. PMLR, 2021.

[18] Wenxuan Zhou, Lerrel Pinto, and Abhinav Gupta. Environment probing interaction policies. *arXiv preprint arXiv:1907.11740*, 2019.

[19] Haotian Fu, Hongyao Tang, Jianye Hao, Chen Chen, Xidong Feng, Dong Li, and Wulong Liu. Towards effective context for meta-reinforcement learning: an approach based on contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7457–7465, 2021.

[20] Bradly C Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118*, 2018.

[21] Evan Z Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In *International conference on machine learning*, pages 6925–6935. PMLR, 2021.

[22] 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/google/jax`.

[23] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[24] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

# Appendix

## Table of Contents

# A Background

We define **environments** formally as Partially Observable Markov Decision Processes (POMDPs, [7]). POMDPs are specified by a tuple $(S, A, p, p_0, R, \Omega, O, \gamma)$, where $S$ is the state space, $A$ the action space, $p : S \times A \to S$ a probabilistic transition function mapping from the current state and action to the next state, $p_0$ a distribution over starting states, $R : S \times A \to \mathbb{R}$ a stochastic reward function, $\Omega$ the space of environment observations, $O : S \to \Omega$ a stochastic function mapping from states to observations, and $\gamma$ the discount factor. The environment starts (at $t = 0$) in a start state $s_0$ according to $p_0$, $s_0 \sim p_0$. Each subsequent time-step, the agent receives the current state's observations $o_i = O(s_i)$, takes an action $a_i$, and the transition function $p$ updates the environment state $s_{i+1} = p(s_i, a_i)$. An episode $\tau$ of length $h$ is then a sequence of time-steps starting from $t = 0$ to $t = h$. The sum of an episode's $\gamma$-discounted rewards is called its return $G(\tau) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)$. **Standard-RL** generally aims to learn a policy $\pi : O \to A$ that maximizes the expected episode return $\mathbb{E}[G(\tau)]$.

**Meta-RL** generalizes several aspects of Standard-RL. First, instead of there being a single environment, there is a distribution $\mathcal{D}$ of environments $E_i = (S_i, A_i, p_i, p_{0,i}, R_i, \Omega_i, O_i, \gamma_i) \sim \mathcal{D}$. Second, the agent performs a *sequence* of $n$ episodes $\tau_0, \ldots, \tau_n$. Third, the agent is provided with some form of access to memory (e.g., via a sequence model), allowing it to adapt based on the history of states, actions and rewards, including those from prior episodes in the sequence. Finally, meta-RL performs *meta-rollout*, defined as follows: a new environment $E_i$ is sampled, then the agent performs in a sequence of episodes within that environment. Each episode, first the start state is resampled from the environment start-state distribution $p_i$, and then the agent acts until the end of the episode (e.g., for a set number of time-steps or until a terminal state is reached). The agent's memory persists across episodes in the sequence. This capacity enables training the agent (via weight updates) to learn (via in-context adaption) from earlier episodes, enabling higher rewards in later episodes.

Meta-RL methods can be split into two approaches that each solve a different problem, with specific algorithms designed and used for each approach [1][2]. **Cumulative-Episode-Reward Meta-RL** trains a single policy to maximize cumulative reward $\sum_{i=1}^{n} G(\tau_i)$. **Final-Episode-Reward Meta-RL** aims to optimize solely for final episode reward $G(\tau_n)$. In this paper we compare and analyze First-Explore in a cumulative-episode-reward setting. However, for the sake of completeness, we discuss final-episode-reward meta-RL and its connection to First-Explore in Appendix L.

# B Related Work

$RL^2$ [4, 5] is one of the first meta-RL approaches, and uses a standard-RL algorithm with an RNN to learn to in-context adapt. It has the advantage of simplicity, and attempts to converge to optimally balance exploration and exploitation across successive episodes.

Subsequently, various meta-RL works have been produced. VariBAD [2] has been shown to improve on $RL^2$ performance on certain domains. VariBAD achieves this improvement via splitting training into producing a posterior belief of the current task (task inference), and a task-posterior conditioned policy. HyperX [3], which augments training rewards to incentivize exploration, has also been shown to improve performance on some environments with sparse rewards. However, as Section 3 shows, these methods can become trapped on local optima of not exploring, when good exploration requires forgoing episode reward. On such challenging environments, First-Explore significantly outperforms $RL^2$, VariBAD and HyperX.

AdA [9] demonstrates that training meta-RL on a sufficiently advanced curricula (a tailored series of learning challenges) can produce agents capable of human-like adaption on complex unseen tasks. However, AdA may struggle to forgo reward: the AdA paper describes how their agent always behaved to maximize current episode return, and failed to learn to prioritize exploration (to enable higher future episode returns) at the cost of current episode return (even when provided the opportunity to theoretically do so). This lack of prioritized exploration suggests AdA's performance potentially relied on training and testing on environments that do not require such sacrificial exploration. Unfortunately, investigating AdA's dynamics is far outside of the scope of this paper, as it would require a huge compute expenditure that dramatically outstrips the capabilities of our academic lab.

[2] [1] use different, and in our view less clear terminology, calling Cumulative-Reward Meta-RL "Few-Shot Meta-RL", and Final-Episode-Reward Meta-RL "Zero-Shot Meta-RL".

## C Architecture and Training

For the experiments in this paper, First-Explore was implemented with a GPT-2-style causal transformer architecture [6], due to the excellent temporal sequence modelling abilities of transformers. For simplicity, the parameters are shared between the explore and exploit policies, differing only in the final linear-layer head.

**Training Method** (see Appendix C.1 for pseudo code): Recall that the exploit policy is intended to produce episodes that have high reward, and the explore policy is intended to produce episodes that maximally inform the exploit policy (in the sense of leading to higher rewards). To achieve these aims, the exploit policy $\pi_{\text{exploit}}$ is trained to produce episodes that meet or exceed the highest achieved episode return in all preceding episodes in the meta-rollout sequence. These episodes are termed 'maximal.' Because this maximal value is not defined for the *first* exploit episode (being the maximum of an empty list), a baseline reward $b$ seeds the list, which is set as a domain specific hyperparameter (but could easily be set automatically via heuristics). Simultaneously, the explore policy $\pi_{\text{explore}}$ is trained to produce episodes that are followed by the exploit policy achieving higher episode returns than those seen so far. These episodes are termed 'informative.'

First, both rollout policies $(\pi_{\text{explore}}, \pi_{\text{exploit}})$ are initialised with random weights. They are then copied to produce a second set of predictor policies $(\phi_{\text{explore}}, \phi_{\text{exploit}})$ that train to model the rollout polices conditioned on them being maximal and informative, see Equation (1). The current exploit and explore policies $(\pi_{\text{explore}}, \pi_{\text{exploit}})$ then produce batched meta-rollouts. In each batch, the exploit episodes $\tau_{\text{exploit}} \sim \pi_{\text{exploit}}$ that are *maximal* and the explore episodes $\tau_{\text{explore}} \sim \pi_{\text{explore}}$ that are *informative* are recorded. For these criteria-satisfying episodes, a cross entropy loss is calculated between the predicted actions of the predictor policies $(\phi_{\text{explore}}, \phi_{\text{exploit}})$ and the actions taken in the associated *maximal* or *informative* episodes. The predictor policy weights are then updated, using gradient descent on the loss. In this way the predictor policies learn to emulate the conditioned rollout policies (conditioned on producing *maximal* exploit episodes and *informative* explore episodes).

$$\begin{aligned} \phi_{\text{explore}} &\approx \pi_{\text{explore}} \mid \text{'informative episodes'} \\ \phi_{\text{exploit},i} &\approx \pi_{\text{exploit}} \mid \text{'maximal episodes'} \end{aligned} \tag{1}$$

After every $T$ meta-rollout batches, the rollout policies $\pi_{\text{explore}}, \pi_{\text{exploit}}$ are set equal to the current predictor policies $\phi_{\text{explore}}, \phi_{\text{exploit}}$. This hyperparamter $T$ determines the balance between maintaining healthy behavioral diversity and learning iteration speed.

During training, the actions are sampled according to the predicted likelihood of the policy selecting that action. However, during inference, the actions are selected greedily (taking the action that is predicted most likely to lead to the desired episode property).

While preliminary experiments found this meta-RL training method performed better than others tried (and lead to improved training stability), we believe the First-Explore meta-RL framework will work with other training approaches too (e.g., common RL algorthims like PPO [10] and Q-learning [11]).

### C.1 Training Pseudocode

```
def rollout(env, π, ψ, c_π, c_ψ):
    ### """perform a single episode
    # inputs: the environment (env),
    # the agent policy π, the prediction policy ψ,
    # and the current policies' contexts c_π, c_ψ
    ### returns the episode return, temp_loss, and updated contexts"""
    temp_loss, r = 0, 0
    # n.b. temp_loss is only used if the episode meets a condition
    # see (*) in the conditional_action_loss function
    s = env.reset()  # state s
    for i in range(env.episode_length):
        # calculate action probabilities p_π, p_ψ for both policies
        # and update context
        p_π, c_π = π.action_probabilities(s, c_π)
        p_ψ, c_ψ = ψ.action_probabilities(s, c_ψ)
        a = sample_action(p_π)
        temp_loss += cross_entropy(a, p_π * p_ψ)  # hardamard product
        # * p_π ensures action diversity by weighting against likely actions
        s = env.step(s, a)
        r += s.reward
    return r, temp_loss, c_π, c_ψ

def conditional_action_loss(φ, θ, D, b):
    ### """calculates First-Explore loss for both exploring and exploiting
    # on domain D using the agent and predictor parameters φ, θ
    ### and baseline reward b"""
    env = sample_env(D)
    π_explore, π_exploit = load_policies(θ)
    ψ_explore, ψ_exploit = load_policies(φ)
    c_π, c_ψ = set(), set()  # the agent and predictor contexts
    loss, best_r = 0, b
    for i in range(D.episode_num):
        r_explore, l_explore, c_π, c_ψ = rollout(env, π_explore, ψ_explore, c_π, c_ψ)
        r_exploit, l_exploit, _, _ = rollout(env, π_exploit, ψ_exploit, c_π, c_ψ)
        # ^exploit context not kept
        # (*) accumulate loss if:
        if r_exploit >= best_r:  # exploit episode is 'informative'
            loss += l_exploit
        if r_exploit > best_r:  # explore episode is 'maximal'
            loss += l_explore
            best_r = r_exploit
    return loss
```

Algorithm 1: Example First-Explore Cross-Entropy Loss.

```
def train(epoch_num, T, D, b):
    """example First-Explore training (ignoring batchsize)
    runs the meta-rollouts, accumulating a loss
    this loss is then auto-differentiated"""
    T_counter = 0
    φ = θ = init_params()
    for i in range(epoch_num):
        # θ is the agent behavior parameters
        # φ is the prediction parameter (for double-DQN-style updates)
        Δφ = ∂/∂φ(conditional_action_loss(φ, θ, D, b))
        φ -= step_size * Δφ
        # Update θ every T epochs
        T_counter += 1
        if T_counter == T:
            θ = φ
            T_counter = 0
    return θ
```

Algorithm 2: Example of Training First-Explore using the Cross-Entropy loss and Auto-Differentiation.

# D Detailed Domain Results:

## D.1 Meta-RL-Deceiving Bandits

**Problem Description:** taking inspiration from Section 1, consider a distribution of bandit environments. Each bandit has ten arms, and in the environment the agent acts by selecting an arm to pull $a \in [1-10]$. The first arm always yields the reward $\mu_1$, while the other arms yield their environment specific arm mean $v_{a\in[2-10]} \sim \mathcal{N}(0,1)$ plus (unlike in Section 1) a normally distributed noise term $\mathcal{N}(0, \frac{1}{2})$, added to make the environments more challenging. The arm mean is fixed once the environment is sampled, but the noise term is resampled each arm pull. The objective is to maximize the expected reward obtained on a newly sampled bandit over 100 pulls. **Treatments:** $RL^2$, First-Explore and two specialized bandit algorithms were all evaluated on the bandit distribution for two values of the guaranteed arm reward $\mu_1 = 0$ and $\mu_1 = 0.5$[3] The bandit algorithms: UCB-1 estimates an upper confidence bound and selects the arm that maximizes it, see Appendix M for details. Thompson Sampling (TS) [8] samples arm means from the posterior distribution, and chooses the arm with the best sampled mean.

Deceptive Bandits in the main paper Fig. 2 refers to the $\mu_1 = 0.5$ setting.

**Results:** First-Explore outperforms all other treatments for both values of $\mu_1$ (green lines above all others in Fig. 3:A1 & A2), with First-Explore having the highest mean and median cumulative-reward (Appendix G). The difference is statistically significant, with $p < 2 \times 10^{-5}$ for outperforming TS in the $\mu_1 = 0.5$ setting, and $p < 6 \times 10^{-8}$ for all the other comparisons, as calculated by a two-tailed Mann-Whitney $U$ Test (TMWU). Furthermore, unlike First-Explore, $RL^2$ is affected by deception. When the guaranteed arm reward of $\mu_1$ is increased (raising the average arm-reward and making the environment *more* rewarding), $RL^2$ achieves significantly less reward (fuchsia lines lower in Fig. 3:A1 & B1 than in A2 & B2). The distributional difference is statistically significant, $p < 6 \times 10^{-8}$ (TMWU). First-Explore overcomes the deception, and achieves high cumulative reward regardless of $\mu_1$.

[3] VariBAD and HyperX were also evaluated, however despite hyperparameter tuning, good performance on either distribution could not be achieved. This performance is possibly due to these algorithms not being suited to the large continuous task space caused by each bandit being specified by the 9 arm means $v_{a\in[2-10]} \sim \mathcal{N}(0,1), \mathbf{v} \in \mathbb{R}^{\mathbf{9}}$. In contrast, [2] evaluates VariBAD on small discrete task spaces, e.g. a goal being in one of 25 locations.

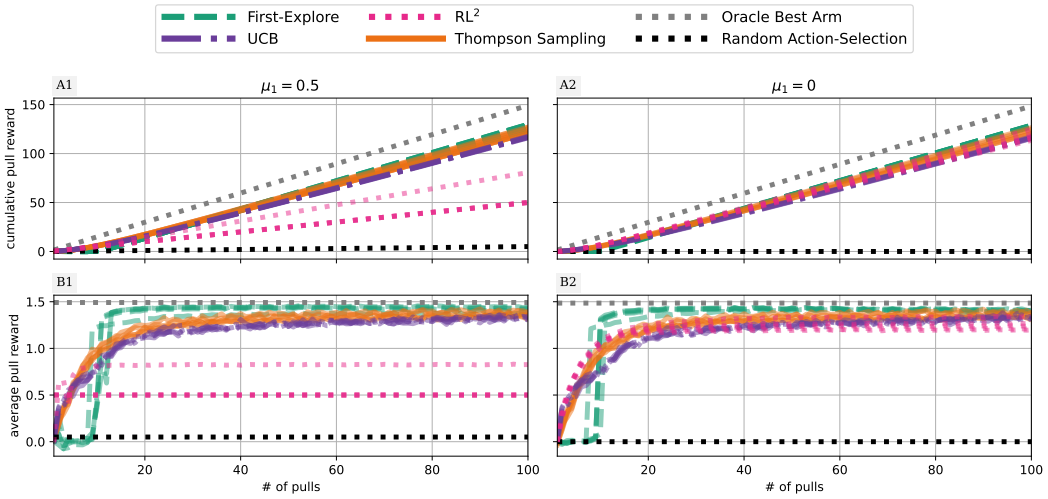



Figure 3: Mean performance (averaged across sampled bandits) of algorithms for $\mu_1 = 0$ and $\mu_1 = 0.5$. 5 independent runs of each algorithm are plotted semitransparent for each $\mu_1$. **A1 & A2 - Cumulative Environment Rewards:** i) First-Explore (green) achieves the highest median and mean cumulative rewards for both values of $\mu_1$. Outperforming specialized bandit algorithms – UCB (purple) and TS (brown) – is significant as First-Explore is not specialized for this setting, and can be readily applied (via training) to other non-bandit settings. ii) $RL^2$ (light pink, and fuchsia when run plots overlap) is hugely affected by the value of $\mu_1$ (unlike the other treatments). On the $\mu_1 = 0.5$ distribution, 4 of 5 training runs achieve an abysmal cumulative reward of 50 (which corresponds to sampling the first arm every pull). **B1 & B2 - Mean Arm Pull Rewards:** i) First-Explore differs from the other strategies in that it switches between exploration and exploitation (depending on the automatically chosen $k$). This behavior is seen in the sharp shift in average First-Explore pull rewards (green) around pull number 10. The individual First-Explore training runs have selected different values of $k$, as seen by the reward shifts occurring at different pull numbers. ii) After the exploration phase, First-Explore achieves the highest average arm-pull rewards, nearing the oracle performance of always pulling the correct arm (dotted gray).

### D.2 Dark Treasure-Rooms

**Problem Description:** Dark Treasure-Rooms (inspired by the Darkroom in [12]) are $9 \times 9$ grids full of treasures and traps. The agent starts in the middle of the grid, navigates (up, left, down, or right) to find treasure, and cannot see its surroundings. Only its current $(x, y)$ coordinates are observed. Each environment has 8 objects (treasures or traps), and when the agent encounters a treasure or trap it consumes/activates it, and receives an associated reward (positive or negative). The treasure and trap values $v_i$ are uniformly distributed in the range $\rho$ to 2, $v_i \sim U[\rho, 2]$, with $\rho$ being the maximum trap penalty. The locations of the objects are sampled uniformly, with overlapping objects having their rewards/penalties summed. Each episode has 9 steps, and the objective is to maximize the expected cumulative episode returns on a newly sampled Dark Treasure-Room over multiple episodes. **Treatments:** $RL^2$, VariBAD, HyperX and First-Explore were all trained to maximize cumulative reward on the Dark Treasure-Room distribution for two values of the maximum trap penalty: $\rho = -4$ and $\rho = 0$. For $\rho = -4$, these policies were trained to maximize reward over a greater number of episodes (10 rather than 6) so as to highlight the compounding value of good early exploration. Random action selection is also included as a baseline for both values of $\rho$.

Deceptive Treasure Rooms in Fig. 2 refers to the $\rho = -4$ Dark Treasure-Room setting.

**Results:** First-Explore outperforms all other treatments for both values of $\rho$ (green lines above all other lines in Fig. 4:A1 & A2), with First-Explore having the highest mean and median cumulative-reward (Appendix G). The difference is statistically significant, with $p < 6 \times 10^{-8}$ for all pairwise comparisons (TMWU). Furthermore, for $\rho = 4$, $RL^2$ and VariBAD converged to the local optima of avoiding exploration, staying still the vast majority of the time (fuchsia and purple lines close to zero in Fig. 4:B1). The issue is that a highly negative trap penalty means that achieving high cumulative

reward requires forgoing reward in early episodes (as to find treasure, the agent must visit unseen locations, which have negative expected value when $\rho < -2$). This requirement of forgoing early episode rewards is deceptive, and can cause convergence to the local optima of not exploring (thus risking no traps). First-Explore overcomes this deception, and performs well regardless of $\rho$.

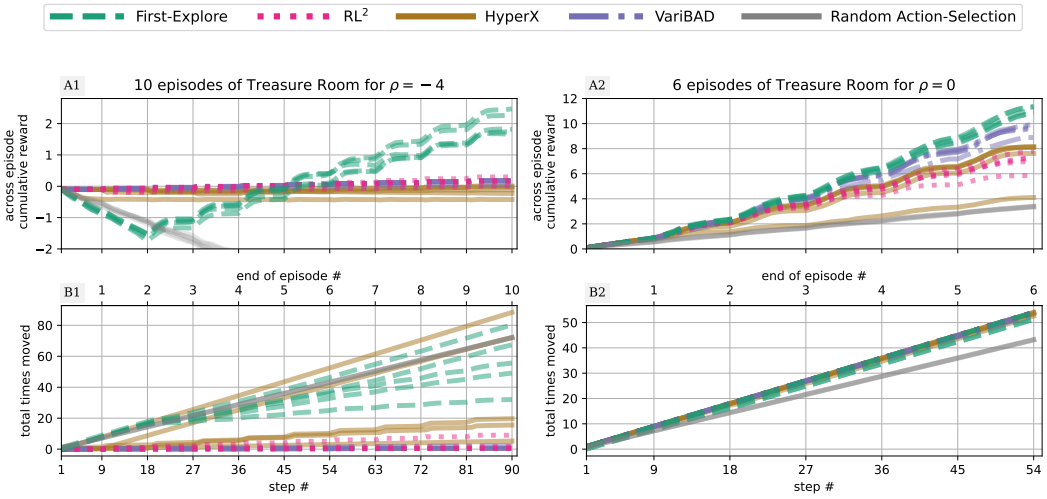
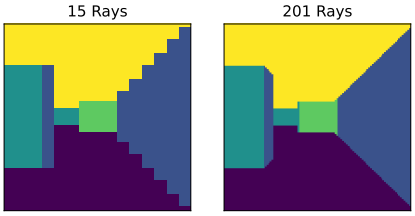



Figure 4: Mean performance (averaged across sampled Dark Treasure-Rooms) for two values of the trap penalty $\rho$. For each $\rho$, 5 runs of each treatment are plotted (with the meta-RL algorithms retrained for each run). Each run was independently evaluated on a large batch of sampled Dark Treasure-Rooms (independent from those used in training). **A1 & A2 - Cumulative Environment Rewards:** i) First-Explore (green) achieves higher rewards for both values of $\rho$, with every First-Explore run outperforming all other runs. ii) By first exploring for two episodes using $\pi_{\text{explore}}$, and then exploiting using $\pi_{\text{exploit}}$, First-Explore's inference policy $\pi_{\text{inference}}$ achieves significant cumulative reward by the end of the $\rho = -4$ setting (despite having to brave a period of negative reward initially). This behavior enables First-Explore to achieve vastly more cumulative reward than the other methods (A1), with the other meta-RL methods – RL$^2$ (light pink & fuchsia when run plots overlap), VariBAD (purple), and HyperX (brown) – achieving close to zero reward, due to the meta-RL deception created by the penalty. Random Action-Selection obtains significant negative reward due to repeatedly encountering traps, highlighting how hostile $\rho = -4$ Dark Treasure-Rooms are. A2 shows that when the environment is not hostile to exploration, no deception exists and all methods can achieve positive cumulative reward (though First-Explore still outperforms them) **B1 & B2 - Cumulative Time Stood Still:** The distribution is only deceptive when $\rho = -4$. This dynamic is reflected in how often the policies move (risking encountering a trap). When $\rho = 0$ all policies move regularly (lines high in B2, top right), but when $\rho = -4$, they mostly stay still. Only First-Explore (green) and two HyperX runs (brown) move consistently in early episodes. While these two HyperX runs exhibit potential exploration, HyperX fails to combine it with effective exploitation (A1).

### D.3 Ray Maze

**Problem Description**: Ray Maze challenges the algorithms with a more complex domain. The agent must navigate a randomly generated maze of impassible walls (the maze layout is different for each sampled Ray Maze) to find one of three goal locations. Each goal location is either a trap or a reward, with a 0.3 probability of the goal being a treasure, and a 0.7 probability of it being a trap. Treasures give reward 1, while traps give penalty $-1$. In this domain, the agent can only receive one goal reward per episode, as triggering the first prevents others activating. To perceive the maze, the agent receives 15 lidar observations (see Fig. 5). Each lidar observation reports the distance to the nearest wall along an angle (relative to the agent's orientation). It further tells the agent whether the lidar ray hit a goal location, as well as the wall orientation (east-west or north-south). However, the lidar measurements do not show whether a goal is a trap or a treasure. The agent has 3 actions, turn left, go forward, and turn right, with rotation turning the agent's field of view.

Figure 5: **Left:** Raw agent observations from a sampled ray maze converted to an image. The agent receives the wall distances and the wall types. Portraying this numerical data as an image, goal locations are **green**, and the two wall orientations are distinguished (**east-west teal**, and **north-south navy**). To aid the eye, the floor has been coloured in **dark purple**, and the **sky yellow**. Although the goal is visible, it could be a treasure or trap. **Right:** the image produced direct ray casting (large number of processed lidar measurements) rather than the 15 the agent receives.




Ray Maze is a challenging domain for several reasons. It has a high-dimensional observation space (15 separate lidar measurements), complex action dynamics (with actions not commuting, e.g. turn left then move forward is different from move forward then turn left) and a randomly generated maze that interacts with both movement and observation. Furthermore, similarly to earlier environments, the agent must learn from experience, and risk the traps in early episodes, so as to exploit and consistently find treasure in later ones. Because of how frequent traps are, the agent can only obtain positive cumulative reward by a) consistently searching for treasures in early episodes (at the cost of expected reward, as the average value of a goal is negative) and b) repeatedly exploiting in later episodes (navigate to identified treasures, while avoiding potential traps).

**Results:** On this challenging domain, First-Explore outperforms all other treatments (**green lines above all other lines**), with First-Explore having the highest mean and median cumulative-reward. The difference is statistically significant, with $p < 6 \times 10^{-8}$ for all pairwise comparisons (TMWU).

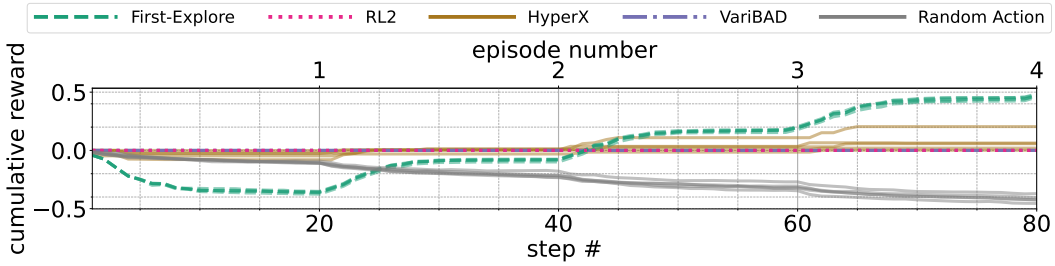



Figure 6: Mean performance averaged across a large sample of Ray Mazes for five runs of each treatment. **RL$^2$ (fuchsia)** and **VariBAD (purple)** learn to never move forward (and thus never risk encountering traps) in all five independent training runs, hence achieving a constant zero cumulative reward. Most **HyperX (brown)** runs move little (achieving near zero reward), with one HyperX run achieving modest cumulative reward). Random action selection **(gray)** achieves highly negative cumulative reward, highlighting the hostility of the domain. In contrast, all **First-Explore runs (green)** achieves significant cumulative reward (outperforming all controls). The performance is because each separate training run has learnt to sacrifice the rewards of the first episode to enable significant subsequent cumulative reward.

## E Limitations and Future Work

As presented, First-Explore does not explore to better enable future exploration. The explore policy $\pi_{\text{explore},\theta}$ only trains to increase the expected reward of the *subsequent* exploit policy episode. Unfortunately, even iterated optimal myopic exploration does not necessarily produce an optimal sequence of explorations (Appendix J). A potential solution is to reward exploration episodes with weighted sums of the rewards of all subsequent exploitation (analogous to summing discounted future reward in standard-RL).

Another limitation is that First-Explore could be unsafe in certain environments. The risk is that First-Explore's explore policy readily receives highly negative rewards, and avoiding negative rewards can be important for safety. For example, if a chef robot is learning a new recipe in a physical home then it is vital the robot does not make mistakes that endanger humans or damage the kitchen, unlike mistakes such as creating ugly or flavorless food. While this is only a subset of environments, with many environments being safe (e.g., simulated environments), addressing this issue is an interesting direction of future work. One potential solution is to separately penalize safety risks while training both the explore and exploit policy (disabling the ability of the explore policy to forgo safety associated incentives). This proposed version of First-Explore could actually result in in-context adaption that is safer than standard-RL training, as the meta-RL policies would have learnt a strong prior of avoiding potentially endangering actions. However, it could also be the case that a strong penalty might prohibit effective training.

A final problem is the challenge of long sequence modelling, with certain environments requiring a very large context and high compute (e.g., can one have a large enough context, and enough compute to allow First-Explore to generalize and act as a replacement for standard-RL?). AdA [9] suggests such a project might be possible. Progress on efficient long-context sequence modelling [13, 14], research on RL transformer applications [12, 15], and Moore's Law all make this application more feasible.

Additionally, given that First-Explore learns a dedicated explore policy, and a dedicated exploit policy, and both have been shown to work well (e.g., Fig. 3), the method may be applicable to final-episode-reward meta-RL settings. Another interesting direction would be running a sequence of explorations *until* sufficient information is obtained (just as standard-RL can be trained *until* convergence). However, despite these applications being highly exciting directions of future work, a proper investigation of either would require its own paper, including many different specialized controls and environments pertinent to the setting.

## F Further Discussion

Given that First-Explore uses RL algorithms to train the meta-RL policy, how might it solve hard-exploration problems that standard-RL cannot, e.g. design a rocket for the first time? We believe that given a suitably advanced curriculum, and sufficient compute, First-Explore could learn powerful exploration heuristics (i.e., develop intrinsic motivations, e.g., an analogue of curiosity) and that these heuristics would enable such hard problems to be tackled (with great sample efficiency). E.g., initially the First-Explore agent can only randomly explore and must learn to exploit based on random exploration. Once it has learnt rudimentary exploitation, the agent can learn rudimentary exploration. Then it can learn better exploitation, then better exploration, and so on, each time relying on there being 'goldilocks zone' tasks [16] that are not too hard and not too easy.

Further, while curricula can aid all of meta-RL, e.g., $RL^2$ and AdA, First-Explore can have a significant training advantage on certain problems (e.g., in the ten-episode Dark Treasure-Room, First-Explore achieves positive cumulative reward while the standard cumulative-reward meta-RL methods catastrophically fail). This advantage could potentially allow far greater compute efficiency, and allow training on otherwise infeasible curricula.

## G Tabulated Results:

Table 1: Meta-RL Deceiving Bandit Results

| | $\mu_1 = 0$ mean | $\mu_1 = 0$ median | $\mu_1 = 0.5$ mean | $\mu_1 = 0.5$ median |
|---|---|---|---|---|
| First-Explore | **128.36** | **128.62** | **127.74** | **128.45** |
| $RL^2$ | 117.90 | 116.99 | 56.12 | 50 |
| UCB-1 | 116.05 | 115.84 | 116.83 | 116.71 |
| TS | 122.73 | 121.91 | 123.28 | 122.53 |

Table 2: Dark Treasure-Room Results

| | $\rho = -4$ mean | $\rho = -4$ median | $\rho = 0$ mean | $\rho = 0$ median |
|---|---|---|---|---|
| First-Explore | **1.99** | **1.81** | **11.07** | **11.04** |
| $RL^2$ | 0.16 | 0.16 | 7.04 | 7.26 |
| VariBAD | 0.15 | 0.18 | 9.58 | 9.68 |
| HyperX | -0.16 | -0.11 | 7.22 | 8.10 |

# H Control Convergence

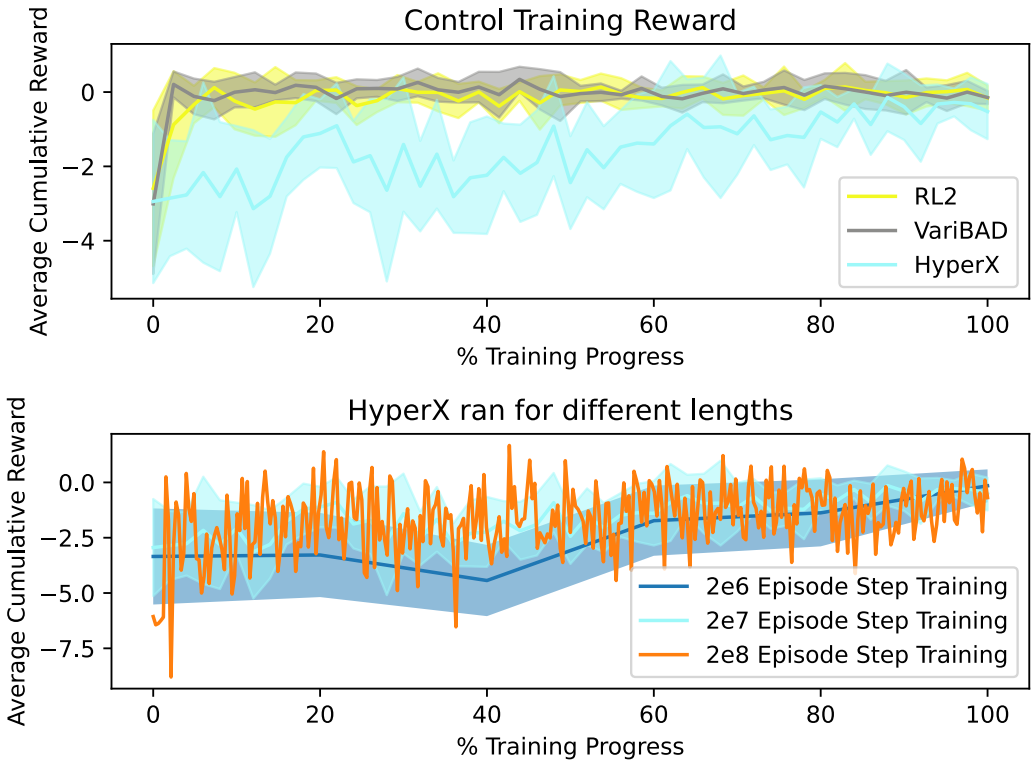



Figure 7: Control Training on the Dark Treasure-Room Environment. **Top:** $RL^2$, VariBAD, and HyperX average cumulative reward plotted against training time. $RL^2$ (yellow) and VariBAD (gray) converge to zero reward almost immediately. This transition corresponds to the policies learning to stay still (as the coverage plots demonstrate in Fig. 4). HyperX (teal) reward increases (toward zero) throughout meta-training. However, this increase in reward comes not from HyperX learning an increasingly sophisticated policy, but instead is the result of the HyperX algorithm's meta-training exploration bonus being linearly reduced from the start to the end of meta-training. Thus, once that bonus is near zero, HyperX also learns to stay still. **Bottom:** HyperX with different training lengths (specified by number of episode steps). When HyperX is run for ten times as long (orange) or ten times less long (blue) than the default training time (light blue) the same behaviour is observed (of slow convergence to (slightly below) zero reward). This behaviour demonstrates the improvement in reward comes from the HyperX algorithm reducing the exploration incentive during the meta-training. It also implies that changing the length of training runs (including running for much longer) would not change the final performance results.

# I Compute Usage

Each training run commanded a single GPU, specifically a Nvidia T4, and up to 8 cpu cores. Table 3 gives the approximate walltime of each run.

Table 3: Compute Usage Per Training Run

| Run | Runtime |
|---|---|
| Meta-RL Deceptive Bandits First-Explore | 18.5 hours |
| Meta-RL Deceptive Bandits $RL^2$ | 40 hours (extended, see below) |
| Dark Treasure-Room First-Explore | 50 hours |
| Dark Treasure-Room HyperX & VariBAD & $RL^2$ | 10 hours (converges early, see Fig. 7 of Appendix H) |
| Dark Treasure-Room HyperX & VariBAD & $RL^2$ | 10 hours (converges early, see Fig. 7) |

Notably, in the Dark Treasure-Room for $\rho = -4$, VariBAD and $RL^2$ rapidly converges to a policy of staying still, and while HyperX seems to slowly improve rewards, the reward increase is an artifact caused by the HyperX having an exploration incentive that is gradually attenuated to zero. This attenuation creates a slow convergence from negative reward (due to moving into traps and not exploiting) to a higher near zero reward (obtained by mostly staying still). Because the attenuation is designed to occur throughout the entire run, scaling the run length merely scales how long HyperX takes to converge to close to zero reward (see Fig. 7 of Appendix H).

Due to a desire to not waste compute on converged policies, once this behaviour was verified, the control runs on this setting were limited to 10 hours. In contrast, First-Explore was run for longer, as it continued to improve with additional training. This is a fair comparison, as due to the controls having converged, increasing the control run training time would not yield better policies, as Fig. 7 demonstrates. To this end, as $RL^2$ showed gradual improvement at 10 hours, it was also trained for longer.

Total compute used for the experiments would then be around 650 hours (5 runs for each treatment). However, there was also hyperparameter search, e.g., for the $RL^2$ bandit parameters. As such, total compute may be over 1100 GPU hours. Furthermore, there were many preliminary experiments to iterate on the First-Explore architecture as well as to research and identify cumulative-meta-RL deception.

## J Myopic Exploration

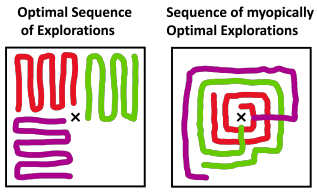



Figure 8: Well-planned sequential exploration can sometimes significantly outperform a sequence of optimal myopic explorations. For example, consider exploring a plain over four days, where each day one must explore by walking from the plain's center. **Sequence of Optimal Myopic Explorations** one 'optimal' way of exploring is to perform a spiral from the center (e.g., the red spiral on the right). This strategy achieves the optimal amount of exploration on day 1 as one never retraces one's steps. However, if one does a spiral on day 1 then on day 2, one must retread old ground - wasting time otherwise spent exploring new locations. Each day bee-lining to unseen areas and then spiralling from there is also optimal for that day, however it increases the amount of retreading tomorrow. **Optimal Sequence of Explorations:** another optimal way of exploring on day 1 is to explore a quadrant, visualized in red on the left. Again, as one does not backtrack, this strategy is optimal on day 1. However, unlike the spiral strategy, this strategy is also part of an optimal sequence of four explorations, as one can explore a new quadrant each of the four days, without ever retreading the same ground.

## K K-Selection Phase

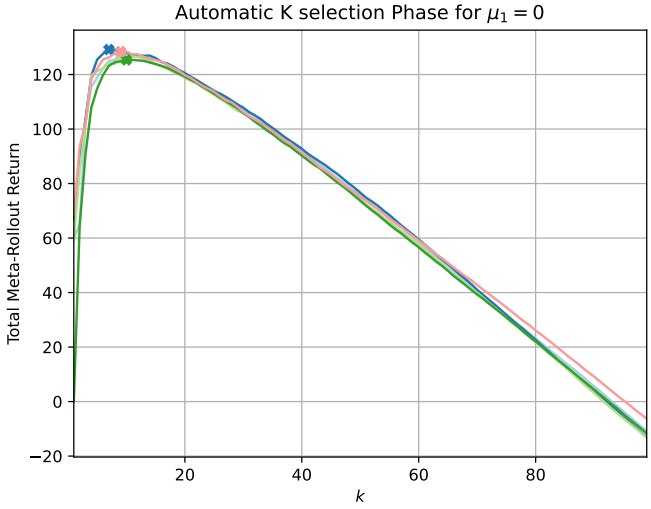



Figure 9: Demonstration of First-Explore's $k$-selection phase, for the bandit distribution, with $\mu_1 = 0$. Five separate First-Explore runs are plotted. The training runs select different values of $k$ (due to the relative strengths of each runs explore and exploit policy), with the associated selected $k$ corresponding to the peak of each curve (marked by a cross).

## L Final-Episode-Reward Meta-RL

Methods such as MetaCURE [17], EPI [18] and CCM [19] learn an exploration policy that aims to extract maximum environment information (independent of whether such information informs good exploitation). These approaches discard grounding exploration in (maximizing) future reward. Not grounding exploration in future exploitation reward means that the policy may learn (via weight updates) to spend meta-rollouts acquiring irrelevant information (e.g., the exact penalty of bad actions). This distraction potentially prevents optimal exploration from ever being learnt.

E-RL$^2$ [20] modifies RL$^2$ to ignore the first-$k$ episode rewards. This modification enables pure exploration (that is not dissuaded by negative rewards). However, E-RL$^2$ introduces an across-episode value assignment problem: identifying which exploration episodes enabled good subsequent exploitation. This problem potentially limits training sample efficiency. Further, the exploratory episodes number $k$ is set as a hyperparameter and constant across all tasks (both at training and at inference), preventing efficient combination with a curriculum that contains different difficulty tasks (as hard tasks may need significantly more exploration episodes than easy ones). Finally, hard coding $k$ limits the flexibility and usefulness of E-RL$^2$ because one cannot explore until a satisfactory policy quality is reached, preventing meta-RL in-context adaptation from off-the-shelf replacing standard RL.

DREAM [21] also separately optimizes exploration and exploitation policies (and grounds exploration in exploitation), but has four complex, manually designed, interacting components and a reliance on knowing unique problem IDs during meta-training. This complexity enables increased sample-efficiency by avoiding the chicken and egg problem of simultaneously learning explore and exploit policies. Unlike E-RL$^2$, because a part of DREAM's machinery must learn to produce the right information per problem based on the (unique, random) problem ID only, it is unable to generalize or handle never-seen-before challenges during meta-training, raising questions about its scalability and generality. For example, DREAM may potentially be difficult to apply to problems where each training environment is unique (e.g., for environments with continuous variables, or samples from otherwise vast search spaces). It may also struggle when each environment is a hard-exploration challenge, as it may be difficult for the model to explore enough to learn which information is required to solve the problem. We believe curricula are necessary to solve such environments. However, because DREAM cannot generalize during meta-training (as described above), it cannot take advantage of a curriculum to build an exploration skill set to tackle harder and harder exploration challenges.

Applying First-Explore to a final-episode-reward meta-RL setting is a promising direction of future work, as First-Explore i) learns grounded exploration ii) can explore until sufficient information is obtained ( rather than having a fixed number of explorations), and iii) does not rely on privileged information (problem IDS), allowing generalization during meta-training.

## M Evaluation Details

Evaluation (sampling the multiple evaluation environments and performing iterated rollouts) was with a single GPU. For the **Bandit Results**, each of the First-Explore evaluations sampled $128 \times 100$ bandit environments, indepedent from those trained on. For the $5$ RL$^2$ bandit evaluations batch size was reduced to $2,000$ (due to taking longer to evaluate). Since there is no meta-RL training variance for UCB1 and TS, five independent evaluations were done, each with an independently sampled $10,000$ bandits.

**UCB**: UCB was implemented according to the description in [4], with $c = 1$. Namely, each pull, UCB picks the arm that maximizes $\text{ucb}_i(t) = \hat{\mu}_i(t-1) + \sqrt{\frac{2 \log t}{T_i(t-1)}}$, where $\hat{\mu}_i(t-1)$ is the estimated mean reward of the $i$th arm, $T_i(t-1)$ is the number of times the $i$th arm has been pulled

For the **Dark Treasure-Room** all policies were evaluated on a batch of $1,000$ environments sampled independently from those trained on.

# N Training Details

## N.1 Controls:

The official VariBAD [2] (VariBAD and $RL^2$) and HyperX [3] (HyperX) codebase ran the meta-RL controls. **Dark Treasure-Room** trained with the default hyperparameters of the coded bases gridworld environments. These were found to perform well, with variations tried not yielding improvement. To provide a strong control on the **Meta-RL Deceptive Bandits** problem, the controls were advantaged by having individual hyperparameter gridsearches for each $\mu_1$ value (unlike First-Explore). See the SI attached configuration file for the exact hyperparameters. Both of these codebases are licensed under a MIT license.

## N.2 First-Explore:

The architecture for both domains is a GPT-2 transformer architecture [6] specifically the Jax framework [22] implementation provided by Hugging Face [23], with the code being modified so that token embeddings could be passed rather than token IDs. The different hyperparameters for the two domains are given in Table 4. The code being provided with a Modified MIT License (allowing free use with attribution).

For both domains each token embedding is the sum of a linear embedding of an action, a linear embedding of the observations that followed that action, a linear embedding of the reward that followed that action, a positional encoding of the current timestep, and a positional encoding of the episode number. See the provided code for details. For the dark treasure-room environments a reset token was added between episodes that contained the initial observations of the environment, and a unique action embedding corresponding to a non-action. The bandit domain had no such reset token.

Table 4: Model Hyperparameters

| Hyperparameter | Bandit | Dark |
|---|---|---|
| Hidden Size | 128 | 128 |
| Number of Heads | 4 | 4 |
| Number of Layers | 3 | 4 |

For training we use AdamW [24] with a piece-wise linear warm up schedule that interpolates linearly from an initial rate of $0$ to the full learning rate in the first $10\%$ training steps, and then interpolates linearly back to zero in the remaining $90\%$ of training steps. Table 5 gives the optimization hyperparameters.

Table 5: Optimization Hyperparameters

| Hyperparameter | Value |
|---|---|
| Batch Size | 128 |
| Optimizer | Adam |
| Weight Decay | 1e-4 |
| Learning Rate | 3e-4 |

Hyperparameters were chosen based on a relatively modest amount of preliminary experimentation. Finally, for efficiency, all episode rollouts and training was done on GPU using the Jax framework [22].

For evaluation, we then sample by taking the argmax over actions, and do not add the $\epsilon$-noise.

Table 6: Training Rollout Hyperparameters

| Hyperparameter | Bandit | Darkroom |
|---|---|---|
| Exploit Sampling Temperature | 1 | 1 |
| Explore Sampling Temperature | 1 | 1 |
| Policy Update Frequency | every training step | every $10,000$ training steps |
| $\epsilon$ chance of random action selection | 0.05 | 0 |
| Baseline Reward | 0 | 0 |
| Training Updates | 200,000 | 1,000,000 |

# O Dark Treasure-Room Visualizations

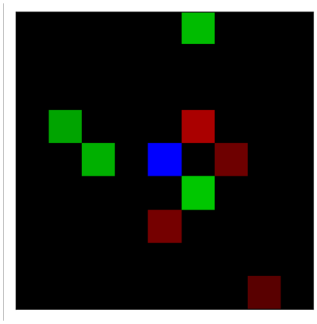


Figure 10: A visualization of the dark treasure-room. The agent's position is visualized by the blue square, positive rewards are in green, and negative rewards are in red, with the magnitude of reward being visualized by the colour intensity. When the agent enters a reward location it consumes the reward, and for that timestep is visualized as having the additive mixture of the two colours.

Here are example iterated First-Explore rollouts of the two trained policies, $\pi_{\text{explore}}$, $\pi_{\text{exploit}}$, visualized for a single sampled darkroom.

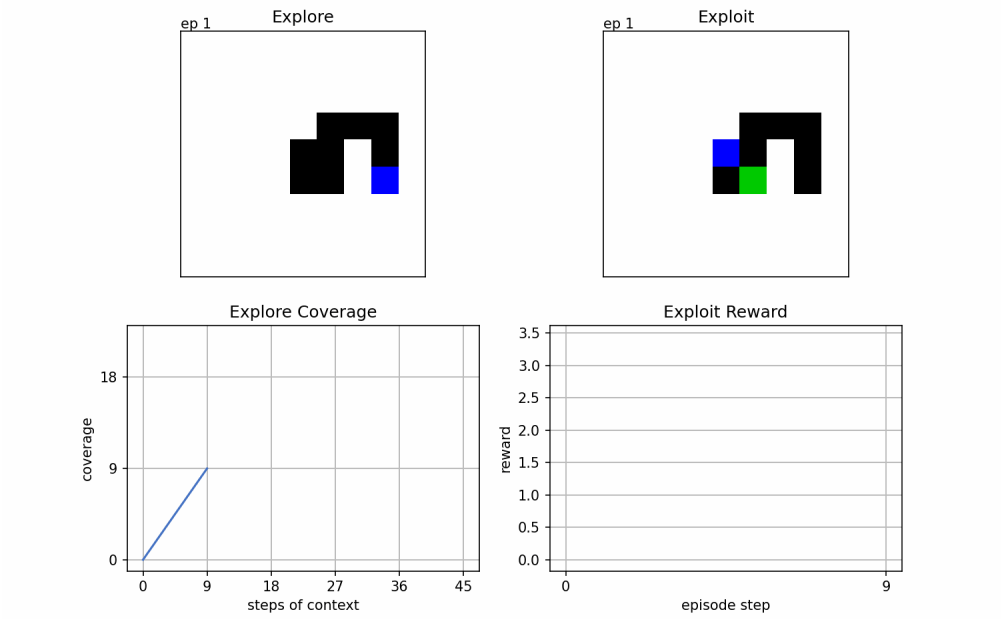



Figure 11: The first (First-Explore) explore episode. **Top left** visualizes the last step of a First-Explore explore episode, with the locations that are not in the cumulative context being coloured white, as the agent is blind to them (having no observations or memory of those locations). This figure plots the end of the first exploration, and shows a reward has been found. **Bottom left** visualizes the coverage of the cumulative context by plotting the total number of unique locations visited by the exploration against the cumulative episode step count. In this explore, the agent never doubled back on itself, which is good as it is optimal to have as many unique locations visited as possible. **Top right** visualizes a step in a First-Explore exploit episode, with the locations that are in context visualized. The agent can effectively 'see' those locations in its memory. **Bottom right** plots the exploit reward against the exploit episode timestep. As this figure plots before the start of the exploit episode, the agent has yet to move and encounter rewards, but will have done so in the subsequent visualizations.

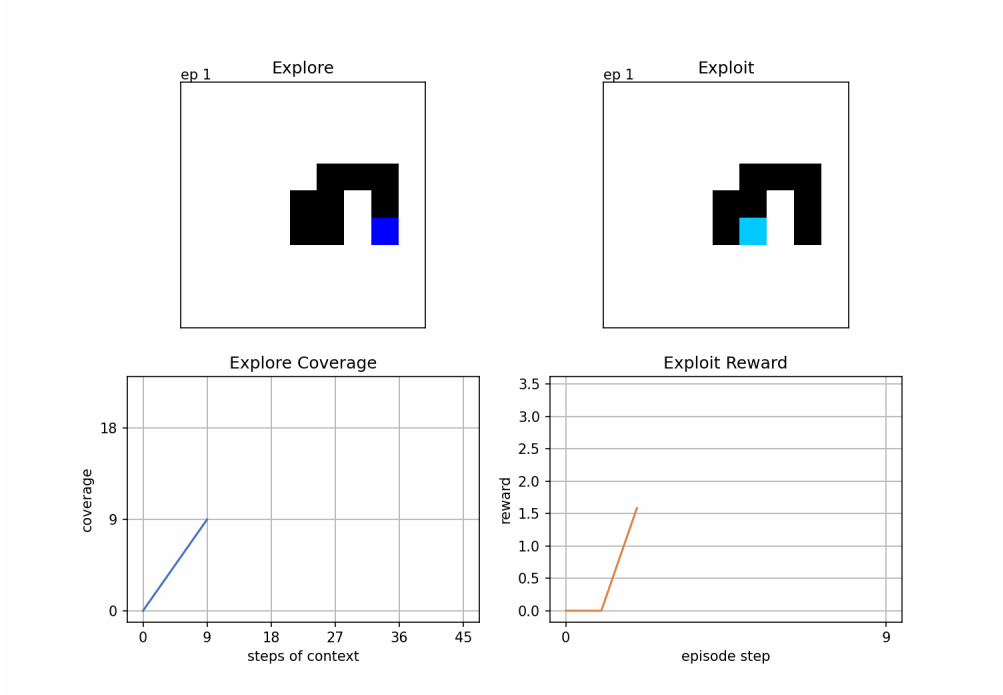



Figure 12: The first (First-Explore) exploit episode. This figure uses the same visualization design as Fig. 11. **Left top and bottom** are the same as in Figure Fig. 11, and of the explore context, not the current exploit episode. **Right top**, the agent (the light blue square) has found the reward in the first two steps. Consuming the reward is visualized by the agent colour and the reward colour being combined. **Right bottom**, the associated episode reward is shown.

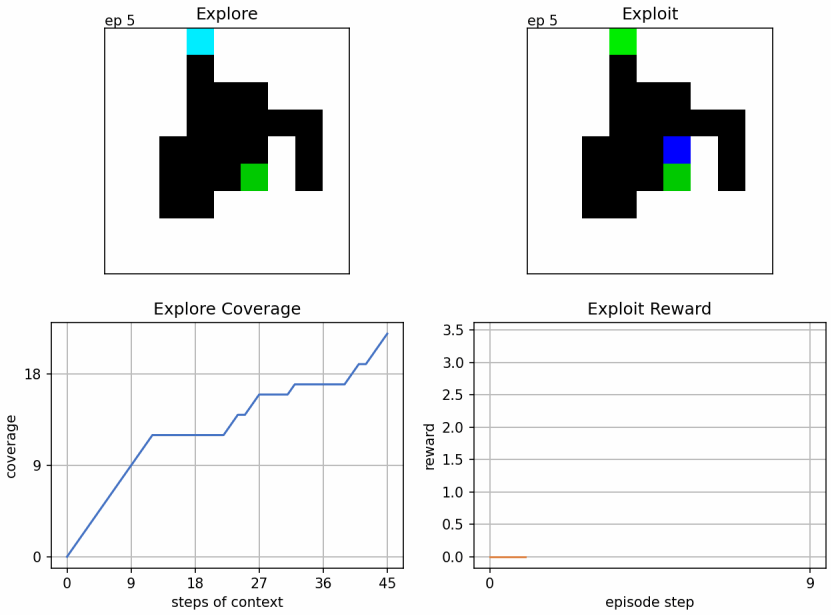



Figure 13: The fifth (First-Explore) explore episode. At the end of the 5th explore episode the agent has discovered a new positive reward at the top of the room, and can now 'see' it in memory. The new information presents an opportunity for the exploit policy to obtain both rewards, but it only has exactly enough time-steps in an episode to navigate to do so, and thus cannot make a mistake navigating.

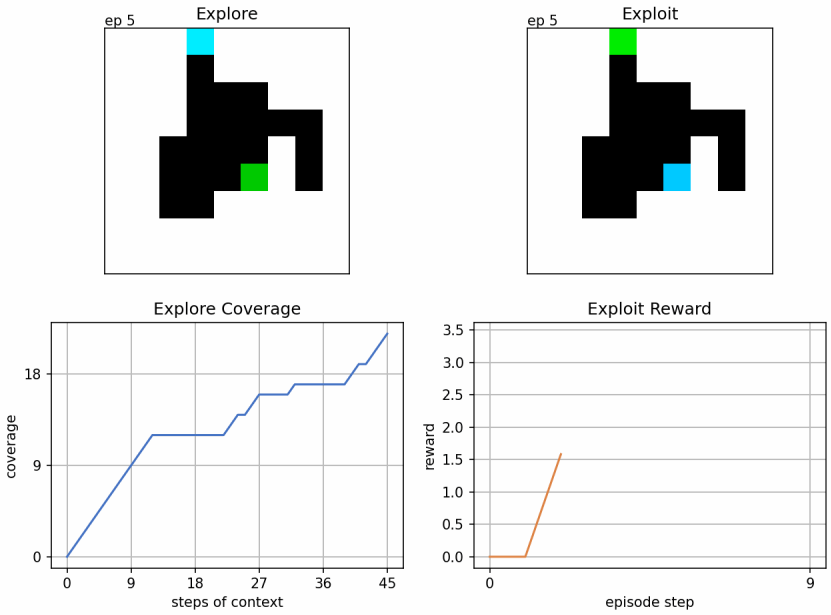



Figure 14: The first reward of the fifth (First-Explore) exploit episode. Two steps into the episode the agent (in consuming, light blue) has consumed the nearby reward.

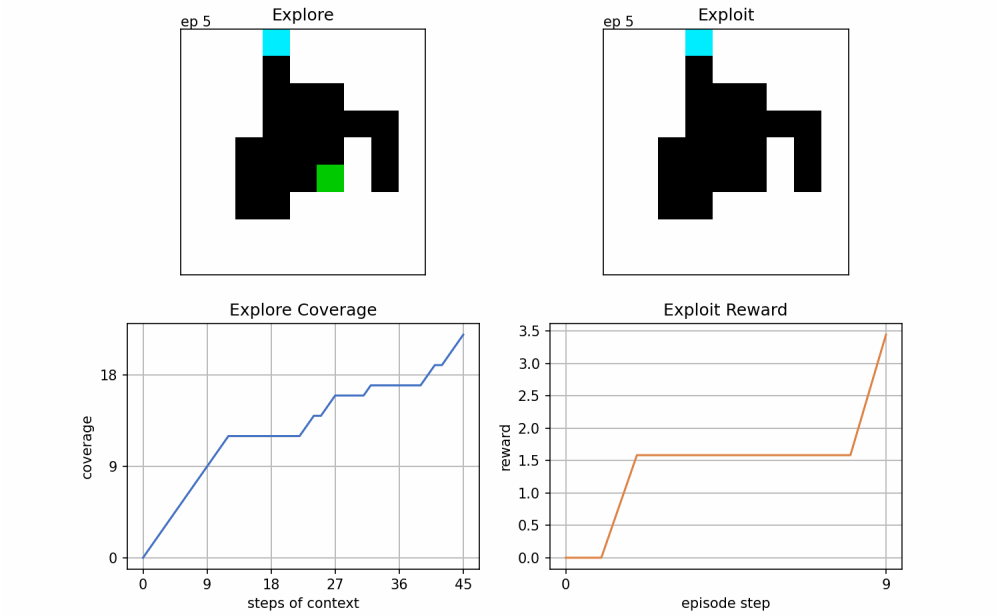



Figure 15: The end of the fifth (First-Explore) exploit episode. After consuming the nearby reward the agent has reached the newly discovered reward at the top of the room and consumed it. This success required making no mistakes and pathing first to the nearby reward then to the top one on the first try. This inference is possible because the quickest the agent can reach both rewards is exactly the length of the episode (9 steps). The pathing in this episode is an example of intelligent exploitation, as after the information reveal (the reward at the top) of a single episode the agent appropriately changes its policy based on the context and using the learnt environment prior (e.g., how to navigate), produces a highly structured behaviour (pathing with no mistakes).