RISING TIDE: OVERCOMING LEARNT EXPLORATION AVOIDANCE

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Rising Tide, a new algorithmic approach for learning performant policies that, unlike reinforcement learning methods, does not explicitly maximise expected return. Given a base policy, Rising Tide learns a sequence of policies indexed by return threshold, each approximating the base policy restricted to trajectories above that threshold, and trains higher-threshold policies by bootstrapping from lower-threshold ones. This design addresses a failure mode we identify as Learnt Exploration Avoidance (LEA): the tendency of RL agents to actively suppress exploratory behaviour because it initially appears suboptimal. We show that, in LEA-inducing environments, LEA can prevent standard RL methods from reaching optimal behaviour even with effectively unlimited training time. Rising Tide is architecture-agnostic; we demonstrate it with Transformer, LSTM, and fully connected architectures. To evaluate LEA in standard RL, we introduce Tree Climb, a new benchmark designed to isolate LEA. On three established meta-RL domains, Rising Tide delivers consistently strong performance, and on Tree Climb it is the only method to solve either of the two hardest variants.

1 INTRODUCTION

Reinforcement Learning (RL) seeks to learn complex, high-performing behaviour through trial and error by iteratively updating a single policy to maximise a reward signal from the environment (e.g., money won in a hand of poker). Agent–environment interaction is structured into episodes. In each episode, the agent repeatedly acts in the environment, which returns new observations and rewards (e.g., cards dealt and money won or lost). RL algorithms are designed to maximise the expected total reward accumulated within an episode, known as the *episode return*. RL proceeds via reinforcement: improving the policy by increasing the probability of actions associated with high episode return and decreasing the probability of actions associated with low return[a].

Learning via trial and error has several advantages. The method does not require a training dataset of examples, and because of this, it is not limited by the *quality* of those examples either. RL illustrates this potential, being employed in a diversity of applications such as controlling plasma in a fusion reactor (Degrave et al., 2022), molecule design (Zhou et al., 2019) and robotic control (OpenAI et al., 2019), as well as achieving super-human performance in many domains, such as chess and Go (Silver et al., 2018). This performance depends on algorithms exploring the space of behaviours sufficiently well to iteratively converge on good behaviour. The difficulty of such exploration is widely recognised as a central and longstanding challenge, and has motivated a large body of work over many years (Schmidhuber, 1991; Lehman & Stanley, 2008; Bellemare et al., 2016; Zintgraf et al., 2021; Ecoffet et al., 2019; Oudeyer & Kaplan, 2007; Barto, 2012; Lehman & Stanley, 2011; Cully et al., 2015; Conti et al., 2018). Many of the most valuable problems, however, remain intractable for RL because exploration is so difficult.

We identify a new exploration failure mode: in hard-exploration domains, RL can actively train policies to *avoid* acquiring and refining new behaviours.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

[a] See Appendix A for formal definitions.

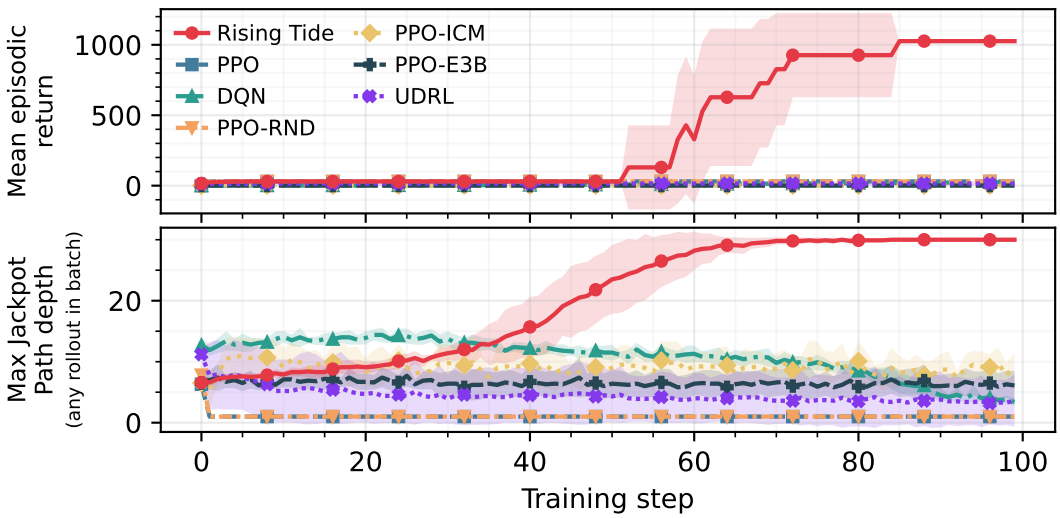


Figure 1: Training on the Combined Modifications Tree Climb environment. Top: return. Bottom: maximum Jackpot Path depth across batched rollouts. Both panels show mean ± one standard deviation over 10 seeds during training. Only **Rising Tide (red •)** persists along the Jackpot Path, which is initially suboptimal but optimal once completed, and thus achieves the jackpot.

Learnt Exploration Avoidance (LEA)

1. In RL, newly discovered behaviours are initially unpractised (i.e., unoptimised by training). For example, a humanoid robot may walk erratically and often fall over.
2. These behaviours are then evaluated against the agent's current, highly optimised behaviour (e.g., reliable crawling).
3. This unfair comparison can make potentially high-performing behaviours appear inferior solely because they are unpractised (e.g., walking appears worse than crawling).
4. The agent therefore learns to avoid the unpractised behaviour (e.g., actively *avoiding* standing up, to eliminate the chance of falling over).
5. Because the agent avoids these behaviours, it rarely practises them, and so potentially high-performing behaviours stay avoided, unpractised, and low performing.
6. Pathologically, if generalisation works well, *any* action or state that enables subsequent exploratory behaviour may be *completely* avoided, preventing meaningful exploration. For instance, each time the agent enters an unfamiliar area, it immediately retreats.
7. We term this general dynamic Learnt Exploration Avoidance (LEA).

Rising Tide is a new algorithmic approach for learning high-performing policies. Like RL, it does not require a dataset; unlike RL, it avoids LEA. This paper:

- Characterises LEA as a general failure mode in RL
- Introduces Rising Tide, a new algorithmic approach
- Explains how Rising Tide promotes exploration rather than suppressing it
- Introduces Tree Climb, a fast, novel, and challenging benchmark for studying LEA in RL.
- Demonstrates that Rising Tide delivers consistently strong performance on the existing environments we identify as LEA-inducing, matching or exceeding the returns of prior methods, and on Tree Climb is the only method to solve either of the two hardest variants.

2 Related Work

Intrinsic Motivation (IM): The dominant RL approach to exploration augments the environment reward with an exploration bonus, such as $+1$ for visiting a previously unseen state (Pathak et al., 2017; Burda et al., 2019b; Henaff et al., 2022). While such bonuses can help agents explore sparse-reward environments, a bonus can also misalign the objective, so that behaviour pathological for the true task becomes optimal under the augmented one: for example, fixating on a noisy TV instead of exploring (Burda et al., 2019a). Further, the modified reward landscape may *still* contain suboptimal

local optima. Training may then drive the policy towards a single behaviour—suppressing the pursuit of competing alternatives that may lead to the global optimum—and induce LEA (Figure 1: the IM methods **PPO-ICM (yellow ⧫)**, **PPO-E3B (navy ✚)** and **PPO-RND (orange ▼)** perform poorly).

First-Explore: Rising Tide is inspired by a problem highlighted by the First-Explore algorithm (Norman & Clune, 2024): when information is costly to acquire but initially difficult to use, agents may learn to avoid acquiring it altogether (e.g., an agent that cannot yet read learns to avoid buying maps). This dynamic is a narrow subset of LEA. First-Explore addresses this problem by partitioning learning into pure exploration and pure exploitation. In the exploration phase, it ignores the immediate cost of acquiring information. However, this design requires a sharp transition from exploration to exploitation, making it ill-suited to domains that require interleaving exploration and exploitation within an episode.

Although First-Explore addresses a subset of LEA, the two methods are fundamentally distinct. First-Explore was developed for meta-RL and is only applicable in that setting.[b] Rising Tide has no such restriction and, despite being a general algorithm designed for standard-RL environments, performs strongly on all of First-Explore's meta-RL domains. Figure 7 in Appendix J illustrates how the two algorithms solve the same environment differently.

Return-conditioned policy learning methods recast RL as a supervised learning problem. Upside-Down Reinforcement Learning (UDRL) (Schmidhuber, 2019) conditions on desired returns over varying horizons (e.g., achieving return x within y steps), while Reward-Conditioned Policies (RCP) (Kumar et al., 2019) condition on a return target without an explicit horizon variable. In both cases, however, training conditions on the exact realised return of each episode, rather than on whether return meets or exceeds a target threshold.

Training on all trajectories can cause exact return conditioning to collapse onto a single behavioural mode: low-return trajectories may simply be degraded executions of a high-return policy, rather than reflecting a distinct strategy. Under such unconstrained feedback, it is unclear what control objective, if any, is induced by the resulting sequence of archive and policy updates. The only unambiguous objects these procedures specify are their final trained policies, which may vary across runs.

More generally, it is unclear what advantage, if any, this recasting of RL provides. Indeed, UDRL performs provably poorly in stochastic environments (Štrupl et al., 2022). In particular, there is neither a theoretical claim nor empirical evidence that it aids exploration, and our results (Figure 1: **UDRL (purple ✖)** fails alongside the RL baselines) show that UDRL is as susceptible to LEA as conventional RL, while also optimising for an unclear objective. In contrast, Rising Tide makes a precise claim about LEA resistance and is motivated by approximating *the base policy* conditioned on meeting *or exceeding* a return threshold.

Novelty Search and Quality Diversity: though algorithmically distinct and differently motivated, Rising Tide has a conceptual connection to both Novelty Search and Quality Diversity (Lehman & Stanley, 2011; Cully et al., 2015; Conti et al., 2018) methods. Appendix C.1 expands on this connection.

3 METHOD

Modelling Objective and Intuition: Unlike RL, which optimises a policy to maximise expected return, Rising Tide models, for each return threshold C, the distribution over behaviours that achieve *at least* C. This distribution is defined relative to a base policy, which determines how likely actions are before filtering for returns. This base policy may range from simple (e.g., a state-independent action distribution) to complex (e.g., a foundation model).

For each possible episode return C, Rising Tide aims to model the action distribution produced by the base policy, restricted to trajectories with return at least C. This corresponds to rolling out many episodes, sorting them by return, discarding those below a return threshold, and then behaviour cloning the remainder. This direct approach to modelling the base policy is generally infeasible. First, it would require behaviour cloning separately for each return threshold. Second, for generic base policies such as random action selection, high-return trajectories may be so rare that a feasibly sized batch would almost surely miss the optimal behaviour. Rising Tide addresses these two issues

[b] A specific RL setting that often requires specialised methods; see Appendix A.1 for further details.

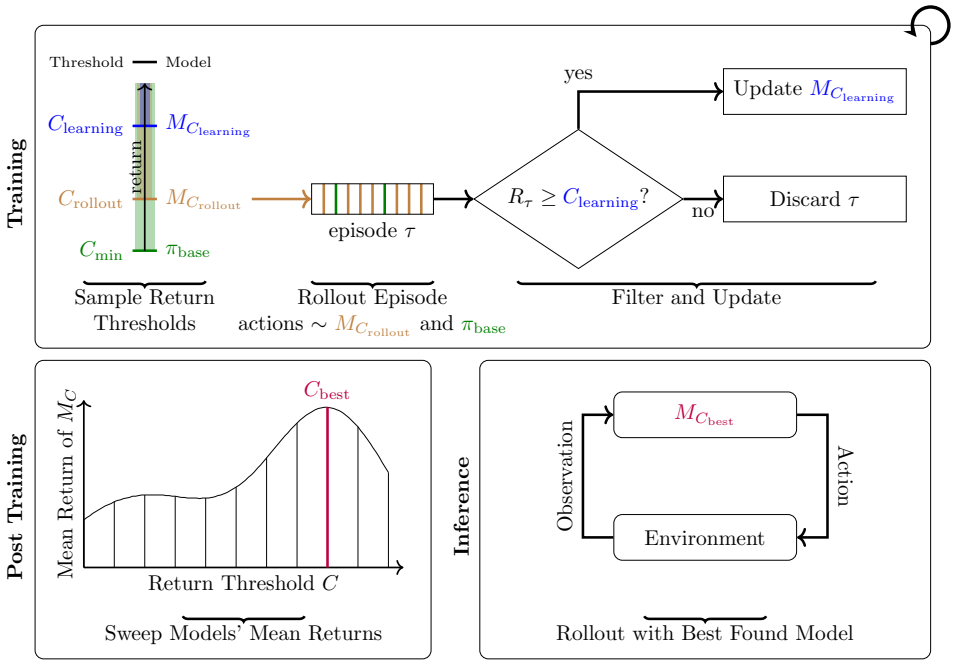


Figure 2: The Rising Tide paradigm, as detailed in Algorithm 1.

by learning a parametrised family of models, one for each return threshold, and then bootstrapping high-return models from low-return ones (Figure 2 and Algorithm 1).

After all model updates, the model with the highest mean return is selected for inference. This step is necessary because the model trained on the highest-return trajectories may be dominated by *unreliable* behaviours. Intuitively, for high return thresholds, Rising Tide behaviour-clones the best episodes, which may depend on intrinsically unreliable behaviours (e.g., running across slippery ice). If such behaviours have lower expected return, this final step selects against them.

Algorithm 1: Rising Tide

Input: a base policy π_{base}, which is fixed throughout training.

0 Initialise a parametrised family of models $\mathcal{M} = \{M_C\}$ indexed by C, where C ranges over the set of possible episode returns. Each model maps observations[A1] to a distribution over actions, $M_C : O \to \Delta(A)$.

1 Training loop:

1.1 Sample the return thresholds:[A2]

i. Sample rollout threshold C_{rollout} from the range of possible episode returns.

ii. Sample learning threshold C_{learning} from the range of possible returns, subject to $C_{\text{learning}} \geq C_{\text{rollout}}$[A3].

1.2 Roll out an episode τ:

- At each timestep, follow the rollout model $M_{C_{\text{rollout}}}$ with probability $1 - \epsilon$, where $\epsilon \in [0, 1]$ is a small hyperparameter.
- Otherwise, follow the base policy π_{base}.

1.3 Filter and update:

- Draw a perturbation $\xi_\tau \sim \mathcal{N}(0, \sigma^2)$ once for the episode, where $\sigma \geq 0$ is the filter temperature ($\sigma = 0$ gives a hard threshold) (Appendix G.1).
- *If* the perturbed episode return satisfies the learning condition, $R_\tau + \xi_\tau \geq C_{\text{learning}}$, *then* update the $M_{C_{\text{learning}}}$ model to maximise the log-likelihood of the observed actions.
- *Otherwise*, discard the episode.

2 Post-training: search for the return threshold C_{best} that maximises the mean rollout return (e.g., by sweeping over candidate values): $C_{\text{best}} = \arg\max_C \mathbb{E}(R_\tau | \tau \sim M_C)$

3 Inference: perform rollouts with the model $M_{C_{\text{best}}}$ identified in the previous step.

[A1]This input can be generalised to include the full history of past observations, actions, and rewards.

[A2]Note: C_{learning} could, in general, be sampled *after* the episode rollout and even be allowed to depend on the achieved episode return. However, such changes require care and potentially modifying the training loss.

[A3]To increase sample efficiency, multiple learning thresholds may be sampled for each rollout threshold.

Because C_{learning} is greater than or equal to C_{rollout}, information flows only in one direction: from the models of low return up to the models of high return. **This one-directional transfer of information (the *rising tide*) is crucial**. Without this constraint, lower-threshold models would learn from higher-threshold models, potentially collapsing the family of models into a single high-return model, and this family is key to addressing LEA; see Appendix F for ablations.

By learning a family of models, Rising Tide can develop rare and complex behaviour:

- **High-return models** capture and refine the highest-return behaviours discovered so far, even when those behaviours are initially unreliable (e.g., walking but often falling over). With practice, they may improve and eventually yield reliable high return.
- **Low-return models** explore behaviours with low immediate return that may, with further development, lead to high return (e.g., persistently wandering a landscape enables discovering treasure).

Unlike RL, which tends to collapse to narrow policies, Rising Tide directly optimises to learn distributions: each return-threshold model captures the full distribution of behaviours found in the associated filtered episodes (including their relative frequencies). Consequently, unlike RL's LEA dynamic, the discovery of one reliable behaviour does not suppress alternatives. For example, once an agent discovers crawling, lower-return models still explore behaviours with lower immediate reward, such as standing without taking any steps. These initially suboptimal behaviours can provide useful intermediate practice, potentially allowing repeated attempts at standing to eventually support learning to walk. Together, these properties enable Rising Tide to learn in environments that can completely stymie RL.

3.1 Implementation Details

Rising Tide was validated on a range of architectures, including a fully connected network, an LSTM, and a Transformer. In each experiment, the model family was instantiated as a single architecture that takes the return threshold as input, as illustrated in Figure 3. In all cases, Rising Tide used uniform random selection over the discrete action space as a simple base policy.

The models were optimised with AdamW (Loshchilov & Hutter, 2019) to minimise the cross-entropy between the action distribution predicted by $M_{C_{\text{learning}}}$ and the action distribution induced by the non-discarded episodes[c]. On Tree Climb, the loss was also augmented with a small entropy regularisation term (none was used in the meta-RL runs). For on-GPU efficiency, each training step was batched across two dimensions: learning threshold and rollout threshold, with multiple learning thresholds per individual rollout.

Similar to Mnih et al. (2015), rather than using a single set of parameters for both rollout generation and learning, the rollout parameters lagged behind the learning parameters, and were only updated to match them every 1,000 batched rollouts. This periodic synchronisation improved training stability.

Rollout thresholds. At each training step, the rollout threshold C_{rollout} was drawn uniformly from $[C_{\min}, v_{\max}]$, where $v_{\max}$ is the highest episode return achieved so far in training and $C_{\min}$ is a hyperparameter; a simple heuristic for $C_{\min}$ is the mean return of the base policy. Where a rare, very large return exists (e.g., Tree Climb's $+1000$ jackpot), $v_{\max}$ was capped at a hyperparameter $c_{\max}$, since otherwise such extreme returns, once discovered, can dominate the uniform sampling of return thresholds (Appendix I). If C_{rollout} fell at or below a threshold u (a hyperparameter), the whole episode was rolled out with the base policy instead.

[c]See the Filter and Update step in Algorithm 1.

Learning thresholds. The learning threshold was then set to $C_{\text{learning}} = C_{\text{rollout}} + \delta + X$ with X exponentially distributed with mean θ, where the minimum gap $\delta \geq 0$ and the scale θ are hyperparameters ($\delta = 0$ imposes no minimum gap).

Evaluation. *After training*, evaluation in the meta-RL environments selected actions by always taking the most probable action rather than sampling from the action distribution, matching the evaluation approach of First-Explore. On Tree Climb, both the argmax policy and the stochastic policy (sampling from the learnt distribution with $\epsilon = 0$) were evaluated; each results table states which it reports.

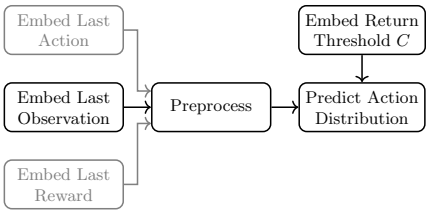


Figure 3: Rising Tide's architecture. In the standard RL setting, the **action and reward embeddings** were omitted, and feed-forward layers were used in place of a sequence model.

4 Experimental Setup

4.1 Domains

Rising Tide was evaluated on four environments: three First-Explore meta-RL domains (further details in Appendix J) and Tree Climb, an RL benchmark introduced in this paper and described below. These meta-RL domains were selected because the problem identified by First-Explore can be viewed as a narrow special case of LEA. Tree Climb isolates LEA in a broader standard-RL setting, outside the scope of First-Explore.

How the meta-RL domains induce LEA. Bandits with One Fixed Arm provides a representative example: it is a distribution of multi-armed bandit problems in which the first arm is guaranteed to be rewarding but rarely optimal. The aim is to maximise cumulative reward over 100 pulls in newly sampled bandits. Always sampling the first arm is easy to learn and performs better than sampling randomly. However, learning this strategy suppresses exploration and prevents discovery of the superior strategy: identifying and exploiting the best arm in each environment.

The other meta-RL domains exhibit the same LEA-inducing structure:

1 The agent interacts with a newly sampled environment over a sequence of episodes.
2 To maximise cumulative reward, the agent must *first explore* to obtain information about the environment and *then exploit* that information in later episodes.
3 However, in each domain, exploration during an episode decreases episode return and is only valuable when followed by skilled exploitation in a *subsequent* episode.
4 Initially lacking such skill, agents learn to avoid exploration, preventing them from learning a better strategy.

This form of LEA has a temporal decomposition. In each instance, the best behaviour within a meta-RL interaction can be approximated by pure exploration followed by pure exploitation. First-Explore is designed to solve this *specific* form of LEA, but Rising Tide nonetheless performs strongly on all three meta-RL domains.

New benchmark: Tree Climb.

Tree Climb is a fully observable deterministic environment designed to exhibit LEA and isolate it from other exploration difficulties, such as reward sparsity.[d] The agent has k timesteps to traverse a full ternary tree of height k. Three actions climb to the node's children, and a backtrack action moves the agent to its parent, negating any parent-to-child transition reward; backtracking from the root leaves the agent there. Nodes are identified by ternary tree addresses, and the observation pairs a fixed pseudorandom fingerprint of the current address with the timestep. Each instantiation has two root-to-leaf paths: the locally learnable Easy Path and the harder but optimal Jackpot Path, which share only the root. All nonzero rewards occur on transitions along these paths.

We study three variants. In the Reward-Gated Jackpot Path Variant, both paths' actions vary across depth, but the first three Jackpot Path climb transitions yield zero reward. In contrast, in the Constant-Action Easy Path Variant, there is no reward gating, but the Easy Path uses the same semantic action across depths. The Combined Modifications Variant combines the reward-gated Jackpot Path with

[d] More precisely, learning signal sparsity: a constant $+1$ reward per step is dense but uninformative.

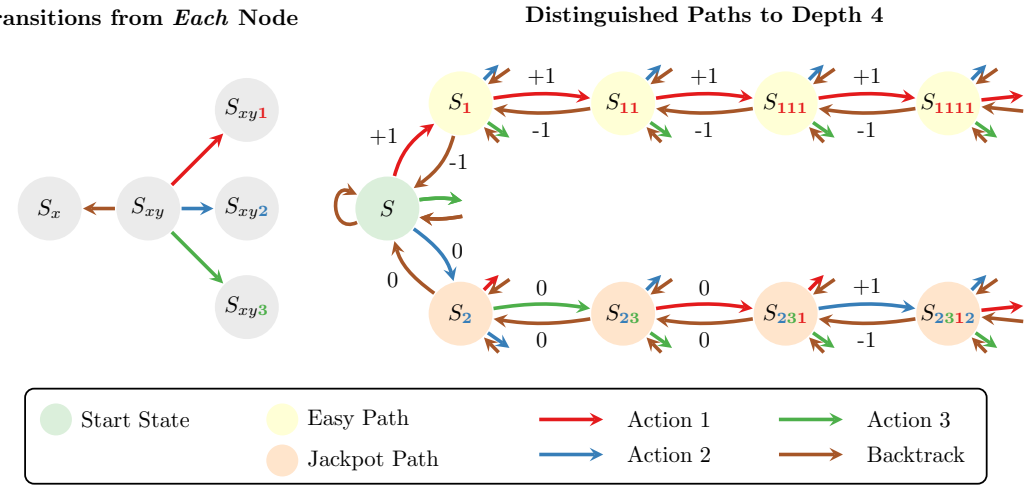


Figure 4: **Tree Climb environment — Combined Modifications variant.** The agent traverses a full ternary tree of height k. Rewards occur only along two distinguished root-to-leaf paths, illustrated on the right; for clarity, the paths are shown only up to depth 4, and all non-distinguished nodes are omitted. The Jackpot Path is harder to learn than the Easy Path but terminates in a large reward, yielding the highest return.

the constant-action Easy Path. Without reward gating, on-path climb transitions yield $+1$, except for the final Jackpot Path transition, which yields $+1000$.

The final state may be any node in the tree, yielding $\sum_{i=0}^{k} 3^i = (3^{k+1}-1)/2$ possible final states: approximately 3.1×10^{14} for $k = 30$. This large state space distinguishes Tree Climb from exploration benchmarks that pair adverse reward incentives with comparatively small state spaces. For example, in Deep Sea (Osband et al., 2020), the agent must discover a single action sequence against such an incentive, but learning can exploit the compact state space: for horizon k, the state space has size k^2, and the terminal state lies among only k possibilities.

Tree Climb gives full expression to RL algorithms' natural tendency to learn to avoid exploration. In each variant, the Easy Path is easier to learn and, once learnt, becomes locally optimal. The agent can then *actively avoid* the globally optimal Jackpot Path: if it stumbles onto it, the backtrack action lets it return to the root and resume the Easy Path. LEA-susceptible algorithms can therefore optimise not merely for suboptimal behaviour, but for avoiding the highest-return trajectory entirely.

4.2 Baselines

On the meta-RL environments, Rising Tide was compared with First-Explore and with the baselines evaluated in the First-Explore paper: RL^2 (Duan et al., 2016; Wang et al., 2016), VariBAD (Zintgraf et al., 2020), HyperX (Zintgraf et al., 2021) (an IM-based meta-RL method), Upper Confidence Bound (UCB1) (Duan et al., 2016), and Thompson Sampling (TS) (Thompson, 1933). Each algorithm was evaluated using its official implementation. For these baselines, we used the hyperparameters reported in First-Explore, except for a higher reward discount factor and larger controllers, which together improved baseline performance.

We standardised wall-clock training time across algorithms at 50 hours per run, increasing the training budget for the baselines relative to First-Explore. Due to the computational cost—e.g., 50 GPU-hours per run for First-Explore—we did not re-sweep the baselines' hyperparameters. Each baseline uses the configuration reported by First-Explore, with the modifications described above, while Rising Tide was configured based on preliminary experiments.

As hyperparameter tuning was not standardised across methods, these comparisons are not necessarily like-for-like. Accordingly, on these domains we claim only that Rising Tide performs strongly against baselines configured according to prior work.

On Tree Climb, the new standard-RL benchmark, we compared Rising Tide with Proximal Policy Optimisation (PPO), Deep Q-Network (DQN), PPO with Random Network Distillation (PPO-RND), PPO with an Intrinsic Curiosity Module (PPO-ICM), PPO with E3B, and Upside-Down Reinforcement Learning (UDRL). The algorithms were implemented using established JAX codebases, with the exception of UDRL, for which we wrote a JAX reimplementation of the most-starred GitHub repository. This reimplementation was necessary to compare against the algorithm within a reasonable training time, and releasing the codebase is a small contribution of this paper. For each algorithm, hyperparameters were selected using a two-stage procedure, first by performing random search over 50 configurations, and then by evaluating the top three configurations over 10 seeds each. We report results for the best-performing configuration. For maximum fairness, the number of environment interactions was kept the same across algorithms (3×10^9 environment steps per run), along with the architecture and rough parameter count[e]. Appendices P to R detail the sweeps and the exact hyperparameters.

The meta-RL environments: To evaluate the meta-RL environments, between 5 and 10[f] runs of each algorithm were trained. Due to the high stochasticity of these environments, the final evaluation measures each training run's performance as the mean total reward over 10,000 independent rollouts (20,000 for RL^2 on Bandits with One Fixed Arm; UCB1 and TS, which require no training, were evaluated as five runs of 1,280 rollouts), and then reports the mean and standard deviation across training runs.

5 RESULTS

On the **meta-RL environments**, Rising Tide performs strongly, achieving the highest mean run return in each domain (Table 1); exact two-sided Mann–Whitney U p-values for each comparison are given in Appendix D.1. On **Tree Climb**, where all methods received the same hyperparameter search, **Rising Tide outperformed all other methods** on the two hardest variants, Reward-Gated Jackpot Path and Combined Modifications (Table 2): it reached the jackpot in 10/10 runs, whereas every baseline reached it in 0/10 (two-sided Fisher exact test, $p = 1.08 \times 10^{-5}$ for each comparison). On the Constant-Action Easy Path variant, PPO-RND learnt the Jackpot Path in 8/10 runs, but interestingly baseline runs that did not reach the jackpot within the first 5% of training never did (Figure 9).

Table 1: Results on the meta-RL environments.

Bandits with One Fixed Arm		Dark Treasure Room		Ray Maze	
Rising Tide	129.61 ± 0.78	Rising Tide	2.41 ± 0.25	Rising Tide	0.49 ± 0.01
First-Explore	127.21 ± 1.62	First-Explore	2.37 ± 0.14	First-Explore	0.40 ± 0.01
RL^2	107.42 ± 11.35	RL^2	0.10 ± 0.25	RL^2	0.00 ± 0.00
UCB1	116.83 ± 0.55	VariBAD	0.03 ± 0.56	VariBAD	0.00 ± 0.00
TS	123.28 ± 2.52	HyperX	-0.91 ± 1.05	HyperX	0.00 ± 0.00

Tree Climb is difficult because of LEA. When backtracking is enabled, Rising Tide outperforms the other methods on the Combined Modifications variant (Figure 1 and Table 2). To isolate this effect, the Combined Modifications Variant was modified by replacing the backtrack action with an always-incorrect action that precludes any subsequent reward. Although this modification leaves strictly fewer rewarding trajectories, DQN solves the resulting environment, matching Rising Tide's jackpot rate and return (Table 8 in Appendix D.2). Thus, DQN fails in the full environment because backtracking provides a learnable way to avoid continued exploration (Figure 21), but succeeds once that avoidance route is removed (Figure 22). Surprisingly, the other baseline methods still fail under this ablation. Even the initial choice between the Easy Path and the Jackpot Path is sufficient to induce LEA, with these agents choosing the Easy Path almost exclusively. As a result, the Jackpot Path is sampled too rarely for optimisation to reinforce it, and the agents never learn it. Appendix S provides heatmaps detailing the training dynamics associated with these effects.

[e] As closely as the algorithms allow, e.g., PPO uses an actor and a critic, whereas DQN uses a single Q-network.

[f] Rising Tide and First-Explore were run an additional five runs to reduce variance in the estimates.

Table 2: Results across the three Tree Climb variants over ten training runs.

	Reward-Gated Jackpot Path		Constant-Action Easy Path		Combined Modifications	
Algorithm	Depth[a,c]	Return[a,b,d]	Depth[a,c]	Return[a,b,d]	Depth[a,c]	Return[a,b,d]
Rising Tide	**30.0**	**1026.0**	**30.0**	**1029.0**	**30.0**	**1026.0**
DQN	1.0	30.0	9.7 ± 13.3	329.7 ± 457.8	1.0	30.0
PPO	1.0	30.0	15.5 ± 14.5	529.5 ± 499.5	1.0	30.0
PPO-E3B	4.1 ± 1.8	0.6 ± 0.4	8.3 ± 11.3	225.5 ± 401.8	4.2 ± 1.7	0.4 ± 0.3
PPO-ICM	5.3 ± 1.2	0.6 ± 0.1	16.3 ± 13.8	523.7 ± 505.4	5.6 ± 1.6	0.7 ± 0.1
PPO-RND	1.0	23.6 ± 11.9	24.2 ± 11.6	829.2 ± 399.6	1.0	30.0
UDRL	1.9 ± 0.9	0.4 ± 0.4	1.8 ± 1.2	6.6 ± 11.7	1.6 ± 1.0	15.2 ± 14.8

[a] Hyperparameters selected to maximise Jackpot Path depth, favouring the baselines.
[b] Returns use the stochastic policy (argmax for DQN, which has no stochastic mode).
[c] Per run, the greater of the argmax and stochastic policies' Jackpot Path depth. Mean depth $= 30$ corresponds to reaching the jackpot in every seed. Per-seed jackpot rates are reported in Appendix D.2.
[d] Across the sweep, all baselines mastered the Easy Path, $R = 30.0$; low Return at the depth-maximising setting reflects that selecting for depth is at odds with maximising return.

6 Discussion and Limitations

To isolate LEA, we evaluate Rising Tide in controlled domains designed or selected to make LEA especially salient. Although these environments use small discrete action spaces and short horizons, they are extreme in other key respects: Tree Climb, introduced here, has over 10^{14} discrete states, and the selected meta-RL environments are highly stochastic with large state spaces. Showing that Rising Tide addresses these stark LEA-inducing environments is itself a central contribution. Extending the method to long-horizon settings remains important future work. One promising approach is to combine Rising Tide with curriculum training, progressively introducing tasks, horizons, or return thresholds rather than requiring full-problem credit assignment upfront.

Rising Tide targets LEA, not reward sparsity. It changes the learning dynamics that cause LEA, so that learning an easy but suboptimal behaviour does not suppress the exploration that could uncover better ones. It does not, by itself, address sparse exploration, like finding a needle in a haystack. Novelty-driven methods such as intrinsic motivation target this discovery problem, and the two approaches may potentially complement each other, as Appendix E details.

As described in Section 4.2, compute constraints prevented a like-for-like comparison on the meta-RL domains. Rising Tide's consistently higher mean returns therefore reflect a comparison against the baselines as configured in prior work, with the improvements described here, rather than against the best achievable configuration of each method.

Rising Tide differs fundamentally from RL and does not directly optimise expected reward. Although comparatively resistant to LEA, it may exhibit distinct failure modes of its own. Appendix G describes one such issue along with possible mitigations. Finally, the present results are empirical. Even under idealised assumptions of sufficient model capacity, we do not yet provide theoretical guarantees regarding convergence or optimality. Developing a theoretical foundation for Rising Tide is an important direction for future work, but lies beyond the scope of this paper.

7 Conclusion

This paper characterises **LEA**, a general failure mode in RL. The new benchmark, **Tree Climb**, isolates a simple instance of this general challenge: how to continue exploring in the presence of easily discovered but suboptimal solutions, when the unexplored state space can be actively avoided. We believe addressing this challenge is central to solving hard exploration problems. To address LEA, we introduce **Rising Tide**, a new algorithm class whose properties differ from those of existing RL methods. Unlike RL algorithms, which have been optimised over decades of research, Rising Tide is an early-stage method based on a fundamentally novel idea. Despite an equal and extensive hyperparameter search, none of the state-of-the-art RL methods we evaluate solve the two hardest Tree Climb variants; Rising Tide solves both. These three contributions advance the development of more powerful algorithms for learning and exploration in challenging domains.

AI USE STATEMENT

We have not used generative AI tools for any of the tasks for which the ICLR 2027 AI Policy requires disclosure. In particular, generative AI was not used to implement the methods studied in this paper, to develop hypotheses or research ideas, to design the experiments, or to interpret results. Additionally, we used generative AI tools for several tasks with recommended disclosure: (i) writing infrastructure code for hyperparameter sweeps, which launched the author-written training code; (ii) creating visualisations and figures, as specified by the authors; (iii) copy editing the text for readability; (iv) routine text tasks such as reformatting tables; and (v) preparing the code release for anonymous review, including flattening the directory structure, cleaning up relative imports, standardising file names, and checking for author-identifying information. Generative AI was not used for research ideation or literature search—however, it did spot several errors in our `.bib` file, which were then fixed by the authors. All AI-assisted work was carried out under close author supervision through specific, narrowly scoped instructions (e.g., adjusting the legend of a given figure, or correcting the grammar of a given sentence), and all outputs were reviewed by the authors. We take responsibility for the final content of this work, including text, claims or artefacts produced with the aid of generative AI.

ETHICS STATEMENT

This work is foundational research on a generic algorithm for solving reinforcement-learning problems, evaluated entirely in simulated benchmark environments (the three meta-RL domains of Norman & Clune (2024), and Tree Climb, a new benchmark introduced in this paper). It involves no human subjects, crowdsourcing, personally identifiable data, or data scraping. The released assets (the Rising Tide implementation, the Tree Climb environment, and a JAX reimplementation of UDRL) are synthetic and pose no misuse risk, and we are not aware of a direct path from this work to malicious use, fairness, privacy, or security concerns. All third-party code was used under its MIT or Apache 2.0 licence with upstream notices preserved (Appendix L). Our own code is released under the MIT-HUMAN-AI-RESEARCH licence (Appendix L).

REPRODUCIBILITY STATEMENT

Rising Tide is specified in Section 3 (Algorithm 1), with implementation details in Section 3.1 and soft filtering and return-threshold sampling in Appendices G.1 and I. Tree Climb is fully specified in Appendix K and the meta-RL domains in Appendix J. For every algorithm on Tree Climb we list the hyperparameter search grids (Appendix P), fixed hyperparameters (Appendix Q), and best configurations (Appendix R). On the meta-RL domains, the baseline hyperparameters follow First-Explore (Section 4.2), and the exact Rising Tide configurations are included in the supplementary code. Compute usage is reported in Table 16. The supplementary material contains, anonymised and under licence, the full code needed to reproduce every reported result: the reference implementation of Rising Tide (model, training loop, and the ablated training rules of Appendix F), the Tree Climb environment, our JAX reimplementation of UDRL, and the other Tree Climb baselines (adapted from CleanRL, PureJaxRL and Craftax), the two-stage hyperparameter-search pipeline with the exact search grids and configurations, the scripts that generate the tables and figures, and, for the meta-RL domains, the training scripts with the exact configurations and seeds of the reported Rising Tide runs, the evaluation protocol, and the official baseline codebases with our environment, configuration, and launch files, together with a README giving the commands for each experiment. Baseline code provenance and licences are given in Appendix L.

REFERENCES

Andrew G. Barto. Intrinsic motivation and reinforcement learning. In *Intrinsically Motivated Learning in Natural and Artificial Systems*, pp. 17–47. Springer, 2012.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. In *International Conference on Learning Representations*, 2019a.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019b.

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. In *Advances in Neural Information Processing Systems*, volume 34, pp. 15084–15097, 2021.

Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

Sebastian Dittert. PyTorch implementation of upside-down RL. `https://github.com/BY571/Upside-Down-Reinforcement-Learning`, 2020.

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.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-Explore: A new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

Mikael Henaff, Roberta Raileanu, Minqi Jiang, and Tim Rocktäschel. Exploration via elliptical episodic bonuses. In *Advances in Neural Information Processing Systems*, volume 35, pp. 37631–37646, 2022.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=68n2s9ZJWF8`.

Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*, 2019.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1179–1191. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf`.

Joel Lehman and Kenneth O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, volume 65, pp. 329–336, 2008.

Joel Lehman and Kenneth O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pp. 211–218, 2011.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16455–16468, 2022.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Thomas Jackson, Samuel Coward, and Jakob Nicolaus Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 35104–35137. PMLR, 2024.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Ben Norman and Jeff Clune. First-Explore, then exploit: Meta-learning to solve hard exploration-exploitation trade-offs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 27490–27528, 2024.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3878–3887. PMLR, 2018.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Ian Osband, Yotam Doron, Matteo Hessel, John Aslanides, Eren Sezener, Andre Saraiva, Katrina McKinney, Tor Lattimore, Csaba Szepesvari, Satinder Singh, et al. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020.

Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? A typology of computational approaches. *Frontiers in Neurorobotics*, 1:108, 2007.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2778–2787. PMLR, 2017.

Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*, 2019.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In Jean-Arcady Meyer and Stewart W. Wilson (eds.), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pp. 222–228. MIT Press, 1991.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144, 2018.

Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. Training agents using upside-down reinforcement learning. *arXiv preprint arXiv:1912.02877*, 2019.

Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

Miroslav Štrupl, Francesco Faccio, Dylan R. Ashley, Jürgen Schmidhuber, and Rupesh Kumar Srivastava. Upside-down reinforcement learning can diverge in stochastic environments with episodic resets. *arXiv preprint arXiv:2205.06595*, 2022.

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.

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.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Qinqing Zheng, Amy Zhang, and Aditya Grover. Online Decision Transformer. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 27042–27059. PMLR, 2022.

Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N. Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *Scientific Reports*, 9(1):10752, 2019.

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 Representations*, 2020.

Luisa M. 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 *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12991–13001. PMLR, 2021.

APPENDIX CONTENTS

A RL Formalism **15**
A.1 Meta-RL Background 15

B More on Intrinsic Motivation **16**

C Expanded Related Works **17**
C.1 Novelty Search and Quality Diversity 18

D Results **18**
D.1 Meta-RL Environments 18
D.2 Tree Climb 19

E Combining with Other Approaches **21**

F Ablation Studies **22**
F.1 Threshold Conditioning 23
F.2 Constrained Information Flow 24
F.3 Ablation Study Results 24

G No Partial Credit **24**
G.1 Soft Filtering 26
G.2 Sensitivity to the Filter Temperature 26
G.3 Interaction with Return Capping 27
G.4 Other Approaches 27

H Notes on Risky Higher Returns **27**

I Notes on Uniform Sampling **28**

J First-Explore's Environments **28**

K Tree Climb Environment **30**
K.1 Environment Structure 30
K.2 Observations 30
K.3 Why Tree Climb Induces LEA 30
K.4 Configurations 31

L Licensed Code Use **31**

M Compute Usage **32**

N Tree Climb Training Curves **33**

O Tree Climb Algorithm Architectures **35**

P Hyperparameter Sweep Grids **39**

Q Fixed Hyperparameters **41**

R Best Hyperparameter Configurations **42**

S Heatmap Figures **49**

A RL FORMALISM

We formalise environments as partially observable Markov decision processes (POMDPs). Throughout, $\Delta(X)$ denotes the set of probability distributions over a set X. A POMDP is specified by a tuple $E = (\mathcal{S}, \mathcal{A}, p, p_0, \omega, r, \gamma, H)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, $p_0 \in \Delta(\mathcal{S})$ is the initial state distribution, $\omega : \mathcal{S} \to \Delta(\mathcal{O})$ is the observation function mapping states to distributions over an observation space $\mathcal{O}$, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1]$ is the discount factor, and H is the episode horizon (the maximum number of timesteps per episode). When ω is the identity ($\mathcal{O} = \mathcal{S}$), the POMDP reduces to a fully observable MDP.

At each timestep t, the agent receives an observation $o_t \in \mathcal{O}$, selects an action $a_t \in \mathcal{A}$, receives a scalar reward r_t, and the environment transitions to a new state $s_{t+1} \sim p(\cdot \mid s_t, a_t)$. A *reactive* policy $\pi : \mathcal{O} \to \Delta(\mathcal{A})$ maps single observations to action distributions. More generally, a *history-dependent* policy $\pi : (\mathcal{O} \times \mathcal{A} \times \mathbb{R})^* \times \mathcal{O} \to \Delta(\mathcal{A})$ conditions on the full sequence of past observations, actions, and rewards.

An *episode* $\tau = (o_0, a_0, r_0, \ldots, o_{H-1}, a_{H-1}, r_{H-1})$ is a sequence of interactions over H timesteps. The episode return is the total reward accumulated over the episode. All environments in this paper use undiscounted returns ($\gamma = 1$):

$$R_\tau = \sum_{t=0}^{H-1} r_t. \tag{1}$$

The standard RL objective is to find a policy π^* that maximises expected return:

$$\pi^* = \arg\max_\pi \ \mathbb{E}_{\tau\sim\pi}\big[R_\tau\big], \tag{2}$$

where the expectation is over environment transitions, the initial state, and the policy's action sampling.

Rising Tide departs from this objective. Rather than searching for a single return-maximising policy, it learns a parametrised family of models $\mathcal{M} = \{M_C : C \in [C_{\text{min}},\, v_{\text{max}}]\}$. Each $M_C : \mathcal{O} \to \Delta(\mathcal{A})$ approximates the base policy's action distribution conditioned on episodes whose return is at least C; after training, the model achieving the highest mean return is selected for deployment (Algorithm 1).

A.1 META-RL BACKGROUND

Meta-reinforcement learning (meta-RL) considers agents that must learn to solve new tasks drawn from a distribution of environments. Formally, a meta-RL problem is defined by a distribution $p(\mathcal{E})$ over POMDPs (or MDPs). Each sampled environment $E_i \sim p(\mathcal{E})$ shares the same action and observation spaces but may differ in its transition dynamics, reward function, or both. For example, in the Bandits with One Fixed Arm domain (Figure 6), each sampled environment assigns means to the bandit arms, with rewards drawn from arm-specific distributions.

In each *trial*, the agent interacts with a single sampled environment E_i over K consecutive episodes (each of horizon H). The agent's goal is to maximise the cumulative return across all K episodes, where τ_k denotes the trajectory of the k-th episode:

$$J = \mathbb{E}_{E_i\sim p(\mathcal{E})}\left[\sum_{k=1}^{K} R_{\tau_k}\right]. \tag{3}$$

This multi-episode structure distinguishes meta-RL from standard RL: early episodes must be used to gather information about E_i, which can then be exploited in later episodes. A purely exploitative strategy will generally perform poorly, since the optimal behaviour for a newly sampled environment is unknown at the start of each trial.

In-context learning. Modern meta-RL methods typically implement the agent as a recurrent or sequence model (e.g., an LSTM or Transformer) whose hidden state persists across episodes within a trial. The model receives the full interaction history—observations, actions, and rewards from all previous episodes—as input, and is trained end-to-end on the outer objective J via standard RL algorithms. This approach, introduced by Duan et al. (2016) and Wang et al. (2016) as RL^2, enables the agent to perform *in-context learning*: the model's weights encode a general learning algorithm, while the recurrent state implements task-specific adaptation at test time without any gradient updates.

Within-trial adaptation emerges as a consequence of optimising J: an agent that fails to explore early will lack the information needed to exploit later, reducing cumulative return. In principle, the training process should therefore incentivise exploration.

Why meta-RL is susceptible to LEA. In many meta-RL environments, the agent must execute information-gathering behaviours across entire episodes (e.g., visiting novel maze locations, pulling unfamiliar bandit arms) before any exploitation benefit is realised. Because these exploratory episodes produce low immediate returns, the RL training signal treats them as undesirable behaviour, reinforcing their avoidance. This dynamic is especially pronounced when:

1 **Exploration has negative expected single-episode return.** In the Ray Maze domain, for instance, visiting goal locations yields reward $+1$ or -1 with a 70% probability of the latter. An agent that explores will *on average* accumulate negative reward per episode, while an agent that stays in place accumulates zero.
2 **A safe default behaviour exists.** If the agent can achieve a non-negative return by doing nothing (or by repeating a known safe action), the RL objective actively penalises deviation from this default, even though deviation is necessary for long-run performance.

Rising Tide is not a meta-RL algorithm *per se*—it does not assume a distribution over environments or multi-episode trials. However, it can be applied to meta-RL settings by treating the full K-episode trial as a single extended interaction: the sequence model receives the concatenated history of observations, actions, and rewards across all K episodes as input (Section 3.1, Figure 3), and the return threshold C is set based on the cumulative trial return $R = \sum_k R_{\tau_k}$. The base policy π_{base} generates diverse trial-level behaviours (some exploratory, some exploitative), and Rising Tide's return-conditioned filtering distils the most effective combinations. This generality allows Rising Tide to perform strongly against methods designed specifically for exploration-hostile meta-RL (Table 1), while also applying unchanged to standard RL domains such as Tree Climb (Section 4) where no task distribution exists.

B More on Intrinsic Motivation

As Section 2 describes, IM methods can be distracted. For example, when exploring a maze, a TV playing white noise can permanently distract some approaches (Burda et al., 2019a), because such methods incentivise seeking unpredictable observations, and white noise is inherently unpredictable. Different algorithms avoid this specific issue, but often introduce related variants of the same problem. For example, if a method incentivises unpredictable rewards rather than unpredictable observations, meaningless noise in the reward signal becomes distracting.

Another issue is that novelty is a dynamic objective, one that decreases as an area becomes familiar. This changing objective can cause issues where the agent eventually avoids a familiar state (e.g., the entrance of a dungeon), despite the familiar state still providing access to valuable exploration (Ecoffet et al., 2019).

In contrast to these methods, Rising Tide is a new algorithm class. It is distinct from RL and does not inherit these particular failure modes.

C Expanded Related Works

Self-Imitation Learning (SIL) (Oh et al., 2018) augments actor–critic training with an imitation objective on past transitions whose realised return exceeds the current value estimate. Although SIL also reuses successful experience, it trains a single return-maximising policy and preferentially reinforces positive-advantage transitions. Rising Tide instead learns a threshold-conditioned family of behaviour distributions and uses directional bootstrapping from lower to higher thresholds, preserving lower-return exploratory behaviours rather than suppressing them once a reliable local strategy is learnt.

Decision Transformer (Chen et al., 2021) learns a return-conditioned policy through supervised action prediction from a fixed offline dataset. Rising Tide is instead a tabula rasa online method designed to address learnt exploration avoidance: it generates its own experience through interaction, uses threshold rather than return-to-go conditioning, and constrains information flow across return thresholds. There is also **Online Decision Transformer** (Zheng et al., 2022), which combines offline pretraining with online fine-tuning and therefore differs from the tabula rasa online setting considered here.

Go-Explore (Ecoffet et al., 2019) is an archive-based framework for exploration that remembers promising states and then resets the environment to explore further from them. It is highly effective at visiting novel states in predictable environments, such as dungeon-like settings whose layout remains fixed. However, because Go-Explore typically requires either hand-coded state representations or access to a pretrained foundation model, we leave it outside the scope of our comparison to strictly tabula rasa methods, such as Rising Tide and PPO.

Although we do not empirically evaluate Go-Explore, the LEA perspective may offer a useful way of understanding its effectiveness: by explicitly preserving and returning to promising states, Go-Explore can help avoid repeatedly losing access to valuable regions of the state space, a dynamic that LEA naturally induces in other methods. Nevertheless, its reliance on engineered or pretrained representations places it outside the class of methods considered here, which can learn exploration-relevant structure without assuming such representations.

Evolutionary Methods can be applied to the RL setting. These methods mimic natural evolution by having a population of agents, such as rule-based policies or neural networks, interact with the environment, and then creating the next generation by mutating higher-performing agents. Neuroevolutionary approaches such as NEAT (Stanley & Miikkulainen, 2002) are a prominent example, evolving both neural network weights and topologies. Although flexible and powerful, such methods typically make less direct use of within-episode observations and rewards than standard RL algorithms, and may therefore scale less efficiently in settings where dense interaction data can be exploited.

First-Explore (Norman & Clune, 2024) is designed to solve the problem of information acquisition being costly (e.g., buying a treasure map) and requiring follow-up (e.g., the skill of map reading). Their paper demonstrates that the combination of these two properties can stymie prior meta-RL methods. First-Explore solves this problem by splitting behaviour into two modes: pure exploration (information acquisition) and pure exploitation, with the exploration mode being trained solely to best-inform the subsequent phase of exploitation.

This mode splitting is highly applicable to meta-RL, as the setting gives agents multiple periods of interaction with an environment, with the environment being reset each time. The agent can acquire pertinent information in the initial interactions (e.g., locate a treasure) and then repeatedly use that information in the later interactions (e.g., repeatedly navigate to the found treasure). The method has not yet been adapted to the non-meta-RL setting.

Finally, only some tasks admit such a decomposition. For example, skill acquisition is itself a fundamental form of exploration, as in learning to walk. This does not naturally fit into First-Explore's framework, because some forms of learning cannot be cleanly partitioned into pure exploration followed by pure exploitation. Intuitively, acquiring information about a skill is not enough: improvement requires repeatedly practising the skill itself over time.

C.1 Novelty Search and Quality Diversity

Novelty Search (NS) is a family of approaches that forgo directly optimising for return, instead pursuing novelty. By seeking novelty, these algorithms aim to explore the space of all possible solutions. After the search is finished, the best behaviour found is used at inference time. In environments where the return signal misleads, these algorithms can outperform directly optimising for return.

Quality Diversity (QD) (Cully et al., 2015) is an approach to optimisation that seeks a diversity of performant solutions, rather than optimise for a single best approach. QD algorithms specify some measure of diversity (e.g., behaviour variation) and some measure of quality (e.g., episode return) and then aim to find the best behaviour possible for each segment of behaviour space. For example, given a six-legged robot, one can constrain how the legs are used, e.g., forbid using leg 1. Given such constraints, a QD algorithm can then aim to find the best behaviour for each constraint[g]. By developing a diverse population of solutions, for a given volume of total compute, these methods can often outperform optimising a single solution (e.g., achieving a better final return).

Rising Tide contains aspects of both NS and QD:

- Similarly to pure NS approaches, Rising Tide does not directly optimise for (expected) episode return, instead it explores a space of behaviours, with the best episode return being selected after training.
- Like QD methods, Rising Tide avoids myopically optimising a single approach:
  1 Rising Tide learns a parametrised family of models, with different models capturing different behaviours (e.g., low vs high return).
  2 Within each model, Rising Tide learns the associated distribution of behaviours.

However, Rising Tide is distinct from both approaches:

- Unlike NS methods, Rising Tide does not optimise for novelty.
- Unlike QD methods, Rising Tide makes no attempt to segment behaviour space (and optimise for the best performing behaviour in each segment)[h].

Finally, many NS and QD methods employ RL and Rising Tide could potentially supplant these methods' uses of RL. This possibility exemplifies how Rising Tide is orthogonal and potentially complementary to both approaches.

D Results

D.1 Meta-RL Environments

Despite both Rising Tide and First-Explore performing well on the challenging meta-RL environments studied, the two algorithms behave differently on the same environments, as Figure 7 illustrates. Tables 3, 4 and 6 report per-environment results with p-values against the best algorithm; with doubled training time on Dark Treasure Room, Rising Tide's lead over First-Explore grows (Table 5).

[g] Though many QD methods rely on a manually specified and segmented behaviour space, some Quality Diversity algorithms learn these features automatically.

[h] Implicitly, Rising Tide may segment behaviours based on return thresholds. However, this segmentation arises from Rising Tide approximating the correct behaviour distribution for each return threshold. In a loose sense, this dynamic is the inverse of QD attempting to maximise the return for each behaviour segment.

Table 3: Mean cumulative reward ± standard deviation of algorithms on the Bandits with One Fixed Arm meta-RL environment. p-values: two-sided exact Mann–Whitney U test against the best algorithm. *Complete rank separation: every run of the best algorithm outscored every run of the compared algorithm, giving the smallest p-value this test can attain.

Algorithm	Mean Reward	p-value to best
Rising Tide	$\mathbf{129.61 \pm 0.78}$	**best**
First-Explore	127.21 ± 1.62	< 0.001
RL^2	107.42 ± 11.35	$< 0.001^*$
UCB1	116.83 ± 0.55	$< 0.001^*$
TS	123.28 ± 2.52	$< 0.001^*$

Table 4: Mean cumulative reward ± standard deviation of algorithms on the Dark Treasure Room meta-RL environment. p-values: two-sided exact Mann–Whitney U test against the best algorithm. *Complete rank separation: every run of the best algorithm outscored every run of the compared algorithm, giving the smallest p-value this test can attain.

Algorithm	Mean Reward	p-value to best
Rising Tide	$\mathbf{2.41 \pm 0.25}$	**best**
First-Explore	2.37 ± 0.14	0.247
RL^2	0.10 ± 0.25	$< 0.001^*$
VariBAD	0.03 ± 0.56	$< 0.001^*$
HyperX	-0.91 ± 1.05	$< 0.001^*$

Table 5: Mean cumulative reward ± standard deviation of Rising Tide and First-Explore on the Dark Treasure Room meta-RL environment, when training time is doubled. p-value: two-sided exact Mann–Whitney U test against the best algorithm.

Algorithm	Mean Reward	p-value to best
Rising Tide	$\mathbf{2.71 \pm 0.20}$	**best**
First-Explore	2.53 ± 0.12	0.036

Table 6: Mean cumulative reward ± standard deviation of algorithms on the Ray Maze meta-RL environment. p-values: two-sided exact Mann–Whitney U test against the best algorithm. *Complete rank separation: every run of the best algorithm outscored every run of the compared algorithm, giving the smallest p-value this test can attain.

Algorithm	Mean Reward	p-value to best
Rising Tide	$\mathbf{0.49 \pm 0.01}$	**best**
First-Explore	0.40 ± 0.01	$< 0.001^*$
RL^2	0.00 ± 0.00	$< 0.001^*$
VariBAD	0.00 ± 0.00	$< 0.001^*$
HyperX	0.00 ± 0.00	$< 0.001^*$

D.2 TREE CLIMB

Tables 7, 9 and 10 break down the Tree Climb results of Table 2 per variant; Table 8 reports the backtracking-ablated variant.

Table 7: Results on the Combined Modifications Tree Climb $k = 30$ over ten training runs.

Algorithm	Jackpot rate[a]	Jackpot Path depth[a]	Return[a]	Best sweep return[b]
Rising Tide	**1.000**	**30.00**	**1026.00**	**1026.00**
DQN	0.000	1.00	30.00	30.00
PPO	0.000	1.00	30.00	30.00
PPO-E3B	0.000	1.50 ± 0.92	0.42 ± 0.31	30.00
PPO-ICM	0.000	1.00	0.70 ± 0.13	30.00
PPO-RND	0.000	1.00	30.00	30.00
UDRL	0.000	1.00	15.17 ± 14.83	30.00

[a] Evaluated at the hyperparameter setting that maximises Jackpot Path depth, favouring the baseline methods; none reaches the jackpot anywhere in the sweep. Jackpot rate and depth use the argmax policy; Return uses the stochastic policy (argmax for DQN, which has no stochastic mode). [b] Since successful learning can induce LEA, the depth-maximising setting need not maximise return. Best sweep return is the maximum return achieved in the initial random search, before the selected configurations were re-run over ten seeds.

Table 8: Results on the Combined Modifications Tree Climb $k = 30$ with backtracking ablated, averaged over ten training runs.

Algorithm	Jackpot rate[a]	Jackpot Path depth[a]	Return[a]	Best sweep return[b]
Rising Tide	**1.000**	**30.00**	**1026.00**	**1026.00**
DQN	**1.000**	**30.00**	**1026.00**	**1026.00**
PPO	0.000	1.00	29.90 ± 0.22	30.00
PPO-E3B	0.000	1.00	0.39 ± 0.46	30.00
PPO-ICM	0.000	1.10 ± 0.30	9.30 ± 13.55	30.00
PPO-RND	0.000	1.00	30.00	30.00
UDRL	0.000	1.00	6.11 ± 11.95	30.00

[a] Evaluated at the hyperparameter setting that maximises Jackpot Path depth, favouring the baseline methods. Jackpot rate and depth use the argmax policy; Return uses the stochastic policy (argmax for DQN, which has no stochastic mode). [b] Since successful learning can induce LEA, the depth-maximising setting need not maximise return. Best sweep return is the maximum return achieved in the initial random search, before the selected configurations were re-run over ten seeds.

Table 9: Results on Reward-Gated Jackpot Path Tree Climb $k = 30$ over ten training runs.

Algorithm	Jackpot rate[a]	Jackpot Path depth[a]	Return[a]	Best sweep return[b]
Rising Tide	**1.000**	**30.00**	**1026.00**	**1026.00**
DQN	0.000	1.00	30.00	30.00
PPO	0.000	1.00	30.00	30.00
PPO-E3B	0.000	1.00	0.62 ± 0.44	30.00
PPO-ICM	0.000	1.00	0.63 ± 0.07	30.00
PPO-RND	0.000	1.00	23.60 ± 11.86	30.00
UDRL	0.000	1.00	0.42 ± 0.41	30.00

[a] Evaluated at the hyperparameter setting that maximises Jackpot Path depth, favouring the baseline methods; none reaches the jackpot anywhere in the sweep. Jackpot rate and depth use the argmax policy; Return uses the stochastic policy (argmax for DQN, which has no stochastic mode). [b] Since successful learning can induce LEA, the depth-maximising setting need not maximise return. Best sweep return is the maximum return achieved in the initial random search, before the selected configurations were re-run over ten seeds.

Table 10: Results on Constant-Action Easy Path Tree Climb $k = 30$ over ten training runs.

Algorithm	Jackpot rate[a]	Jackpot Path depth[a]	Return[a]	Best sweep return[b]
Rising Tide	**1.000**	**30.00**	**1029.00**	**1029.00**
DQN	0.300 ± 0.458	9.70 ± 13.29	329.70 ± 457.80	1029.00
PPO	0.500 ± 0.500	15.50 ± 14.50	529.50 ± 499.50	30.00[c]
PPO-E3B	0.200 ± 0.400	8.30 ± 11.29	225.50 ± 401.84	1029.00
PPO-ICM	0.500 ± 0.500	15.50 ± 14.50	523.65 ± 505.45	1029.00
PPO-RND	0.800 ± 0.400	24.20 ± 11.60	829.20 ± 399.60	1029.00
UDRL	0.000	1.00	6.62 ± 11.70	30.00

[a] Evaluated at the hyperparameter setting that maximises Jackpot Path depth, favouring the baseline methods. Jackpot rate and depth use the argmax policy; Return uses the stochastic policy (argmax for DQN, which has no stochastic mode). [b] Since successful learning can induce LEA, the depth-maximising setting need not maximise return. Best sweep return is the maximum return achieved in the initial random search, before the selected configurations were re-run over ten seeds. [c] For PPO, no run in the initial random search reached the jackpot (best sweep return 30.00), but five of the ten re-runs of the selected configuration did, so its mean return exceeds its best sweep return.

E Combining with Other Approaches

Rising Tide learns a family of return-conditioned models from observed behaviour. This design can be combined with existing approaches. We outline five combinations, none yet evaluated empirically, as directions for future work. Table 11 gives an overview; details follow.

Table 11: Five ways to combine Rising Tide (RT) with RL. None have been evaluated empirically.

Combination	**Direction**	**What transfers**	**Benefit**	**Risk**
RT as RL initialisation	RT $\rightarrow$ RL	Policy ($M_{C_{\text{best}}}$)	RL starts from strong prior	Fine-tuning may reintroduce LEA
RT rollouts for offline RL	RT $\rightarrow$ RL	$\mathcal{M}$ rollouts	RL sees full return spectrum	Narrow per-model coverage
Parallel training	RT $\leftrightarrow$ RL	Shared rollouts	RL gets replay data immediately	Reverse direction breaks info flow
RL as RT base policy	RL $\rightarrow$ RT	Base policy	Richer rollouts for all M_C	LEA-affected RL avoids key regions
Modified rewards	—	Reward signal	Compatible with any reward modification	Non-stationary rewards shift C

Rising Tide as initialisation for RL. The best-performing model $M_{C_{\text{best}}}$ could initialise an RL policy, which then fine-tunes via policy gradient methods. Because Rising Tide already captures high-return behaviour, RL begins from a strong prior rather than from scratch. The risk is that RL fine-tuning may reintroduce LEA. Constraining the fine-tuned policy to stay close to the Rising Tide initialisation (e.g., via a KL penalty) could mitigate this.

Rising Tide rollouts as offline RL training set. After training, the model family $\mathcal{M}$ can generate rollouts at a range of conditioning levels. These rollouts form a dataset of return-diverse trajectories from competent policies, suitable for offline RL methods (e.g., CQL, Kumar et al., 2020; IQL, Kostrikov et al., 2022). Unlike policy initialisation, this gives RL access to behaviour across the full return spectrum—not just $M_{C_{\text{best}}}$. The risk is that the model family's rollouts may lack the state–action coverage that online exploration would provide, since each M_C produces relatively narrow behaviour around its conditioning level.

Parallel training on shared rollouts. Rising Tide and an off-policy RL method (e.g., DQN) could train simultaneously on rollouts generated by Rising Tide's base policy. The primary benefit flows from Rising Tide to RL: the base policy rollouts populate the RL replay buffer from the start of training. The reverse direction—feeding RL experience back into Rising Tide—is less clean. Using the RL policy as Rising Tide's base policy makes the training non-stationary, since the data-generating distribution shifts as RL improves. Training Rising Tide directly on RL-generated episodes is more concerning: those episodes come from a policy optimised for high return, leaking optimised behaviour into every conditioning level and breaking the one-directional information flow. After training, both

are available and the better-performing model can be deployed. More speculatively, a single network could be trained with both objectives, taking C as an additional input, though the interaction between training signals is unclear.

RL policy as Rising Tide's base policy. Rising Tide's base policy can be any policy, not just random. A pre-trained RL policy would generate higher-return rollouts, giving every model in the family $\mathcal{M}$ richer training data. Unlike the parallel training scenario above, training RL first and then running Rising Tide on the resulting policy avoids non-stationarity: the base policy is fixed throughout Rising Tide's training. The resulting $M_{C_{\text{best}}}$ may then exceed the RL policy's performance by extracting structure from its best trajectories that RL itself failed to consolidate. The risk is that an RL policy which has already succumbed to LEA would generate rollouts that systematically avoid high-return regions, providing a poor prior for Rising Tide.

Applying Rising Tide to modified reward signals. Rising Tide operates on episode returns and is agnostic to their source. Any technique that modifies the reward signal—intrinsic motivation (e.g., curiosity bonuses), diversity incentives, reward shaping—changes the return that Rising Tide conditions on. If the reward modification is stationary, no other changes are needed. If it is non-stationary (e.g., curiosity bonuses that decrease as states become familiar), the meaning of a given return level C shifts over training, and the return threshold sampling or $v_{\max}$ cap (Appendix I) may need to adapt as the return distribution shifts.

F Ablation Studies

To investigate how the components of Rising Tide's design contribute to the algorithm's performance and LEA resistance, we evaluate ablations of two core design elements: (1) threshold conditioning and (2) constrained information flow. The associated results show that each design element is essential, and that full LEA resistance emerges only when both are combined.

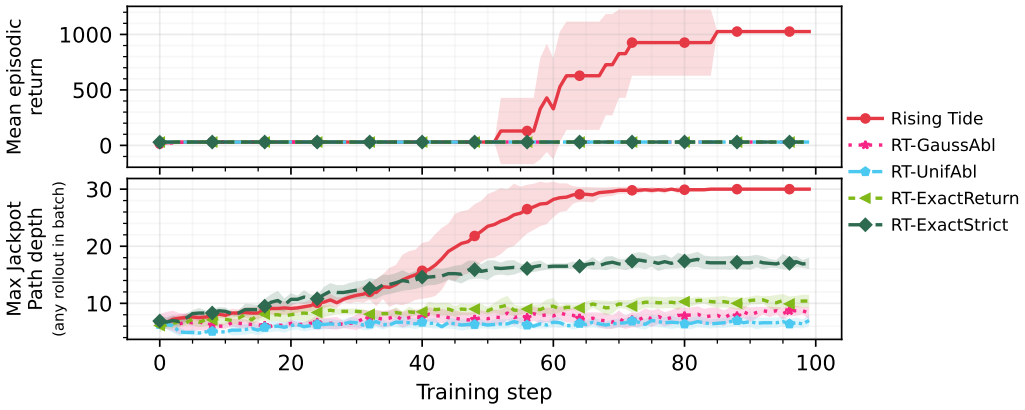


Figure 5: **Rising Tide ablations on the Combined Modifications Tree Climb environment.** **Rising Tide (red •)** reaches the end of the Jackpot Path (bottom) and achieves high return (top), whereas all four ablations succumb to LEA. The ablations remove threshold conditioning, one-way information flow, or both.

Integral to Rising Tide is a *rising tide* of information from models of lower return to models of higher return. Two design elements enable this tide effect.

1 **Threshold conditioning:** By modelling the trajectories whose return meets *or exceeds* a given return threshold, Rising Tide's models of a given return often generate trajectories that achieve more return than requested. These trajectories then serve as training data for models of higher return.
2 **Constrained information flow:** Rising Tide restricts information flow such that lower-threshold models are never trained on episodes generated by higher-threshold models.

The two design elements play complementary roles. Threshold conditioning helps lower-threshold models generate data to train higher-threshold models. The information-flow constraint preserves

diversity by blocking the following dynamic: because fewer behaviours meet high thresholds, higher-threshold models are more likely to collapse onto a narrow set of behaviours. Without the constraint, unsuccessful rollouts from these models could then dominate the training data for lower-threshold models and reduce their behavioural diversity. Preventing this dynamic allows Rising Tide to sustain exploration and resist LEA.

F.1 Threshold Conditioning

Prior return-conditioned methods, such as UDRL, RCP, and Decision Transformer (Chen et al., 2021)[i], model behaviour associated with achieving an exact return, e.g., behaviours with a return of exactly 5. We call this approach *exact-return modelling*. Rising Tide instead models behaviour that meets or exceeds a return threshold, e.g., behaviour achieving a return of 5 or more. We call this approach *threshold conditioning*.

Exact-return modelling may impede learning because a model conditioned on a low return is trained to suppress trajectories that achieve higher returns. In a deterministic environment, an idealised model that perfectly reproduces the exact-return behaviour distribution would *never* generate a return above the requested value, thereby limiting trial-and-error discovery of more rewarding behaviour.

Threshold conditioning avoids this problem because trajectories that exceed a threshold remain part of the target distribution. Lower-threshold models can therefore generate such trajectories and provide training data for higher-threshold models.

F.1.1 Constrained Exact-Return Modelling (RT-ExactStrict)

This ablation replaces threshold conditioning with exact-return modelling while preserving Rising Tide's information-flow constraint.

Background: In Rising Tide, an episode τ is generated using a rollout threshold C_{rollout}. A learning threshold C_{learning} is then sampled such that $C_{\text{learning}} \geq C_{\text{rollout}}$.

If the episode return R_τ meets or exceeds the learning threshold,

$$R_\tau \geq C_{\text{learning}} \geq C_{\text{rollout}}, \tag{4}$$

then $M_{C_{\text{learning}}}$ is updated using the episode. Otherwise, the episode is discarded.[j]

In contrast, exact-return modelling approaches learn as follows: for each generated episode τ, based on the achieved episode return R_τ, they update the model of that achieved return M_{R_τ} using the episode τ.

Modification: RT-ExactStrict follows Rising Tide's procedure, but sets $C_{\text{learning}} = R_\tau$, rather than sampling C_{learning} normally. This modification means that M_{R_τ} is updated using τ and results in exact-return modelling.

However, for RT-ExactStrict, we maintain Rising Tide's constrained-information-flow filtering criterion, that models never learn from trajectories generated by higher-return models. This criterion is not present in prior exact-return modelling approaches such as UDRL and RCP.

This constraint corresponds to Equation (4), which here reduces to

$$R_\tau \geq C_{\text{rollout}}.$$

Episodes satisfying this criterion are used to update M_{R_τ}; all others are discarded.

F.1.2 Basic Exact-Return Modelling (RT-ExactReturn)

RT-ExactReturn removes RT-ExactStrict's requirement that $R_\tau \geq C_{\text{rollout}}$. Every episode is used to update the model corresponding to its achieved return, M_{R_τ}, regardless of whether $R_\tau \geq C_{\text{rollout}}$. The ablation therefore removes both threshold conditioning and constrained information flow, making its training analogous to prior methods such as UDRL and RCP.

[i] Decision Transformer is an *offline* RL method. Although it does not address LEA or online exploration, it is relevant here as a return-conditioned method.

[j] All Tree Climb experiments use the hard filter, $\sigma = 0$, so $\xi_\tau = 0$; see Appendix G.1.

F.2 Constrained Information Flow

Rising Tide enforces one-way information flow by requiring

$$C_{\text{learning}} \geq C_{\text{rollout}}.$$

In our experiments, C_{learning} is obtained by adding an exponential random variable to C_{rollout}, which guarantees this inequality. We evaluate two ablations that replace this sampling rule with distributions that can produce $C_{\text{learning}} < C_{\text{rollout}}$.

F.2.1 Unconstrained Uniform Threshold Selection (RT-UnifAbl)

RT-UnifAbl samples C_{learning} uniformly between the return-threshold lower bound C_{min} and the running maximum return v_{max} (capped at c_{max}), independently of C_{rollout}. Consequently, there is a high probability that $C_{\text{learning}} < C_{\text{rollout}}$, violating the information-flow constraint.

F.2.2 Unconstrained Gaussian Threshold Selection (RT-GaussAbl)

RT-GaussAbl samples C_{learning} by adding a Gaussian random variable to C_{rollout}. Because the Gaussian random variable can be negative, there is a substantial probability that $C_{\text{learning}} < C_{\text{rollout}}$, violating the information-flow constraint.

F.3 Ablation Study Results

The ablations were evaluated on the Combined Modifications Tree Climb environment, the hardest variant of the LEA benchmark introduced in this paper. To ensure a fair comparison, they followed the same protocol used for Rising Tide and the baselines: first, a random hyperparameter sweep of size 50; then, each of the top three configurations was trained with 10 different seeds. The results for the best-performing configuration are reported here, averaged over 10 independent training runs.

All four ablations succumbed to LEA: none discovered the Jackpot in any of the 10 runs, whereas Rising Tide succeeded in every run (Figure 5 and Table 12). These results demonstrate that Rising Tide's design is integral to its performance. For each reported performance metric, every comparison between Rising Tide and an ablation exhibited complete rank separation, yielding $p = 1.08 \times 10^{-5}$ under a two-sided Mann–Whitney U test.

Table 12: **Rising Tide ablations evaluated on the Combined Modifications Tree Climb environment:** Rising Tide's design is integral to its LEA resistance, and ablating this design reduces the Jackpot rate from 100% to zero. Each metric is reported as the mean over 10 independent training runs, with the standard deviation shown where non-zero.

Algorithm	Jackpot rate	Jackpot Path depth	Return	Information flow	Return conditioning
Rising Tide	**1.000**	**30.00**	**1026.00**	low $\rightarrow$ high	$R \geq C$
RT-GaussAbl	0.000	8.30 ± 1.10	30.00	low $\leftrightarrow$ high	$R \geq C$
RT-UnifAbl	0.000	7.00 ± 1.00	30.00	low $\leftrightarrow$ high	$R \geq C$
RT-ExactReturn	0.000	10.40 ± 1.11	29.70 ± 0.64	low $\leftrightarrow$ high	$R = C$
RT-ExactStrict	0.000	17.00 ± 1.00	30.00	low $\rightarrow$ high	$R = C$

G No Partial Credit

At $\sigma = 0$, Rising Tide's filter includes an episode in model M_C's training set if and only if the episode return R meets or exceeds the conditioning level C. This is all-or-nothing: inclusion jumps from 0 to 1 at $R = C$, with no partial credit for performance near the boundary.

An episode with return $R = C - 0.01$ receives the same treatment as one with $R = C - 1000$: both are discarded. The filter does not reward near-misses. Symmetrically, an episode with $R = C$ receives the same treatment as one with $R = C + 1000$: both are fully included. The filter does not reward exceeding the threshold.

This lack of partial credit creates two problems:

1 **Below the threshold, model M_C is out-of-distribution.** Because every episode with $R < C$ is excluded—no matter how close to C—model M_C never observes behaviour from situations where C is unachievable. In variable environments, certain instances make high return impossible regardless of agent behaviour (e.g., a multi-armed bandit where all arm rewards are low). No episode from such an instance can pass a high threshold, so model M_C has no training data for these instances. When deployed on one, it may act erratically. The same applies within a single episode: if achieving C becomes impossible partway through—due to bad actions or bad luck—model M_C is equally out-of-distribution.
2 **Above the threshold, there is no incentive to exceed it.** If the episode return already satisfies $R \geq C$, additional return does not affect inclusion. Behaviours that greatly exceed the threshold are not preferentially reinforced over behaviours that barely meet it, so there is no training signal driving the model towards higher return once C is satisfied.

Both problems stem from the same cause: the hard filter gives no partial credit. Below C, better performance does not increase the chance of inclusion. Above C, better performance does not increase the chance of inclusion either. In neither regime does marginal improvement matter.

Which domains this affects. In Bandits with One Fixed Arm, the lack of partial credit is rarely an issue, because the reward dynamics create an incentive to do better however well the agent is currently doing: the Gaussian noise accumulated over the pulls means that any trial return remains possible, and the better the agent has performed, the likelier it is to pass the threshold. Similarly, in Dark Treasure Room a stochastic policy occasionally visits one or more new locations in the last episode of a trial, potentially collecting several rewards drawn uniformly from $[-4, 2]$, which again rewards increasing return at every level of performance.

Ray Maze's trial return distribution is very different. The agent must navigate a procedurally generated maze using 15 lidar-like observations to reach goals, and can reach at most one goal per episode. 30% of goals give $+1$ and 70% give -1, and a goal's value can only be determined by visiting it, so in the first episode of a trial visiting a goal has negative expected reward. Exploration and exploitation are therefore easy to identify: exploration *requires* visiting a new goal of uncertain value, while exploitation *requires* revisiting a goal identified as positive in an earlier episode of the trial. There are only nine possible trial returns, the integers from -4 to 4, and the optimal behaviour varies the number of explorations: if the goal found in the first episode is negative, the agent should explore once more; if it is positive, the agent should exploit.

The optimal behaviour's outcomes therefore fall into two groups: a return of 4 or 2 in the 51% of tasks where one of the first two goals visited is positive, and a minimised penalty in the 49% where both are negative. Without partial credit, Rising Tide cannot learn this behaviour. Consider the model M_2 conditioned on $C = 2$:

(i) M_2 trains only on trials with a return of at least 2.
(ii) From these trials, M_2 learns the right behaviour: visit a goal, and if it is negative, visit another.
(iii) After two negative goals, the best possible return is 0, from finding a positive third goal and visiting it twice, so no such trial meets the threshold.
(iv) M_2 therefore never trains on data from two negative goals.
(v) Yet in 49% of tasks, the right behaviour finds two negative goals. M_2 is then in a situation absent from its training data, where it may act erratically (problem 1).

Because of the hard filter, M_2 trains only on the behaviour's lucky outcomes, and never on its unlucky ones, and so learns only half the behaviour: how to act when things go well.

Tree Climb has deterministic returns, so the hard filter partitions behaviours by their true return and is exactly right; as such, every Tree Climb experiment in this paper uses $\sigma = 0$.

In environments such as this, where returns take few discrete values and the best behaviour's outcomes split into lucky and unlucky groups, soft filtering (Appendix G.1) supplies the partial credit that Bandits with One Fixed Arm and Dark Treasure Room provide naturally: a lucky and an unlucky outcome of the same behaviour both contribute to the same models, with weight increasing in return. Its effect on Ray Maze is shown in the sensitivity analysis of Table 14, which reports the failure under the hard filter and the response to σ.

G.1 Soft Filtering

Soft filtering restores partial credit by applying the threshold to a perturbed copy of each episode's return, drawn once after the episode ends:

$$R_{\text{soft}} = R + \xi_\tau, \quad \text{where } \xi_\tau \sim \mathcal{N}(0, \sigma^2). \tag{5}$$

The filter threshold is then applied to R_{soft} rather than to R; the rewards the agent observes during the episode, and the actions each model is trained to imitate, are unchanged. For $\sigma > 0$, this gives every episode a nonzero probability of passing any finite filter threshold. For return threshold C, the inclusion probability is

$$\begin{aligned} p_{\text{incl}}(R; C) &= \Pr_{\xi_\tau \sim \mathcal{N}(0,\sigma^2)} (R + \xi_\tau \geq C) \\ &= \Pr_{\xi_\tau \sim \mathcal{N}(0,\sigma^2)} (\xi_\tau \geq C - R) \\ &= 1 - \Phi\left(\frac{C - R}{\sigma}\right) \\ &= \Phi\left(\frac{R - C}{\sigma}\right), \end{aligned} \tag{6}$$

where Φ is the standard normal CDF. Differentiating with respect to R gives

$$\frac{\partial p_{\text{incl}}}{\partial R} = \frac{1}{\sigma}\phi\left(\frac{R - C}{\sigma}\right), \tag{7}$$

where ϕ is the standard normal PDF. Since $\sigma > 0$ and $\phi(x) > 0$ for all finite x,

$$\frac{\partial p_{\text{incl}}}{\partial R} > 0. \tag{8}$$

Thus, for any fixed return threshold C, higher unperturbed return strictly increases the probability of inclusion, enabling partial credit.

1 **Below the threshold:** with an appropriate temperature σ, an episode with R close to C has a meaningful probability of passing the soft filter. Model M_C therefore trains on some episodes from instances where C was not achieved, keeping it closer to in-distribution when deployed on such instances.

2 **Above the threshold:** the perturbation can be negative, so an episode with R barely above C may be excluded. Higher return provides insurance against this. The agent therefore retains an incentive to exceed the threshold, not merely meet it.

To illustrate, consider return threshold $C = 5$, temperature $\sigma = 1$, and an episode with unperturbed return $R = 3$. Under the hard filter, this episode is excluded from model M_5's training set. With soft filtering, inclusion requires $\xi_\tau \geq 2$, giving probability $\Pr(\xi_\tau \geq 2) \approx 2.3\%$. By contrast, an episode with $R = 4.5$ is included when $\xi_\tau \geq 0.5$, which occurs with probability $\Pr(\xi_\tau \geq 0.5) \approx 30.9\%$. Near-miss episodes can therefore pass the filter appreciably often, while higher-return episodes always pass more often. Because inclusion probability drops off rapidly with distance below C, the training set remains dominated by high-return episodes; only near-miss episodes contribute appreciably.

G.2 Sensitivity to the Filter Temperature

The temperature σ controls how much partial credit the filter provides, relative to the scale of returns in the environment. When $\sigma = 0$, filtering is hard and there is no partial credit: model M_C learns nothing about instances where the return threshold is unachievable. When σ is small relative to the standard deviation of the base policy's returns, near-threshold episodes occasionally pass and models stay close to the true threshold-conditioned distribution. When σ is large (several times this standard deviation), the distinction between model M_C for different values of C blurs, as all models end up training on similar data.

We conduct a sensitivity analysis on Dark Treasure Room and Ray Maze.

On Dark Treasure Room (Table 13), performance was relatively insensitive over $\sigma \in \{0, 1, 2, 4\}$, with mean return ranging from 2.338 to 2.459. At the substantially larger value $\sigma = 8$, performance

decreased and variability increased. Ray Maze was more sensitive to σ (Table 14). Under the hard filter ($\sigma = 0$) training failed, with mean best training return -0.574 ± 0.218, the failure predicted above for this domain. Performance was strongest over the intermediate range $\sigma \in \{1.33, 2, 3\}$, peaking at $\sigma = 2$, and declined at both smaller and larger values.

Table 13: Sensitivity analysis of the filter temperature σ on Dark Treasure Room: mean best training return across targets $\pm$ standard deviation.

σ	**Mean best training return across targets**
0	2.338 ± 0.238
1	2.459 ± 0.134
2	2.410 ± 0.093
4	2.437 ± 0.262
8	1.644 ± 0.685

Table 14: Sensitivity analysis of the filter temperature σ on Ray Maze: mean best training return across targets $\pm$ standard deviation.

σ	**Mean best training return across targets**
0	-0.574 ± 0.218
0.89	0.038 ± 0.038
1.33	0.413 ± 0.055
2	0.430 ± 0.022
3	0.346 ± 0.020
4.5	0.277 ± 0.023
6.75	0.054 ± 0.104

G.3 Interaction with Return Capping

Appendix I describes capping $v_{\max}$ to prevent outlier returns from dominating the sampling of C_{rollout}. The cap applies to *sampling* (which return thresholds are targeted), not to the *filter check* (which episodes pass). Consequently, R_{soft} can exceed $v_{\max}$. This is acceptable because the filter check only affects model M_C's training set composition, not the sampling of C values, which is where outlier returns cause instability.

G.4 Other Approaches

Soft filtering is a partial solution: the partial credit vanishes for $|R - C| \gg \sigma$, so the incentive disappears far from the threshold. We briefly sketch two further directions.

Dynamic Target Adjustment. Rather than fixing C before each episode, the return threshold could be adjusted dynamically as the episode unfolds, so that it is always both feasible and ambitious. At inference time this is straightforward: a learnt classifier could estimate, given the history so far, what return is still achievable, and lower C accordingly. This would restore full partial credit at deployment, ensuring model M_C is never conditioned on an unachievable target. Applying dynamic adjustment during training is more delicate, since modifying the target mid-episode changes which episodes enter each model's training set, potentially disrupting the one-directional bootstrapping structure that Rising Tide relies on. We leave this direction to future work.

RL Fine-Tuning. After training, the most performant model $M_{C_{\text{best}}}$ could be fine-tuned with RL to directly maximise return. Because RL optimises expected reward at every timestep, it provides a continuous gradient by construction, addressing both the out-of-distribution and no-incentive-to-exceed problems. The risk is that RL fine-tuning may reintroduce LEA.

H Notes on Risky Higher Returns

As described in Section 3, Rising Tide models the base policy's actions in those episodes whose return R is *greater than or equal to* a return threshold C, $R \geq C$. It could be that the only way to

achieve high reward is a risky behaviour that is actually bad in expectation (e.g., buying a lottery ticket). However, this dynamic is not an issue because Rising Tide learns a model for each threshold. Learning this family of models means that, after training, the reliable model with the highest return can be used for inference.

Further, Rising Tide models the *proportion* of behaviours and more reliable behaviours are likely a greater proportion of observed behaviour (e.g., if behaviour 1 has a 90% chance of meeting the return threshold C_{goal} vs behaviour 2 having only a 10% chance of meeting the threshold, then all else being equal, we expect a ratio of 9:1 between behaviour 1 and behaviour 2). However, if, for whatever reason, behaviour 1 is very unlikely in the base policy, then Rising Tide could learn to focus on behaviour 2.

I Notes on Uniform Sampling

As described in Section 3.1, the rollout thresholds C_{rollout} were selected uniformly in the interval $[C_{\text{min}}, v_{\text{max}}]$, where C_{min} was a fixed lower bound (a hyperparameter, Appendix Q), and v_{max} was the highest episode return yet encountered in training.

Why is it necessary to cap the training return thresholds in some environments? The issue is that, under uniform sampling, if one behaviour yields returns that are orders of magnitude larger than those of other behaviours (e.g., 100 versus 1), then uniformly sampling return thresholds (e.g., $C_{\text{rollout}} \sim U[0, 100]$) effectively focuses almost entirely on thresholds associated with that outlier behaviour. Once such a behaviour is discovered, the sudden shift in the sampling distribution can destabilise training and undermine the core logic of Rising Tide.

Future work could improve how these return thresholds are sampled to avoid this issue.

J First-Explore's Environments

Meta-RL Domain 1: Bandits with One Fixed Arm

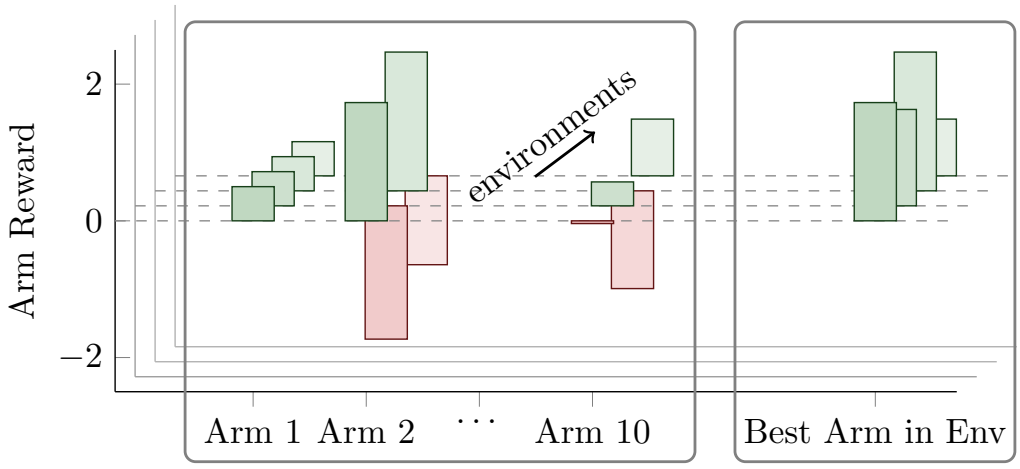


Figure 6: The Bandits with One Fixed Arm environment. Each environment has different arm means. Arm 1 guarantees a reward of 0.5 but is rarely the best arm. To maximise cumulative reward across 100 pulls, the agent must first explore to identify the best arm in the environment and then exploit it by resampling it.

Domain description: Each bandit (Figure 6) consists of ten arms. At each step, the agent selects one arm to pull, $a \in \{1, \ldots, 10\}$. The first arm always returns the fixed reward μ_1, while the remaining arms, $a \in \{2, \ldots, 10\}$, return an environment-specific arm mean $\mu_a \sim \mathcal{N}(0, 1)$ plus Gaussian noise ($\sigma_{\text{arm}} = 0.5$), making the task more challenging. The arm means are sampled once when the

environment is created and remain fixed throughout the episode, whereas the noise term is resampled independently on every arm pull. The objective is to maximise the expected cumulative reward obtained from a newly sampled bandit over a horizon of 100 pulls.

In our experiments, we apply this setup solely to the deceptive case where $\mu_1 = 0.5$ (the deceptive LEA-inducing setting).

Meta-RL Domain 2: Dark Treasure Room

The Dark Treasure Room domain is a 9×9 grid world domain, with 5 actions (up, right, down, left, stay still). Each environment has 8 treasure locations at uniformly random cells, with values drawn uniformly from $[-4, 2]$; each pays out at most once per episode. Each episode lasts 9 steps, in which the agent, observing only its own position, moves around the grid world to blindly encounter treasures or traps. The goal is to maximise cumulative reward across ten consecutive episodes in a newly sampled environment. This is challenging as on average (across environments in the domain), the mean reward of all locations is negative. The only way to obtain positive reward is to first explore in early episodes and find good treasure locations and then return to them in later episodes.

Meta-RL algorithms exhibit LEA on this domain, as it is easy to learn to stay still (and avoid the negative expected reward of visiting new locations). Once learnt, this strategy prevents exploring (and discovering a better strategy).

First-Explore and Rising Tide both address this form of LEA, but as Figure 7 illustrates, they do so in fundamentally different ways.

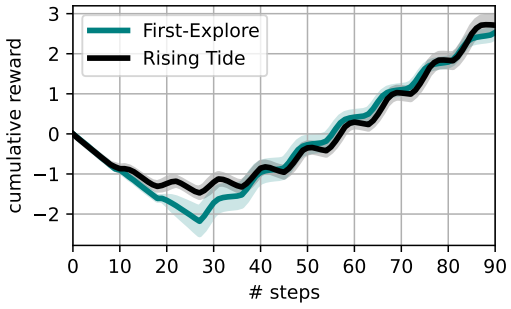


Figure 7: Mean cumulative reward against episode timestep of long training runs of Rising Tide (black) and First-Explore (teal) on the Dark Treasure Room environment. Each nine steps is a mini episode. When the cumulative reward decreases, the agent is exploring at the cost of reward. To increase cumulative reward requires using previous exploration. **Rising Tide learns to blend exploration and exploitation** (return increasing and decreasing each mini episode), while **First-Explore cannot blend the two**, and is restricted to choosing a time to switch from pure exploration (no attempt to get reward) to pure exploitation (no attempt to obtain information).

Meta-RL Domain 3: Ray Maze

The Ray Maze Domain is more complex. Each environment in the domain has a small random maze with three goal locations, and the agent must navigate using lidar observations of its surroundings. It can turn left, turn right, and move forward, and can see the location of walls and goal locations. However, only the first goal location the agent reaches in an episode gives a reward, of either $+1$ or -1, and 70% of goal locations are negative. The agent is trained to maximise cumulative reward over four 20-step episodes on newly sampled mazes. The agent must explore to find a good goal location, and then exploit it in subsequent episodes.

Like the prior domains, this domain induces LEA, with all algorithms other than Rising Tide and First-Explore learning to spin in place (and so never risk a negative goal location).

K TREE CLIMB ENVIRONMENT

This appendix provides a complete specification of the Tree Climb environment introduced in Section 4. Tree Climb is a deterministic, branching-tree environment with two rewarding paths; Section 4 motivates the design. The source code is available in the accompanying codebase.

K.1 ENVIRONMENT STRUCTURE

At each timestep the agent selects one of n *climb actions* ($n = 3$ in our experiments) or an optional *backtrack* action. Two paths are rewarding, each defined by a pre-specified sequence of climb actions. The agent's first action selects which path to attempt (action 1 for the Easy Path, action 2 for the Jackpot Path); any other action at the root immediately puts the agent off-sequence. Subsequent actions must match the chosen path's sequence to advance.

The backtrack action (action $n + 1$) undoes the previous step: depth decreases by one and the agent receives the negation of the reward it gained at that transition (so a climb–backtrack pair nets zero). Backtracking at the root is a no-op that consumes a timestep. By repeatedly backtracking, the agent can backtrack from any mistake — including an off-sequence state — all the way back to the root.

Formally, each path $i \in \{1, 2\}$ (path 1 the Easy Path, path 2 the Jackpot Path) is defined by:

- A **sequence** $s_i = (s_i^1, s_i^2, \ldots, s_i^{k-1})$ of climb actions, where k is the tree depth (as in the main text) (i.e. the number of depth transitions, including the path-selection step at depth 0) and each $s_i^t \in \{1, \ldots, n\}$.
- A **reward vector** $r_i = (r_i^0, r_i^1, \ldots, r_i^{k-1})$, where r_i^t is the reward received for advancing from depth t to depth $t + 1$ along path i. The initial reward r_i^0 is received upon selecting the path.

At depth $d = 0$, the correct action for path i is i itself (i.e., the first action determines the path); for $d \geq 1$ it is s_i^d. If the agent is *on-sequence* (all previous actions were correct) and selects the correct action, it advances to depth $d + 1$ and receives reward r_i^d.

An incorrect climb action puts the agent *off-sequence*: depth still increments (consuming a timestep) but the action yields zero reward. Once off-sequence, no further reward can be earned unless the agent backtracks to undo every incorrect action, returning to the depth where it first deviated.

Depth tracks the agent's position in the tree and can decrease via backtracking; the timestep counter always increments. Episodes terminate after k timesteps regardless of depth.

K.2 OBSERVATIONS

The Tree Climb observation pairs a path fingerprint in $[0, 1)^{kn}$ — a pseudorandom function of the agent's path through the tree — with a length-k one-hot encoding of the current timestep. Identical paths share a fingerprint while distinct paths receive uncorrelated ones; in particular, backtracking restores the fingerprint of the re-entered depth. Together with the timestep, the fingerprint fully determines the underlying state, so the environment is fully observable. The fingerprint carries no structural information about the path — not the actions taken, the current depth, or which paths share a prefix — so the agent must learn each path independently rather than generalise across similar paths.

K.3 WHY TREE CLIMB INDUCES LEA

One path is designed to be easier to find and exploit. Once an RL agent discovers it, the Easy Path's returns are maximised and the path is practised frequently, while the Jackpot Path, yielding zero reward for its initial steps and/or requiring a non-obvious action sequence, appears worse not because it *is* worse but because it is unpractised. The agent learns to avoid it (Section 1).

In configurations with backtracking, an agent that fails on the Jackpot Path can return to the root and switch to the Easy Path within the same episode, reinforcing avoidance. The Easy Path provides a useful reference point: if a baseline succeeds on the Easy Path but fails on the jackpot one, this suggests that the failure is specific to LEA rather than to a broader inability to solve the task.

K.4 CONFIGURATIONS

The two forms of difficulty described in Section 4 are controlled by the path sequences and reward vectors (Table 15):

Table 15: Tree Climb configurations. The Easy Path awards $+1$ per step in every configuration. The Jackpot Path sequence is a fixed random sequence in all configurations.

Config	Easy Path sequence	Jackpot Path rewards	Difficulty source
Reward-Gated Jackpot Path	Fixed random sequence	$0, 0, 0, 1, 1, \ldots, 1000$	Jackpot Path initially unrewarding
Constant-Action Easy Path	All action 1	$1, 1, \ldots, 1, 1000$	Easy Path trivial to learn
Combined Modifications	All action 1	$0, 0, 0, 1, 1, \ldots, 1000$	Both difficulties combined

Reward-Gated Jackpot Path. Both paths require a fixed random action sequence, but the Jackpot Path yields zero reward for the first three steps. An agent that discovers the Easy Path (which rewards immediately) has little incentive to persist through this unrewarding phase. The Jackpot Path terminates with a $+1000$ jackpot.

Constant-Action Easy Path. The Easy Path's sequence is all action 1, making it trivially discoverable. The Jackpot Path requires a specific random sequence. Both paths reward $+1$ per step, but only the Jackpot Path has a $+1000$ jackpot. The difficulty is purely navigational.

Combined Modifications. The Easy Path is trivial to learn (all action 1) *and* the Jackpot Path is initially unrewarding and requires a specific random sequence. This combines both sources of difficulty.

Combined Modifications, backtracking ablated. Identical to Combined Modifications, except that the backtrack action is replaced by an always-incorrect action: taking it puts the agent off-sequence, and since mistakes can no longer be undone, it precludes any subsequent reward. The action space is unchanged, so policies remain architecture-compatible. This variant isolates the role of backtracking in LEA (Section 5, Table 8). Despite there being strictly fewer rewarding action sequences, the ablated domain is easier, with DQN solving it reliably (Figure 10).

In all configurations, path sequences labelled "random" were generated once and held fixed across all experiments. All configurations were evaluated at tree depth $k = 30$.

At $k = 30$, the Easy Path yields a maximum return of 30. The Jackpot Path yields 1026 (Reward-Gated Jackpot Path / Combined Modifications) or 1029 (Constant-Action Easy Path): per-step reward of $+1$ (except the three zero-reward steps in Reward-Gated Jackpot Path / Combined Modifications) plus a $+1000$ jackpot at the final step. The large gap between Easy Path optimum (30) and Jackpot Path optimum (≥ 1000) makes success or failure on the Jackpot Path unambiguous.

L LICENSED CODE USE

The VariBAD (Zintgraf et al., 2020), HyperX (Zintgraf et al., 2021), and First-Explore (Norman & Clune, 2024) codebases were used under their MIT licences. The Craftax (Matthews et al., 2024) codebase was likewise used under its MIT licence to train the IM PPO variants. Our DQN and non-IM PPO baselines use the JAX implementation released alongside Lu et al. (2022), used under its Apache 2.0 licence with the upstream licence and copyright notice preserved; that implementation's DQN is itself based on CleanRL (Huang et al., 2022), which is MIT-licensed. The Rising Tide model's Flax GPT-2 module is a modified copy of the Hugging Face Transformers implementation (Wolf et al., 2019), used under its Apache 2.0 licence with the upstream copyright notice preserved.

Our own code is released under the MIT-HUMAN-AI-RESEARCH licence (see supplementary): the MIT licence with the following additional restriction.

> The Software, in whole or in part, and any modification of it, may not be used as training data for machine-learning models, nor included in any dataset used to train or fine-tune such models. For this purpose, any reproduction or derivative work of the Software, whether produced by a person or by an automated system,

is treated as the Software. This restriction does not limit running, modifying or building upon the Software, including using the Software itself to run the training of machine-learning models.

Our JAX implementation of UDRL is in part based on the existing PyTorch UDRL implementation (Dittert, 2020), which is also MIT-licensed. We release our JAX port under the same licence, retaining the upstream copyright notice. The original UDRL papers (Srivastava et al., 2019; Schmidhuber, 2019) are not accompanied by a reference codebase.

M Compute Usage

Table 16: Compute used to produce the core results in this paper. "Runs" counts independent training runs per (algorithm, environment) pair; "Wall" is the wall-clock budget per run. Each run used a single NVIDIA L40S GPU; the grand total is approximately 1.44×10^4 L40S-hours.

Experiment block	Envs	Algs	Runs	Wall (h)	L40S-hours
Meta-RL — standard training budget					
Bandits, base seeds [a]	1	3	5	50	750
Dark Treasure Room & Ray Maze, base seeds [b]	2	5	5	50	2,500
Extra seeds for Rising Tide & First-Explore [c]	3	2	5	50	1,500
Meta-RL — doubled training budget (Dark Treasure Room, Table 5)					
Rising Tide & First-Explore	1	2	10	100	2,000
Tree Climb — random search + top-3 re-evaluation					
All algorithms [d]	4	7	80	3	6,720
Rising Tide ablations [e]	1	4	80	3	960
Total					**14,430**

[a]Three trained algorithms on Bandits with One Fixed Arm: Rising Tide, First-Explore, and RL^2. UCB1 and Thompson Sampling are also reported in Table 3 but, being classical closed-form bandit algorithms, contribute negligible compute and are excluded from this accounting.
[b]Five algorithms per environment on Dark Treasure Room and Ray Maze: Rising Tide, First-Explore, RL^2, VariBAD, and HyperX.
[c]Rising Tide and First-Explore each received 5 additional seeds (10 total) per meta-RL environment to strengthen their statistical comparison.
[d]Per (algorithm, environment) pair, hyperparameters were chosen by a 50-configuration random search followed by re-evaluating the top 3 configurations over 10 seeds ($50 + 3 \times 10 = 80$ runs). Environments are the three Tree Climb variants (Reward-Gated Jackpot Path, Constant-Action Easy Path, Combined Modifications) plus the backtracking-ablated Combined Modifications variant. Algorithms are Rising Tide, DQN, PPO, PPO-RND, PPO-ICM, PPO-E3B, and UDRL.
[e]The four ablations of Appendix F (RT-GaussAbl, RT-UnifAbl, RT-ExactReturn, RT-ExactStrict) on Combined Modifications, under the same search protocol as footnote d.

The figures in Table 16 cover the core results of this paper. Because the work is novel and exploratory — introducing a new algorithm class (Rising Tide), a new failure mode (Learnt Exploration Avoidance), and a new benchmark (Tree Climb) — the compute consumed during the broader research process likely significantly exceeds the figure above. This includes iteration on the algorithmic design of Rising Tide, preliminary experiments used to isolate and characterise LEA, and the design and stress-testing of the Tree Climb environment and its variants before settling on the final configurations.

N TREE CLIMB TRAINING CURVES

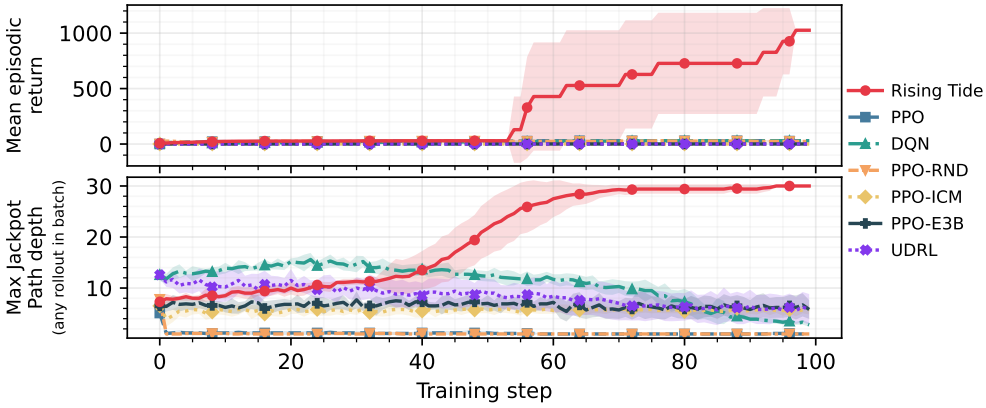


Figure 8: Mean episodic return (top) and max Jackpot Path depth (bottom) over training on *Reward-Gated Jackpot Path Tree Climb* $k = 30$. Lines are means and bands are ± 1 standard deviation across the ten seeds of each algorithm's selected configuration (the best of the three top random-search configurations after ten-seed re-evaluation; Appendix R). All algorithms report their training-time on-policy evaluation return (e.g., argmax for Q-learning, action sampling for PPO, best return threshold for Rising Tide).

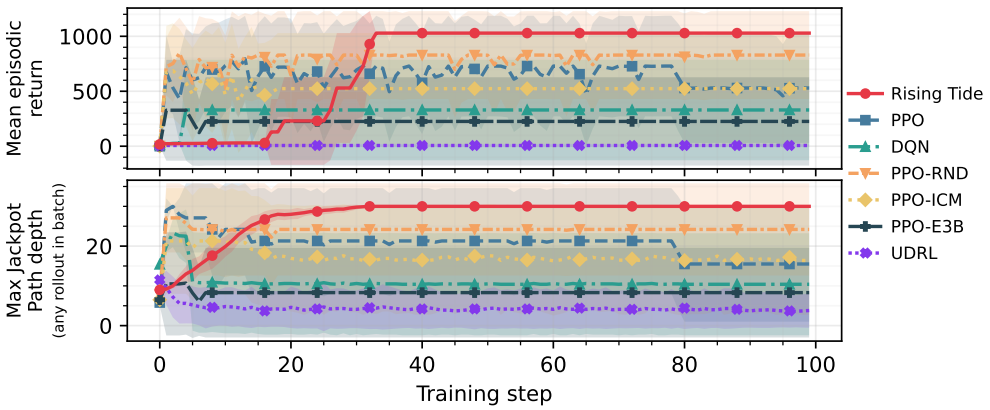


Figure 9: Mean episodic return (top) and max Jackpot Path depth (bottom) over training on *Constant-Action Easy Path Tree Climb* $k = 30$. Lines are means and bands are ± 1 standard deviation across the ten seeds of each algorithm's selected configuration (the best of the three top random-search configurations after ten-seed re-evaluation; Appendix R). All algorithms report their training-time on-policy evaluation return (e.g., argmax for Q-learning, action sampling for PPO, best return threshold for Rising Tide). Intriguingly, the baselines' partial performance on this environment seems based on a burst of early initial exploration; after this initial learning the mean returns and depths are mostly stationary (DQN, PPO-E3B, UDRL) or even decrease across training (PPO-ICM, PPO).

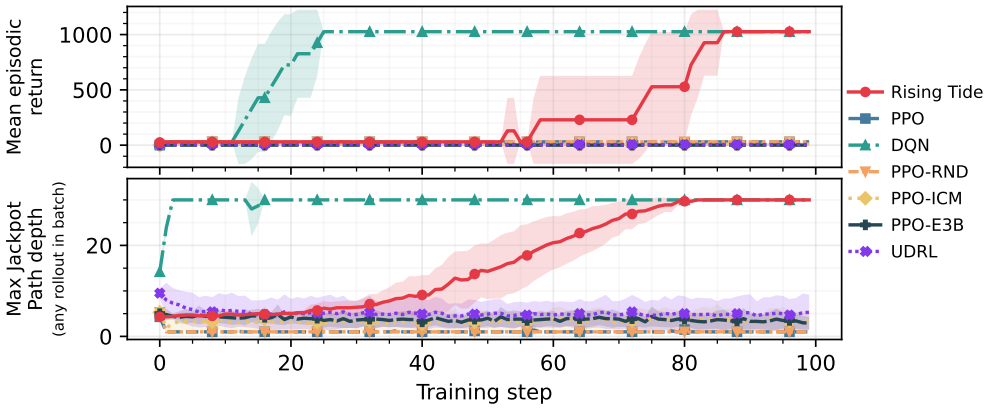


Figure 10: Mean episodic return (top) and max Jackpot Path depth (bottom) over training on *Combined Modifications Tree Climb* $k = 30$ *with backtracking ablated*. Lines are means and bands are ± 1 standard deviation across the ten seeds of each algorithm's selected configuration (the best of the three top random-search configurations after ten-seed re-evaluation; Appendix R). All algorithms report their training-time on-policy evaluation return (e.g., argmax for Q-learning, action sampling for PPO, best return threshold for Rising Tide). Notably, with backtracking ablated Q-learning (DQN) completely solves this environment, cleanly isolating the Learnt Exploration Avoidance (LEA) dynamic: when backtracking is available (the unablated environment), the agent can actively *avoid* exploration by reversing the accidental ε-greedy excursions down the Jackpot Path that would otherwise have enabled learning.

O TREE CLIMB ALGORITHM ARCHITECTURES

Table 17 lists each algorithm's parameter counts; the figures below illustrate the architectures.

Table 17: Parameter counts of each algorithm's policy (decision-making) network alongside its auxiliary networks (critic / value heads, plus intrinsic-reward modules where present). The intrinsic-reward modules' widths are swept (Appendix P); their counts use each algorithm's best configuration on Combined Modifications Tree Climb $k = 30$ (Appendix R), and PPO-RND's include its fixed random target network. Architectures are illustrated in the figures in this section.

Algorithm	Policy	Critic	Intrinsic	Total parameters
DQN	229,380	–	–	229,380
PPO	229,380	228,609	–	457,989
PPO-E3B	229,380	228,609	469,316	927,305
PPO-ICM	229,380	228,609	136,388	594,377
PPO-RND	229,380	457,218	113,536	800,134
UDRL	230,148	–	–	230,148
Rising Tide	229,892	–	–	229,892

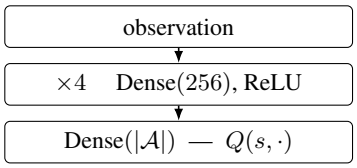


Figure 11: Network architecture of DQN. A single Q-network of four hidden Dense layers of width 256 with ReLU activations, mapping observations to $|\mathcal{A}|$ Q-values.

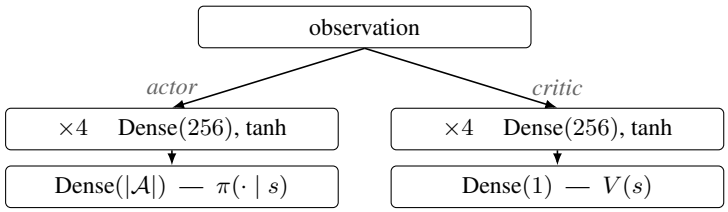


Figure 12: Network architecture of PPO. Parallel actor and critic MLPs, each four hidden Dense layers of width 256 with tanh activations, sharing only the input observation (no parameter sharing).

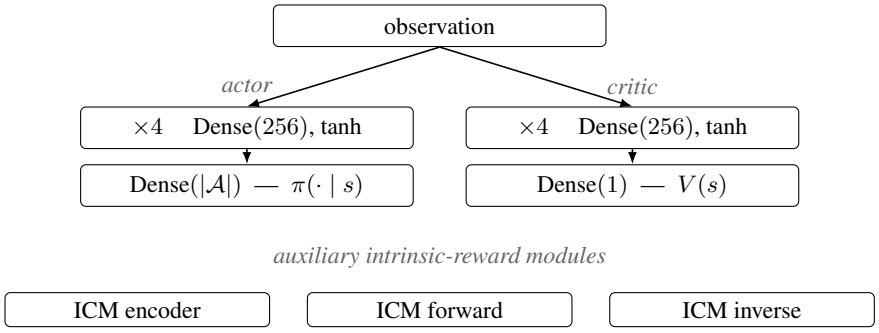


Figure 13: Network architecture of PPO-E3B. The actor–critic is identical to PPO. PPO-E3B additionally trains three ICM modules (encoder, forward model, inverse model). Each module is three Dense layers of width equal to the ICM hidden size with ReLU, followed by a final Dense output projection (to the ICM latent dimension for the encoder and forward model, to $|\mathcal{A}|$ for the inverse model); both are swept (Table 20), with selected values in Appendix R. PPO-E3B further maintains a per-environment inverse-covariance matrix in the encoder's latent space (the matrix itself adds no trainable parameters).

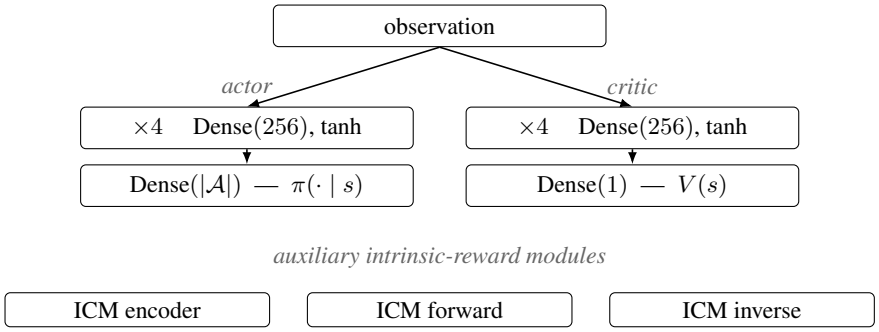


Figure 14: Network architecture of `PPO-ICM`. The actor–critic is identical to PPO. PPO-ICM additionally trains the three ICM modules shown (encoder, forward model, inverse model). Each module is three Dense layers of width equal to the ICM hidden size with ReLU, followed by a final Dense output projection (to the ICM latent dimension for the encoder and forward model, to $|\mathcal{A}|$ for the inverse model); both are swept (Table 21), with selected values in Appendix R. The forward-model error provides the intrinsic reward.

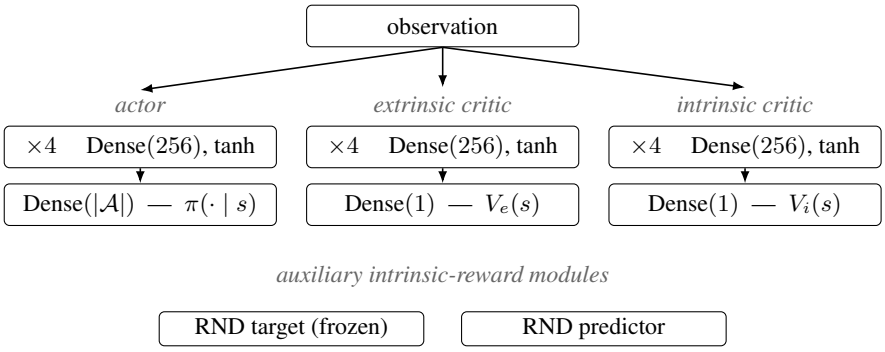


Figure 15: Network architecture of `PPO-RND`. Actor–critic with two value heads (extrinsic and intrinsic), each four hidden Dense layers of width 256 with tanh activations. Auxiliary RND modules: a randomly initialised frozen target network and a trainable predictor. Each is three Dense layers of width equal to the RND hidden size with ReLU, followed by a final Dense output projection to the RND output dimension (both swept, Table 22; selected values in Appendix R); the predictor is trained to match the target on observations actually visited.

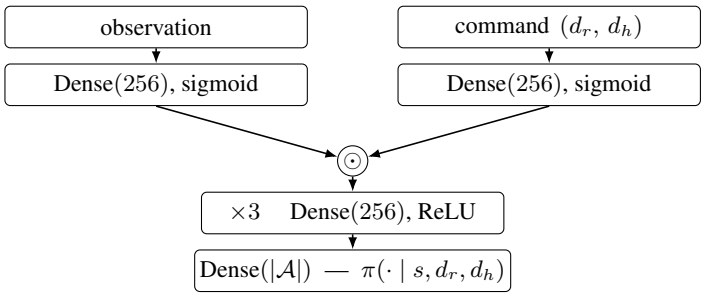


Figure 16: Network architecture of `UDRL`. Observation and command (desired return d_r and time horizon d_h) are independently embedded by a sigmoid Dense layer, then combined elementwise (denoted $\odot$). Three ReLU Dense layers and an action head produce the conditional policy.

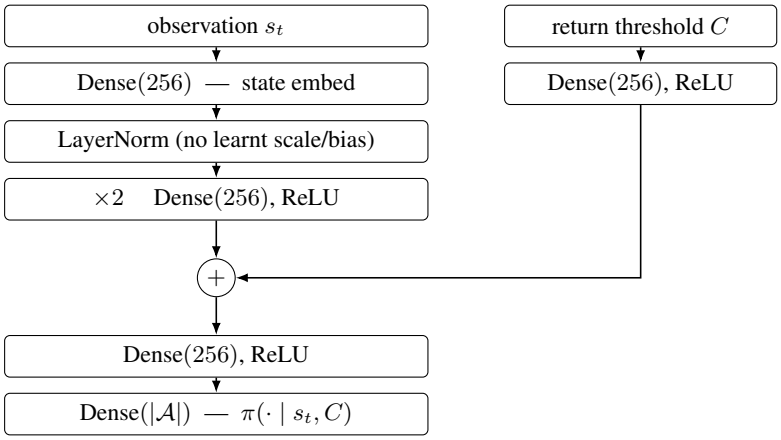


Figure 17: Network architecture of `Rising Tide`. State and return threshold are embedded independently by Dense(256) layers. The state embedding is normalised by an unscaled LayerNorm and passed through two Dense(256) layers with ReLU activations (used in place of a sequence model; alternative configurations replace these with a Transformer or LSTM). The ReLU-activated return-threshold embedding is added to this output, after which a final Dense(256) $\rightarrow$ ReLU $\rightarrow$ Dense$(|\mathcal{A}|)$ stack produces the categorical action distribution.

P HYPERPARAMETER SWEEP GRIDS

For each algorithm, we performed a random search over the value sets listed below. Tables in Appendix Q list the hyperparameters held constant across every configuration.

Table 18: Hyperparameter sweep grid for `DQN`. Learning starts and the replay sample batch size are constrained by the sampled buffer size: learning starts is set to the largest listed value $\leq \min(\text{buffer}, \max(10^4, \text{buffer}/2))$, and the batch size to the largest listed value $\leq \min(\text{buffer}, \max(32, \text{learning starts}))$. The ε anneal fraction is the fraction of the 3×10^9-step training budget over which ε is annealed.

Hyperparameter	Sweep values
Replay buffer size	{10000, 30000, 50000, 100000, 200000, 300000}
Replay sample batch size	{64, 128, 256, 512}
Training interval (steps)	{1, 2, 4, 8, 16, 32}
Learning starts (step)	{10000, 25000, 50000, 100000, 200000}
Final ε	{0.005, 0.01, 0.02, 0.05, 0.1}
ε anneal fraction	{0.02, 0.05, 0.1, 0.2, 0.5, 1}
Discount factor γ	{0.99, 0.995, 0.999, 1}
Learning rate	$\{3 \times 10^{-5}, 7 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 3 \times 10^{-4}\}$
Linear LR decay	{True, False}
Target network update interval	{500, 1000, 2000, 5000, 10000}

Table 19: Hyperparameter sweep grid for `PPO`.

Hyperparameter	Sweep values
Learning rate	$\{3 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}\}$
PPO clip ε	{0.05, 0.1, 0.2}
Entropy coefficient	{0, 0.01, 0.03, 0.1}
Discount factor γ	{0.99, 0.995, 0.999}
GAE λ	{0.9, 0.95, 0.97, 0.99}
Update epochs per rollout	{3, 5, 10}
Minibatches per epoch	{2, 4, 8}
Value-function coefficient	{0.25, 0.5, 1}
Linear LR decay	{True, False}

Table 20: Hyperparameter sweep grid for `PPO-E3B`.

Hyperparameter	Sweep values
Learning rate	$\{3 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}\}$
PPO clip ε	{0.05, 0.1, 0.2}
Entropy coefficient	{0, 0.01, 0.03, 0.1}
Discount factor γ	{0.99, 0.995, 0.999}
GAE λ	{0.8, 0.9, 0.95, 0.97, 0.99}
Update epochs per rollout	{3, 5, 10}
Minibatches per epoch	{2, 4, 8}
Value-function coefficient	{0.25, 0.5, 1}
Linear LR decay	{True, False}
ICM/E3B/RND update epochs	{1, 2, 4, 8}
ICM learning rate	$\{1 \times 10^{-4}, 3 \times 10^{-4}, 0.001, 0.003\}$
ICM forward-loss weight	{0.1, 0.3, 1, 3}
ICM inverse-loss weight	{0.1, 0.3, 1, 3}
ICM hidden size	{128, 256, 512}
ICM latent dimension	{8, 16, 32}
E3B reward coefficient	{0.03, 0.1, 0.3, 1, 3, 10, 30, 100}
E3B inverse-covariance ridge λ	{0.01, 0.03, 0.1, 0.3, 1, 2}

Table 21: Hyperparameter sweep grid for `PPO-ICM`.

Hyperparameter	Sweep values
Learning rate	$\{3 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}\}$
PPO clip ε	$\{0.05, 0.1, 0.2\}$
Entropy coefficient	$\{0, 0.01, 0.03, 0.1\}$
Discount factor γ	$\{0.99, 0.995, 0.999\}$
GAE λ	$\{0.8, 0.9, 0.95, 0.97, 0.99\}$
Update epochs per rollout	$\{3, 5, 10\}$
Minibatches per epoch	$\{2, 4, 8\}$
Value-function coefficient	$\{0.25, 0.5, 1\}$
Linear LR decay	{True, False}
ICM/E3B/RND update epochs	$\{1, 2, 4, 8\}$
ICM reward coefficient	$\{0.01, 0.03, 0.1, 0.3, 1, 3, 10\}$
ICM learning rate	$\{1 \times 10^{-4}, 3 \times 10^{-4}, 0.001, 0.003\}$
ICM forward-loss weight	$\{0.1, 0.3, 1, 3\}$
ICM inverse-loss weight	$\{0.1, 0.3, 1, 3\}$
ICM hidden size	$\{128, 256, 512\}$
ICM latent dimension	$\{8, 16, 32\}$

Table 22: Hyperparameter sweep grid for `PPO-RND`.

Hyperparameter	Sweep values
Learning rate	$\{3 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}\}$
PPO clip ε	$\{0.05, 0.1, 0.2\}$
Entropy coefficient	$\{0, 0.01, 0.03, 0.1\}$
Discount factor γ	$\{0.99, 0.995, 0.999\}$
GAE λ	$\{0.8, 0.9, 0.95, 0.97, 0.99\}$
Update epochs per rollout	$\{3, 5, 10\}$
Minibatches per epoch	$\{2, 4, 8\}$
Value-function coefficient	$\{0.25, 0.5, 1\}$
Linear LR decay	{True, False}
ICM/E3B/RND update epochs	$\{1, 2, 4, 8\}$
RND hidden size	$\{64, 128, 256, 512\}$
RND output dimension	$\{64, 128, 256, 512, 1024\}$
RND learning rate	$\{1 \times 10^{-5}, 3 \times 10^{-5}, 1 \times 10^{-4}, 3 \times 10^{-4}, 0.001, 0.003\}$
RND reward coefficient	$\{0.05, 0.1, 0.2, 0.5, 1, 2, 5\}$
RND loss coefficient	$\{1 \times 10^{-4}, 3 \times 10^{-4}, 0.001, 0.003, 0.01, 0.03, 0.1\}$
RND GAE coefficient	$\{0, 0.001, 0.003, 0.01, 0.03, 0.1\}$
RND episodic intrinsic reward	{True, False}

Table 23: Hyperparameter sweep grid for `UDRL`.

Hyperparameter	Sweep values
Top-N episodes for command sampling	$\{25, 50, 75, 100, 1000\}$
Update minibatch size	$\{256, 512, 768, 1024, 1536, 2048\}$
Horizon scale (command)	$\{0.01, 0.015, 0.02, 0.025, 0.03\}$
Return scale (command)	$\{0.01, 0.015, 0.02, 0.025, 0.03\}$
Learning rate	list(np.logspace(-4, -2, num=101)) (101 values)
Replay buffer size	$\{1000, 2000, 4000, 10000, 20000\}$

Table 24: Hyperparameter sweep grid for `Rising Tide`.

Hyperparameter	Sweep values
Entropy regularisation weight	$\{0.003, 0.01, 0.03, 0.1\}$
Minimum learning-threshold gap δ	$\{0.2, 0.5, 1, 1.5, 2, 3, 4\}$
Exponential gap mean θ	$\{0.01, 0.03, 0.1, 0.3, 1, 3, 10\}$
Prior-action rate ϵ	$\{0.025, 0.0\dot{3}, 0.05, 0.0\dot{6}\}$
Unconditioned-rollout return threshold u	$\{0, 1, 2, 3, 4\}$
Learning rate	$\{1 \times 10^{-5}, 3 \times 10^{-5}, 1 \times 10^{-4}\}$

Q FIXED HYPERPARAMETERS

The hyperparameters below were held constant across every sweep configuration and every algorithm–environment pair. Together with the swept values from Appendix P, they fully specify each training run.

Table 25: Fixed (non-swept) hyperparameters for `DQN`.

Hyperparameter	Value
Initial ε	1
Number of parallel environments	512
Target soft-update rate τ	1
Hidden size	256
Activation function	ReLU

Table 26: Fixed (non-swept) hyperparameters for `PPO`.

Hyperparameter	Value
Activation function	tanh
Max gradient norm	0.5
Number of parallel environments	512
Hidden size	256

Table 27: Fixed (non-swept) hyperparameters for `PPO-E3B`.

Hyperparameter	Value
Activation function	tanh
Hidden size	256
Max gradient norm	0.5
Number of parallel environments	1024

Table 28: Fixed (non-swept) hyperparameters for `PPO-ICM`.

Hyperparameter	Value
Activation function	tanh
Hidden size	256
Max gradient norm	0.5
Number of parallel environments	1024

Table 29: Fixed (non-swept) hyperparameters for `PPO-RND`.

Hyperparameter	Value
Activation function	tanh
Hidden size	256
Max gradient norm	0.5
Number of parallel environments	1024

Table 30: Fixed (non-swept) hyperparameters for `UDRL`.

Hyperparameter	Value
Hidden size	256
Rollout episodes per iteration	1000
Gradient updates per iteration	10
Activation function	ReLU (hidden) + sigmoid (input/command gate)

Table 31: Fixed (non-swept) hyperparameters for `Rising Tide`.

Hyperparameter	Value
Batch size	1000
Number of dense layers	2
Embed actions	False
Embed rewards	False
Hidden size	256
Learning thresholds per rollout	10
Return-threshold lower bound $C_{\min}$	$-(\delta+1)$, δ the minimum learning-threshold gap
Rollout-parameter sync interval	1,000 training steps (batched rollouts)
Sequence model	None (2 ReLU dense layers)
Activation function	ReLU
Return-threshold cap $c_{\max}$	40
Filter temperature σ	0 (hard filter)
Evaluation return thresholds	40 integers, 0–39
Optimiser	AdamW, weight decay 10^{-4}
Gradient clipping	element-wise, ± 0.25
Learning-rate schedule	linear warm-up over the first 1% of updates, then constant

R Best Hyperparameter Configurations

For each (algorithm, environment) pair we report the swept-key values of the configuration selected by the two-stage procedure: of the top three configurations from the random search over the grids of Appendix P, the one with the best ten-seed evaluation. **These tables show only the values of the swept hyperparameters; the fixed hyperparameters listed in Appendix Q apply to every configuration reported here.** Reproducing a row therefore requires both the row's swept values *and* the corresponding algorithm's fixed hyperparameter table. The best configurations for the backtracking-ablated Combined Modifications variant (Table 8) and for the Rising Tide ablations (Table 12) are listed in the supplementary material.

Table 32: Best hyperparameter configuration for `DQN` on Combined Modifications Tree Climb $k = 30$.

Hyperparameter	Value
Replay buffer size	200000
Replay sample batch size	512
Training interval (steps)	16
Learning starts (step)	100000
Final ε	0.1
ε anneal fraction	1
Discount factor γ	0.995
Learning rate	1×10^{-4}
Linear LR decay	True
Target network update interval	5000

Table 33: Best hyperparameter configuration for `PPO` on Combined Modifications Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	2×10^{-4}
PPO clip ε	0.1
Entropy coefficient	0
Discount factor γ	0.995
GAE λ	0.99
Update epochs per rollout	5
Minibatches per epoch	4
Value-function coefficient	0.5
Linear LR decay	False

Table 34: Best hyperparameter configuration for `PPO-E3B` on Combined Modifications Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	5×10^{-5}
PPO clip ε	0.05
Entropy coefficient	0.1
Discount factor γ	0.995
GAE λ	0.9
Update epochs per rollout	3
Minibatches per epoch	4
Value-function coefficient	1
Linear LR decay	False
ICM/E3B/RND update epochs	4
ICM learning rate	0.003
ICM forward-loss weight	1
ICM inverse-loss weight	0.3
ICM hidden size	256
ICM latent dimension	32
E3B reward coefficient	0.1
E3B inverse-covariance ridge λ	0.1

Table 35: Best hyperparameter configuration for `PPO-ICM` on Combined Modifications Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	3×10^{-5}
PPO clip ε	0.1
Entropy coefficient	0.03
Discount factor γ	0.99
GAE λ	0.8
Update epochs per rollout	3
Minibatches per epoch	8
Value-function coefficient	1
Linear LR decay	False
ICM/E3B/RND update epochs	4
ICM reward coefficient	10
ICM learning rate	1×10^{-4}
ICM forward-loss weight	1
ICM inverse-loss weight	3
ICM hidden size	128
ICM latent dimension	32

Table 36: Best hyperparameter configuration for `PPO-RND` on Combined Modifications Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	5×10^{-5}
PPO clip ε	0.1
Entropy coefficient	0.1
Discount factor γ	0.99
GAE λ	0.9
Update epochs per rollout	3
Minibatches per epoch	4
Value-function coefficient	1
Linear LR decay	True
ICM/E3B/RND update epochs	4
RND hidden size	128
RND output dimension	64
RND learning rate	1×10^{-4}
RND reward coefficient	2
RND loss coefficient	0.1
RND GAE coefficient	0
RND episodic intrinsic reward	False

Table 37: Best hyperparameter configuration for `UDRL` on Combined Modifications Tree Climb $k = 30$.

Hyperparameter	Value
Top-N episodes for command sampling	25
Update minibatch size	768
Horizon scale (command)	0.025
Return scale (command)	0.01
Learning rate	6.61×10^{-4}
Replay buffer size	1000

Table 38: Best hyperparameter configuration for `Rising Tide` on Combined Modifications Tree Climb $k = 30$.

Hyperparameter	Value
Entropy regularisation weight	0.01
Minimum learning-threshold gap δ	1
Exponential gap mean θ	3
Prior-action rate ϵ	$0.0\dot{3}$
Unconditioned-rollout return threshold u	1
Learning rate	1×10^{-4}

Table 39: Best hyperparameter configuration for `DQN` on Reward-Gated Jackpot Path Tree Climb $k = 30$.

Hyperparameter	Value
Replay buffer size	30000
Replay sample batch size	512
Training interval (steps)	4
Learning starts (step)	10000
Final ε	0.05
ε anneal fraction	1
Discount factor γ	0.99
Learning rate	3×10^{-4}
Linear LR decay	True
Target network update interval	5000

Table 40: Best hyperparameter configuration for PPO on Reward-Gated Jackpot Path Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	1×10^{-4}
PPO clip ε	0.1
Entropy coefficient	0.1
Discount factor γ	0.995
GAE λ	0.99
Update epochs per rollout	3
Minibatches per epoch	4
Value-function coefficient	0.25
Linear LR decay	False

Table 41: Best hyperparameter configuration for PPO-E3B on Reward-Gated Jackpot Path Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	5×10^{-5}
PPO clip ε	0.05
Entropy coefficient	0.1
Discount factor γ	0.995
GAE λ	0.9
Update epochs per rollout	3
Minibatches per epoch	4
Value-function coefficient	1
Linear LR decay	False
ICM/E3B/RND update epochs	4
ICM learning rate	0.003
ICM forward-loss weight	1
ICM inverse-loss weight	0.3
ICM hidden size	256
ICM latent dimension	32
E3B reward coefficient	0.1
E3B inverse-covariance ridge λ	0.1

Table 42: Best hyperparameter configuration for PPO-ICM on Reward-Gated Jackpot Path Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	2×10^{-4}
PPO clip ε	0.1
Entropy coefficient	0.01
Discount factor γ	0.99
GAE λ	0.97
Update epochs per rollout	10
Minibatches per epoch	8
Value-function coefficient	0.5
Linear LR decay	False
ICM/E3B/RND update epochs	1
ICM reward coefficient	10
ICM learning rate	3×10^{-4}
ICM forward-loss weight	0.1
ICM inverse-loss weight	3
ICM hidden size	128
ICM latent dimension	8

Table 43: Best hyperparameter configuration for `PPO-RND` on Reward-Gated Jackpot Path Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	2×10^{-4}
PPO clip ε	0.1
Entropy coefficient	0.03
Discount factor γ	0.99
GAE λ	0.97
Update epochs per rollout	3
Minibatches per epoch	4
Value-function coefficient	0.5
Linear LR decay	False
ICM/E3B/RND update epochs	2
RND hidden size	64
RND output dimension	1024
RND learning rate	3×10^{-5}
RND reward coefficient	0.5
RND loss coefficient	0.001
RND GAE coefficient	0.001
RND episodic intrinsic reward	False

Table 44: Best hyperparameter configuration for `UDRL` on Reward-Gated Jackpot Path Tree Climb $k = 30$.

Hyperparameter	Value
Top-N episodes for command sampling	50
Update minibatch size	768
Horizon scale (command)	0.01
Return scale (command)	0.025
Learning rate	1.66×10^{-3}
Replay buffer size	1000

Table 45: Best hyperparameter configuration for `Rising Tide` on Reward-Gated Jackpot Path Tree Climb $k = 30$.

Hyperparameter	Value
Entropy regularisation weight	0.03
Minimum learning-threshold gap δ	2
Exponential gap mean θ	1
Prior-action rate ϵ	$0.0\dot{3}$
Unconditioned-rollout return threshold u	2
Learning rate	1×10^{-4}

Table 46: Best hyperparameter configuration for `DQN` on Constant-Action Easy Path Tree Climb $k = 30$.

Hyperparameter	Value
Replay buffer size	30000
Replay sample batch size	512
Training interval (steps)	8
Learning starts (step)	10000
Final ε	0.01
ε anneal fraction	0.05
Discount factor γ	1
Learning rate	3×10^{-4}
Linear LR decay	True
Target network update interval	2000

Table 47: Best hyperparameter configuration for PPO on Constant-Action Easy Path Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	1×10^{-4}
PPO clip ε	0.05
Entropy coefficient	0.1
Discount factor γ	0.999
GAE λ	0.9
Update epochs per rollout	3
Minibatches per epoch	8
Value-function coefficient	0.25
Linear LR decay	False

Table 48: Best hyperparameter configuration for PPO-E3B on Constant-Action Easy Path Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	3×10^{-5}
PPO clip ε	0.1
Entropy coefficient	0
Discount factor γ	0.999
GAE λ	0.99
Update epochs per rollout	3
Minibatches per epoch	8
Value-function coefficient	1
Linear LR decay	True
ICM/E3B/RND update epochs	1
ICM learning rate	1×10^{-4}
ICM forward-loss weight	0.1
ICM inverse-loss weight	1
ICM hidden size	512
ICM latent dimension	8
E3B reward coefficient	0.03
E3B inverse-covariance ridge λ	0.03

Table 49: Best hyperparameter configuration for PPO-ICM on Constant-Action Easy Path Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	3×10^{-5}
PPO clip ε	0.05
Entropy coefficient	0
Discount factor γ	0.995
GAE λ	0.9
Update epochs per rollout	10
Minibatches per epoch	8
Value-function coefficient	0.5
Linear LR decay	False
ICM/E3B/RND update epochs	1
ICM reward coefficient	1
ICM learning rate	3×10^{-4}
ICM forward-loss weight	3
ICM inverse-loss weight	3
ICM hidden size	512
ICM latent dimension	32

Table 50: Best hyperparameter configuration for `PPO-RND` on Constant-Action Easy Path Tree Climb $k = 30$.

Hyperparameter	Value
Learning rate	1×10^{-4}
PPO clip ε	0.05
Entropy coefficient	0.03
Discount factor γ	0.999
GAE λ	0.9
Update epochs per rollout	10
Minibatches per epoch	4
Value-function coefficient	0.25
Linear LR decay	True
ICM/E3B/RND update epochs	8
RND hidden size	128
RND output dimension	128
RND learning rate	0.001
RND reward coefficient	1
RND loss coefficient	0.1
RND GAE coefficient	0.03
RND episodic intrinsic reward	True

Table 51: Best hyperparameter configuration for `UDRL` on Constant-Action Easy Path Tree Climb $k = 30$.

Hyperparameter	Value
Top-N episodes for command sampling	25
Update minibatch size	768
Horizon scale (command)	0.025
Return scale (command)	0.01
Learning rate	6.61×10^{-4}
Replay buffer size	1000

Table 52: Best hyperparameter configuration for `Rising Tide` on Constant-Action Easy Path Tree Climb $k = 30$.

Hyperparameter	Value
Entropy regularisation weight	0.003
Minimum learning-threshold gap δ	4
Exponential gap mean θ	0.3
Prior-action rate ϵ	0.025
Unconditioned-rollout return threshold u	2
Learning rate	1×10^{-4}

S HEATMAP FIGURES

This appendix collects the heatmaps that most directly show the learning dynamics behind the Tree Climb results: how the baseline methods learn to avoid the Jackpot Path, and how Rising Tide's family of return-conditioned models develops instead. All plots are for the Combined Modifications Tree Climb environment ($k = 30$) or—for Figure 22—its backtracking-ablated variant. Unless its caption says otherwise, each plot shows a single example seed of the best top-3 sweep configuration; mean-across-seeds versions of every plot, together with the corresponding plots—programmatically generated and captioned—for the other Tree Climb variants and algorithms, are provided in the supplementary material.

Reading the heatmaps. In each action-distribution plot (Figures 18 and 20 to 22), rows index training time (snapshot index) and columns index depth along the Easy Path (upper row of panels) or the Jackpot Path (lower row). Each cell is the on-sequence observation at that depth and training time, i.e., the state reached by following the path correctly up to that depth. Actions are grouped into the *correct* action (the forward action that continues the path), the *reverse* action (backtrack or, in the backtracking-ablated variant, the always-incorrect action that replaces it), and the remaining *incorrect* actions; the three panels in a row show the probability mass the policy assigns to each group. DQN acts greedily, so in its single-seed plot every cell is 0 or 1. Colours use a power-scaled mapping ($\gamma = 0.3$) that affects only the display, not the underlying probabilities. Vertical structure within a panel therefore reflects change over training, and horizontal structure reflects change with depth.

PPO learns the Easy Path and makes no sustained progress on the Jackpot Path. In Figure 18, PPO rapidly assigns high probability to the correct action along essentially the whole Easy Path and maintains it throughout training, with little mass on reverse or incorrect actions. The Jackpot Path shows a different pattern. There is no progressively advancing region of high correct-action probability; instead, high correct-action probability appears only at particular depths, as vertical bands that persist across training. These bands plausibly reflect behaviour learnt for the Easy Path, at depths where the two paths happen to require the same action, rather than a learnt sequence of Jackpot Path actions. At the other depths, substantial mass remains on reverse or incorrect actions.

PPO's value function (Figure 19) shows how this dynamic is written into its training signal. States near the start of the Easy Path receive comparatively high values, which decrease with depth as less Easy-Path reward remains, whereas the early states of the Jackpot Path are valued below the corresponding Easy-Path states. Because entering the Jackpot Path is assigned low value, updates that increase expected return reduce the probability of the actions that enter or continue along it. Fewer visits then leave fewer opportunities to refine the later Jackpot Path actions, or to associate the initial ones with the large return available on completion, and the policy increasingly favours the already-learnt Easy Path before the alternative has been developed.

DQN learns parts of the Jackpot Path that it then cannot reach. DQN (Figure 20) behaves differently from PPO. It develops localised regions of correct-action behaviour on the Jackpot Path: once sufficiently far along it, the greedy policy selects the correct forward action over a run of subsequent depths, forming an island of learnt Jackpot Path behaviour. Learning the correct actions at those deeper states does not, however, make them reachable. At earlier Jackpot Path depths DQN selects the reverse (backtrack) action, so the policy abandons the path before reaching the states at which it already behaves correctly. Useful behaviour exists within the learnt policy but is inaccessible, because the policy has learnt not to execute the exploratory actions required to reach it. The mean over ten seeds (Figure 21) shows the same structure. Replacing the backtrack action with an always-incorrect one (Figure 22) removes this avoidance route, and DQN then learns the full Jackpot Path (Table 8).

These plots show that LEA need not arise because a policy is incapable of representing the useful behaviour, or because that behaviour is never encountered. A policy can learn parts of a high-return behaviour while simultaneously learning earlier behaviour that prevents those parts from being reached and refined. Learning useful behaviour at some states is not enough when training also teaches the policy to avoid the sequence of actions that leads to them.

Rising Tide develops different behaviours at different return thresholds. Figures 23 to 28 show Rising Tide's family of return-conditioned models on the same environment. Each figure is a return-threshold sweep with one panel per return threshold C; within a panel, the layout is the same

as above, and the cells give the probability of the correct (Figures 23 and 24), reverse (Figures 25 and 26), or incorrect (Figures 27 and 28) action group along the Easy Path and the Jackpot Path.

Behaviour differs across return thresholds at the same stage of training: the panels do not converge together to a single action distribution. Models at high return thresholds progressively develop high correct-action probability along the Easy Path and the Jackpot Path, while models at other return thresholds retain different distributions, so different forms and stages of behaviour coexist within the family during training. In particular, correct behaviour along parts of the Jackpot Path develops at intermediate return thresholds before reliable correct behaviour across the whole path is established at the highest return thresholds (Figure 24); higher-return behaviour emerges progressively across the family rather than only once a single policy has learnt the complete behaviour. The probability of reversing (Figures 25 and 26) likewise varies with both the return threshold and the stage of training, rather than being governed by one action distribution shared across all performance levels. These sweeps illustrate the mechanism described in Section 3 and tested in Appendix F: lower-threshold models keep exploring and are not overwritten, while higher-threshold models refine the best behaviour found so far.

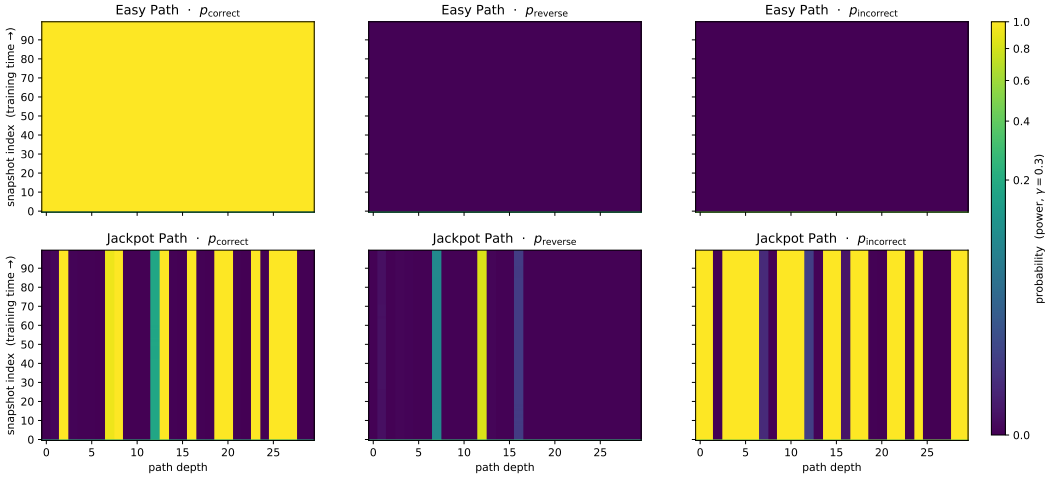


Figure 18: **Action distribution: PPO on Combined Modifications Tree Climb $k = 30$.** Each cell shows the policy's probability mass on the correct (forward), reverse (backtrack), or incorrect (remaining) action group at the on-sequence observation for that (training step, depth) cell on the Easy Path and Jackpot Path. Rows index training steps; columns index depth along the path. *An example seed of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*

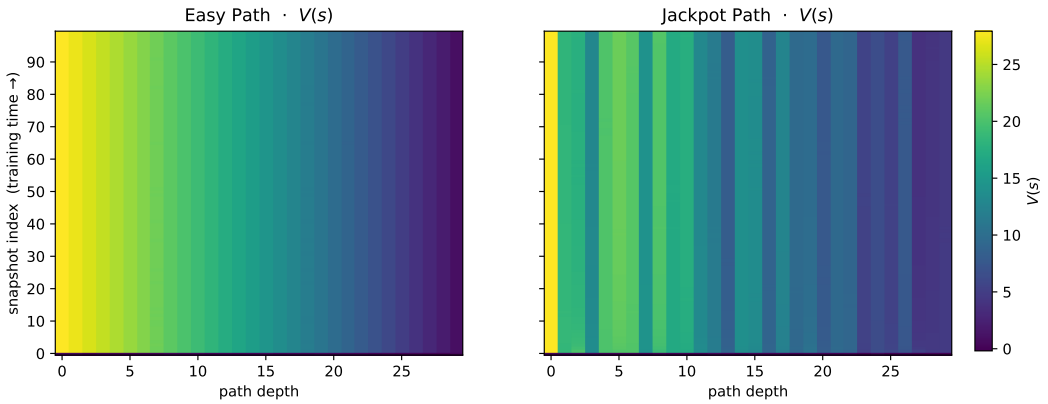


Figure 19: **Value function: PPO on Combined Modifications Tree Climb $k = 30$.** Each cell shows the estimated $V(s)$ (or per-action $Q(s, a)$ for DQN) at the on-sequence observation for that (training step, depth) cell on the Easy Path and Jackpot Path. Rows index training steps; columns index depth along the path. *An example seed of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*

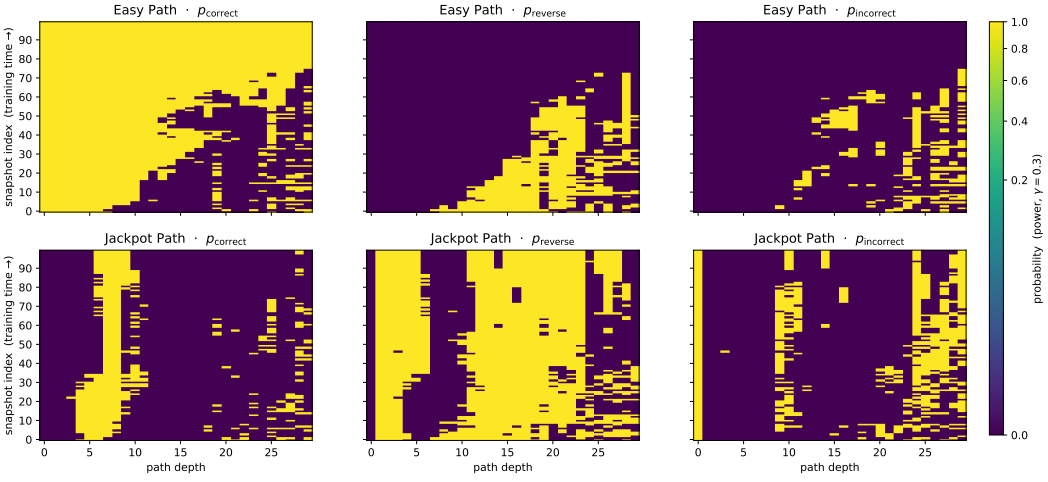


Figure 20: **Action distribution: DQN on Combined Modifications Tree Climb $k = 30$.** Each cell shows the policy's probability mass on the correct (forward), reverse (backtrack), or incorrect (remaining) action group at the on-sequence observation for that (training step, depth) cell on the Easy Path and Jackpot Path. Rows index training steps; columns index depth along the path. *An example seed of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*

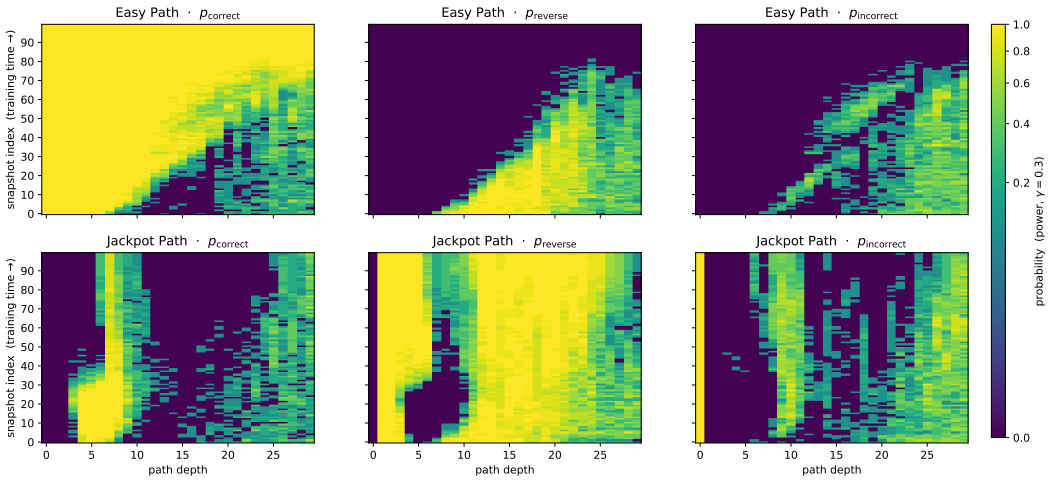


Figure 21: **Action distribution: DQN on Combined Modifications Tree Climb $k = 30$.** Each cell shows the policy's probability mass on the correct (forward), reverse (backtrack), or incorrect (remaining) action group at the on-sequence observation for that (training step, depth) cell on the Easy Path and Jackpot Path. Rows index training steps; columns index depth along the path. *The mean across 10 seeds of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*

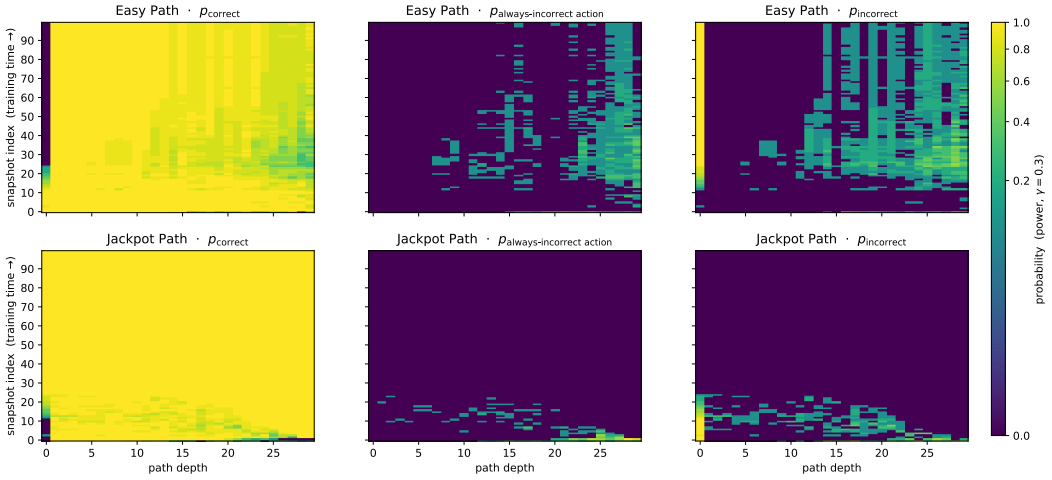


Figure 22: **Action distribution: DQN on Combined Modifications Tree Climb $k = 30$ with backtracking ablated.** Each cell shows the policy's probability mass on the correct (forward), always-incorrect action (which replaces backtracking in this variant and precludes any subsequent reward), or incorrect (remaining) action group at the on-sequence observation for that (training step, depth) cell on the Easy Path and Jackpot Path. Rows index training steps; columns index depth along the path. *The mean across 10 seeds of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*

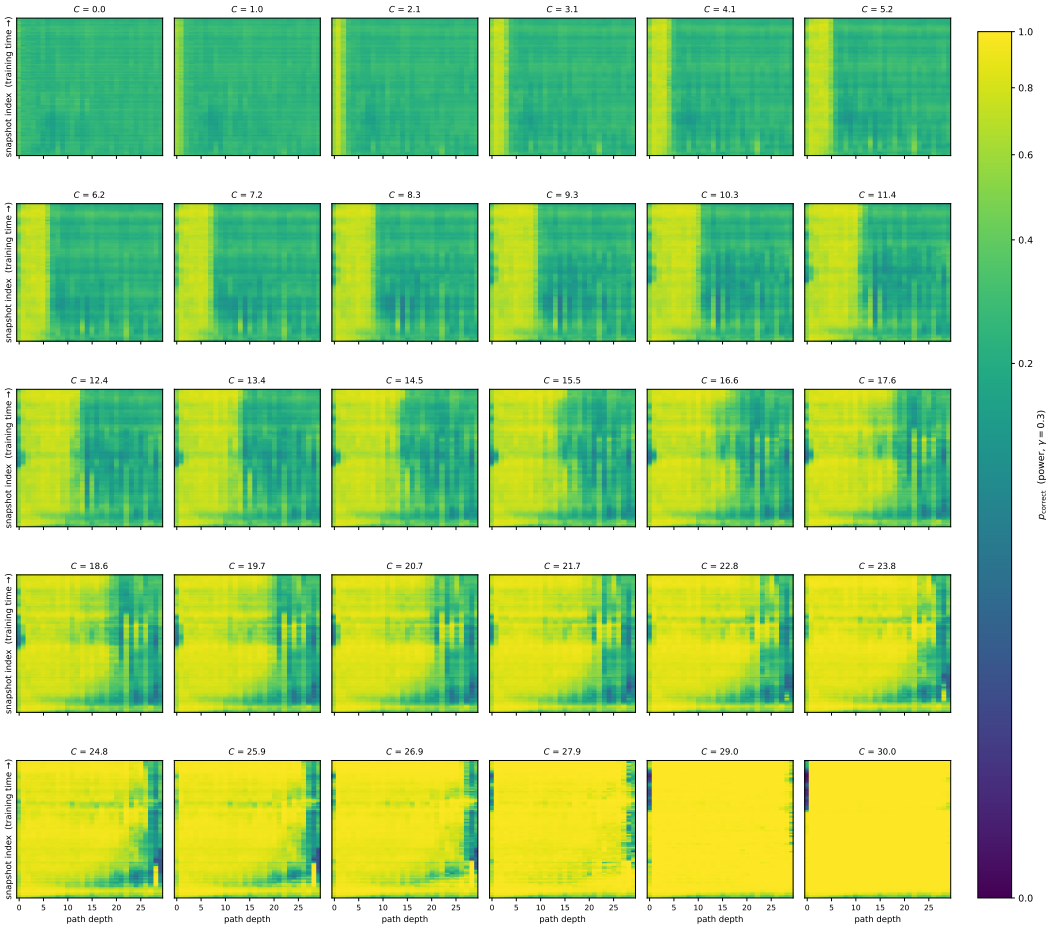


Figure 23: $P(\textbf{correct})$ **on the Easy Path — Rising Tide on Combined Modifications Tree Climb** $k = 30$**.** Return-threshold sweep: one panel per return threshold. Within each panel, each cell shows the policy's $P(\text{correct})$ when conditioned on that threshold, evaluated at the on-sequence observation for that (training step, depth) cell. Rows index training steps; columns index depth along the path. *An example seed of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*

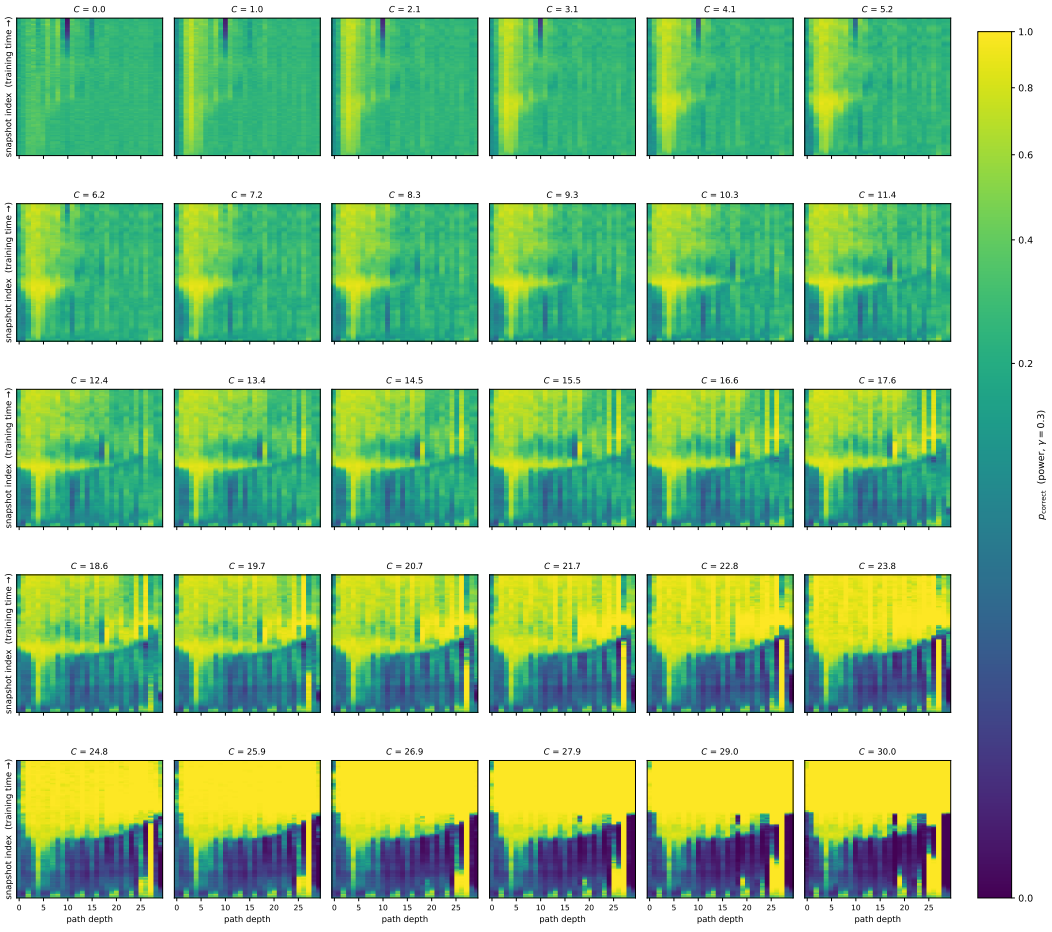


Figure 24: $P(\textbf{correct})$ **on the Jackpot Path — Rising Tide on Combined Modifications Tree Climb** $k = 30$**.** Return-threshold sweep: one panel per return threshold. Within each panel, each cell shows the policy's $P(\text{correct})$ when conditioned on that threshold, evaluated at the on-sequence observation for that (training step, depth) cell. Rows index training steps; columns index depth along the path. *An example seed of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*

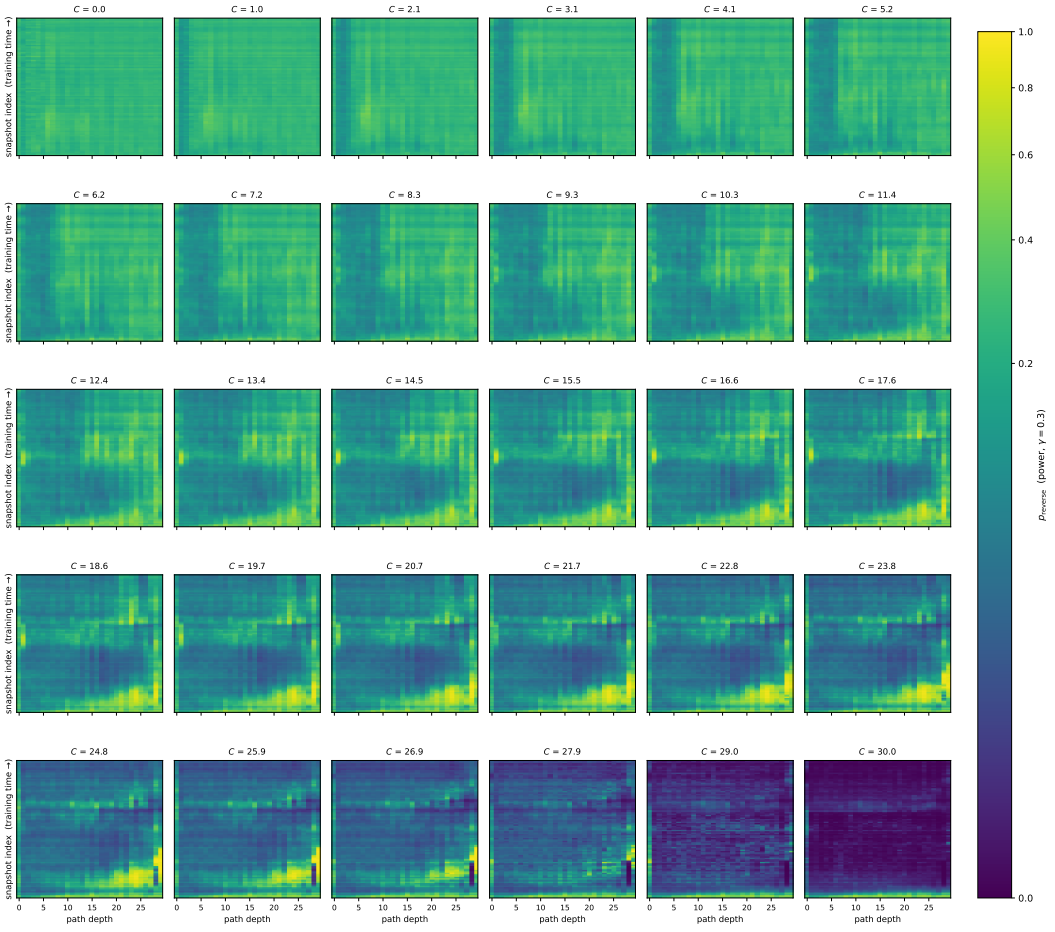


Figure 25: $P(\textbf{reverse})$ **on the Easy Path — Rising Tide on Combined Modifications Tree Climb** $k = 30$**.** Return-threshold sweep: one panel per return threshold. Within each panel, each cell shows the policy's $P(\text{reverse})$ when conditioned on that threshold, evaluated at the on-sequence observation for that (training step, depth) cell. Rows index training steps; columns index depth along the path. *An example seed of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*

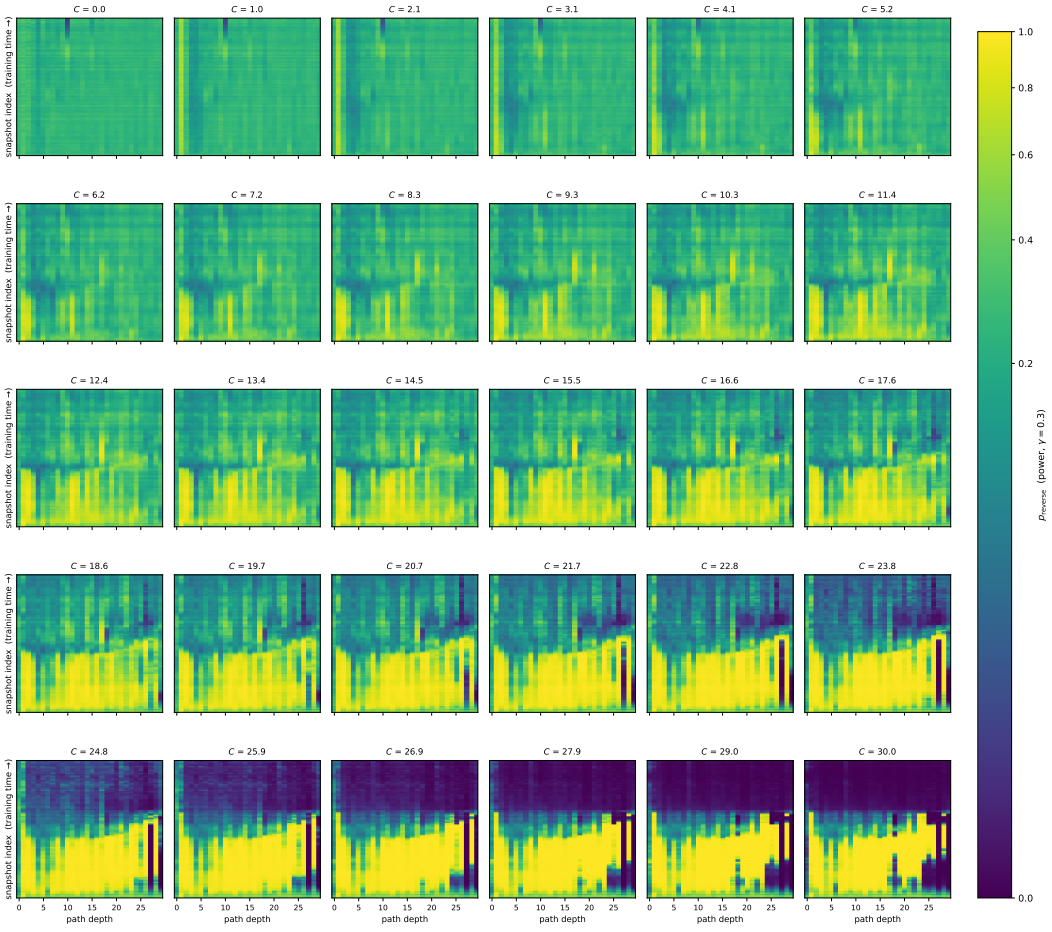


Figure 26: $P(\textbf{reverse})$ **on the Jackpot Path — Rising Tide on Combined Modifications Tree Climb** $k = 30$**.** Return-threshold sweep: one panel per return threshold. Within each panel, each cell shows the policy's $P(\text{reverse})$ when conditioned on that threshold, evaluated at the on-sequence observation for that (training step, depth) cell. Rows index training steps; columns index depth along the path. *An example seed of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*

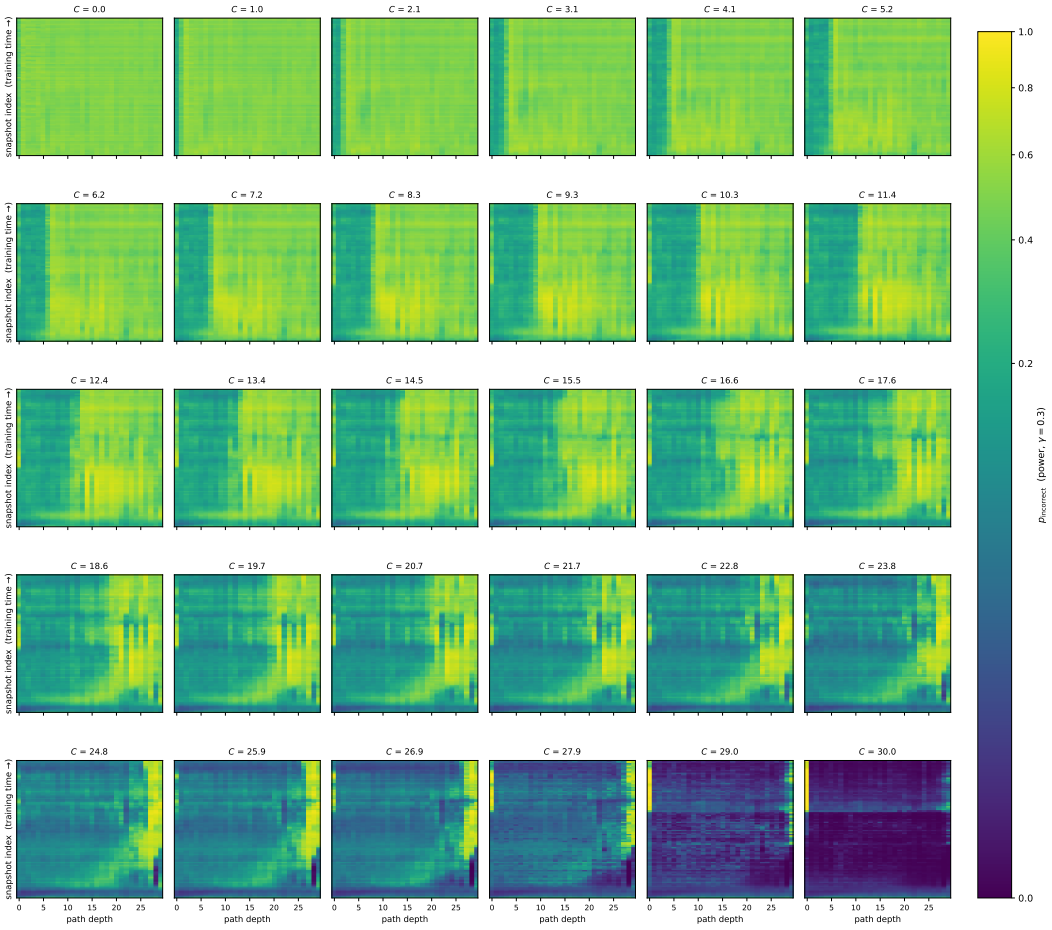


Figure 27: $P(\textbf{incorrect})$ **on the Easy Path — Rising Tide on Combined Modifications Tree Climb** $k = 30$**.** Return-threshold sweep: one panel per return threshold. Within each panel, each cell shows the policy's $P(\text{incorrect})$ when conditioned on that threshold, evaluated at the on-sequence observation for that (training step, depth) cell. Rows index training steps; columns index depth along the path. *An example seed of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*

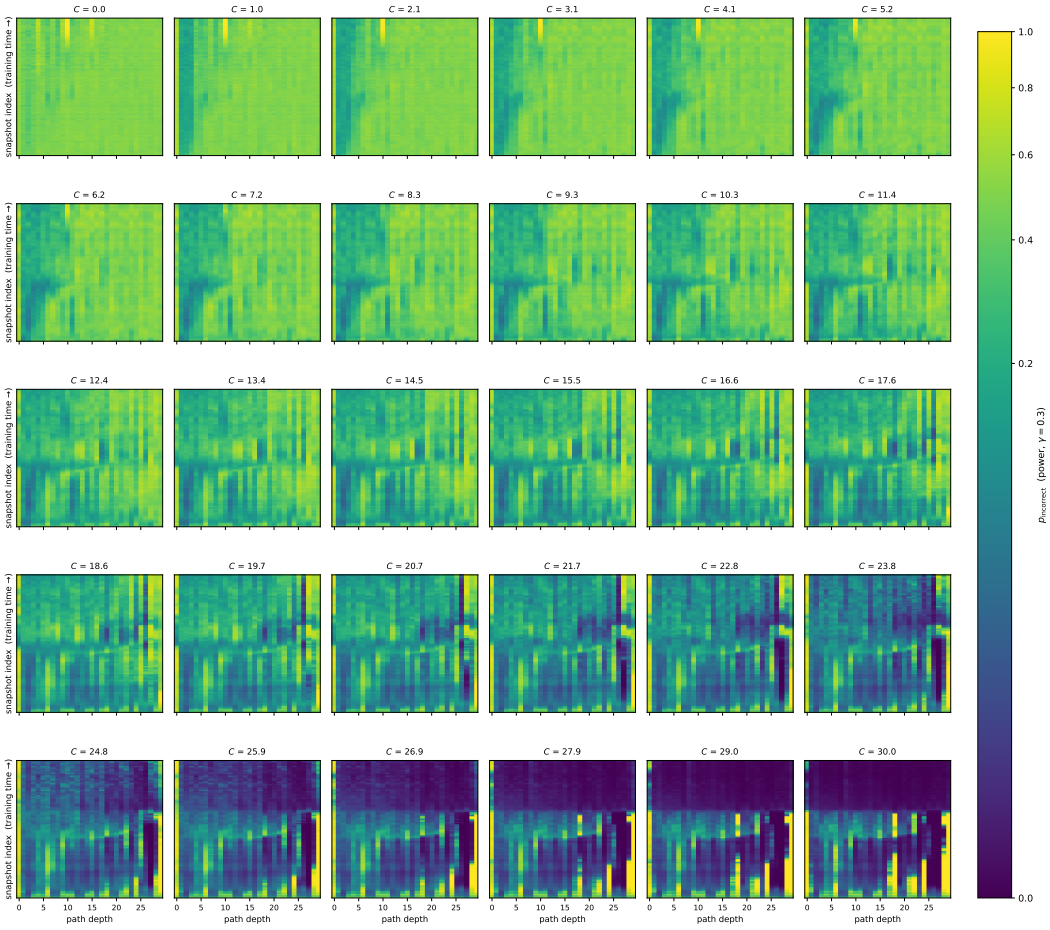


Figure 28: $P(\textbf{incorrect})$ **on the Jackpot Path — Rising Tide on Combined Modifications Tree Climb** $k = 30$**.** Return-threshold sweep: one panel per return threshold. Within each panel, each cell shows the policy's $P(\text{incorrect})$ when conditioned on that threshold, evaluated at the on-sequence observation for that (training step, depth) cell. Rows index training steps; columns index depth along the path. *An example seed of the best of the top 3 sweep (ranked by seed-averaged evaluation depth).*