PALS: Preference-guided Active Automata Learning for Symbolic Reinforcement Learning in Games

author names withheld

Under Review for NExT-Game 2026

Abstract

We introduce *PALS* (Preference-guided Active automata Learning for Symbolic reinforcement learning), an active automata learning framework that learns fully-symbolic policies for goal-directed games from a preference oracle and LTL safety specifications. *PALS* extends classical L^* by allowing both the hypothesis and the preference oracle to evolve as queries accumulate, with an MCTS-driven audit stage that surfaces deviations preferred over the current hypothesis and a shielding layer that patches the oracle whenever the hypothesis violates the safety specification. We demonstrate the utility of *PALS* on the Taxi Driver game from the Gymnasium benchmark, evaluate it against standard Q-learning and MCTS baselines on a suite of game-theoretic benchmarks, and provide a proof sketch establishing optimality under modest assumptions on the game structure. To the best of our knowledge, *PALS* is the first algorithm that fully symbolically learns reinforcement-learning policies for agents in games via automata learning.

1. Introduction

Symbolic reinforcement learning (SRL) sits at the intersection of formal methods and reinforcement learning, using symbolic representations and formal guarantees to address three core challenges in RL, namely verifiability, sample efficiency, and interpretability. Existing SRL techniques operate at varying degrees of symbolic commitment. Our work targets the fully-symbolic end of this landscape, where the policy itself is a symbolic object that can be inspected, verified, and reused. Recent work in this regime has shown promise but faces sharp limitations. On the synthesis side, Kouteili et al. [9] mine temporal stream logic specifications from demonstrations and synthesize optimal controllers for the Gymnasium toy text games [14], demonstrating that symbolic methods can extract policies through synthesis rather than merely constrain them. Neider et al. [11] sidestep the 2EXPTIME cost of synthesis with an L*-style automata learning approach for two-player games over infinite graphs. However, this framework is limited only to safety games and cannot express the liveness guarantees that goal-directed RL tasks require.

We introduce *PALS* (Preference-guided Active automata Learning for Symbolic reinforcement learning), an active automata learning framework that produces fully-symbolic policies for goal-directed games. Unlike prior fully-symbolic approaches, PALS handles both safety and liveness objectives, and learns policies that are not merely correct but preferentially optimal. The learner is guided by a preference oracle that encodes user-supplied notions of good play, lifting the framework out of the correctness-only regime that limits Neider et al. and avoiding the specification-design and scaling burdens that limit synthesis-based methods. We demonstrate *PALS* utility on a taxi driver

game, describe the algorithim, prove convergence of the learned behavior and evaluate *PALS* on a suite of game theory games to demonstrate it's overall utility.

2. Motivation

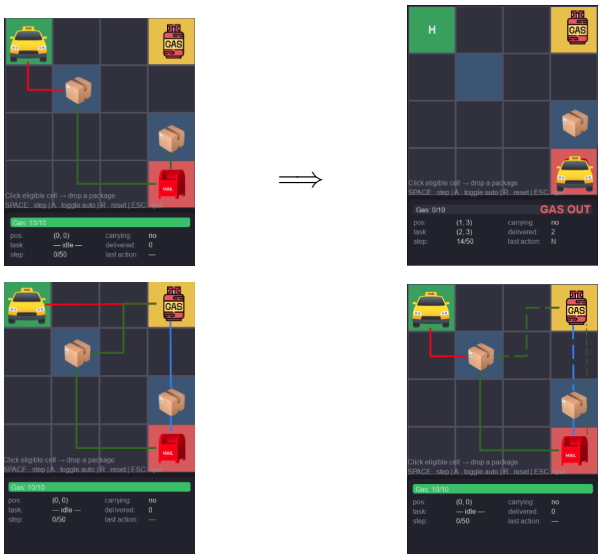

Figure 1: Three controllers for the Taxi Driver game. **Top:** Q-learning with a distance-minimizing reward follows the red path, then loops the green path and runs out of fuel. **Bottom-left:** controller synthesized from $F(P_1 \wedge P_2) \wedge G(Gas \neq Empty)$ delivers both packages safely but refuels before every pickup. **Bottom-right:** *PALS* delivers efficiently, refueling only when needed.

Reward hacking is a fundamental problem in reinforcement learning. Rather than acquiring the intended policy, RL agents often learn to exploit under-specified reward signals [13]. For example, a Tetris agent rewarded for survival pauses the game indefinitely [10], or a CoastRunners agent repeatedly crashing into walls to maximize a reward instead of finishing the race [4]. RL engineers spend considerable time hand-crafting reward functions that capture both task objectives and safety constraints, and empirically validating the learned behavior to avoid these failure modes.

Reactive synthesis offers a distinct paradigm for constructing controllers that execute specified behavior in games against an adversarial environment. Unlike RL, synthesis produces controllers that provably satisfy the task specification. However, this guarantee comes with significant caveats. Specification design is itself a difficult engineering problem, and a synthesized controller that satisfies the letter of its specification often fails to match what the engineer actually intended. Encoding notions of optimal strategy within a specification is similarly challenging, since temporal logics naturally express correctness rather than preference. Finally, synthesis scales poorly, as the classical reactive synthesis problem is 2EXPTIME-complete [12], limiting its applicability to small specifications.

The dichotomy between correctness and optimality is demonstrated in the popular RL game Taxi Driver from the Gymnasium benchmark [14]. This game requires a taxi driver to pick up and deliver packages to a predesignated drop-off site, while not running out of fuel. The taxi driver must drop

off a package before picking up a new one and can only move following taxicab geometry[1]. The top row of Figure 1 illustrates a failure mode of an underspecified reward learned using traditional RL. Rewarding the agent solely for minimizing the Manhattan distance to its current goal produces a policy that ignores refueling and eventually stalls. In contrast, the bottom-left of Figure 1 shows a controller synthesized from an LTL specification requiring that the taxi eventually delivers the package and never run out of gas. Given a realizable environment specification, synthesis returns a controller that provably satisfies these properties. However, correctness does not imply optimality: the synthesized controller refuels before every delivery, satisfying the specification but wasting time. Furthermore, even if the synthesized strategy was guaranteed to be optimal, the intractability of synthesis on grid world games is well documented [9].

We introduce PALS, an active automata learning algorithm [7] for learning controllers, offering better scalability than reactive synthesis by replacing exhaustive specification-based construction with query-guided exploration. PALS is driven by a preference oracle that encodes a locally optimal strategy, paired with an MCTS-based stage that explores alternatives to the learned hypothesis and updates the oracle's preferences where they prove suboptimal, so the controller progressively evolves toward a globally preferred behavior. To enforce correctness, PALS combines a standard PAC equivalence check with a shielding-style safety layer [1] that extracts counterexamples whenever the hypothesis violates the safety specification and updates the preferences accordingly. The bottom-right of Figure 1 learns an optimal controller using a preference oracle that is equivalent to the reward system. However, the safety specification ensures that this preferential behavior is overwritten when it leads to unsafe behaviors. The resulting controller follows the oracle's optimal behavior and only deviates where it would otherwise be unsafe, sidestepping the dichotomy between reward optimality and safety without requiring a hand-engineered complicated reward function.

3. *PALS* Algorithm

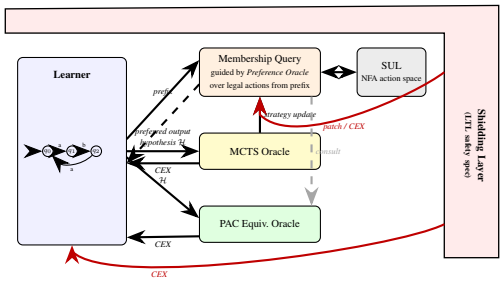

Figure 2: Architecture of *PALS*. An L*-style Learner queries a Preference Oracle over the SUL's actions; each hypothesis is then refined by an MCTS Oracle (UCB-scored deviations), a PAC Oracle (probabilistic agreement), and a Shielding Layer (LTL safety patches).

PALS departs from classical active automata learning by learning a policy over an action space from a preference oracle rather than a fixed language from a teacher who knows the target. Both the learner's hypothesis and the preference oracle guiding it are therefore allowed to evolve as queries accumulate. Specifically, the preference oracle evolves so that it is consistent with downstream traces.

1. The taxi driver can only move up, down, left, and right. No diagonal movement

3.1. *PALS* Setup

Let $A = (Q, \Sigma, \Gamma, \delta, q_0)$ be a nondeterministic finite automaton (NFA) with output, encoding the action space of the system player. Here Q is a finite set of states, Σ is the input alphabet, Γ is the output alphabet, $q_0 \in Q$ is the initial state, and $\delta : Q \times \Sigma \to 2^{Q \times \Gamma}$ is the transition relation. The nondeterminism lies on the output: at each state q with input σ, the relation $\delta(q, \sigma) \subseteq Q \times \Gamma$ specifies the set of legal output-successor pairs.

A trace of A over n time steps is an alternating I/O sequence $\tau = \sigma_1 \lambda_1 \sigma_2 \lambda_2 \cdots \sigma_n \lambda_n$, where at each step i the system reads $\sigma_i \in \Sigma$ and emits $\lambda_i \in \Gamma$. Writing q_i for the state reached after step i, the trace is admissible in A if $(q_i, \lambda_i) \in \delta(q_{i-1}, \sigma_i)$ for all i.

A preference oracle is a total order $\succeq$ over admissible traces, where $\tau_1 \succeq \tau_2$ asserts that τ_1 is at least as good play for the system player as τ_2. Given an admissible prefix $\tau\sigma$ at step $n+1$, the oracle selects the locally optimal output

$$\lambda^*(\tau\sigma) \;=\; \arg\max_{\lambda} \; \tau\sigma\lambda \quad \text{over } \lambda \text{ such that } \tau\sigma\lambda \text{ is admissible,}$$

with ties broken consistently but arbitrarily.

The system under learning is the pair $(A, \succeq)$, and a membership query at prefix $\tau\sigma$ returns $\lambda^*(\tau\sigma)$. Suppose $\succeq$ is globally correct, in the sense that for every prefix the locally optimal $\lambda^*(\tau\sigma)$ extends to a globally optimal trace, and suppose the PAC oracle is replaced by a true equivalence oracle over A. Then every membership query returns the optimal output, the MCTS oracle finds no preferred deviation and yields no counterexample, and the shielding layer is vacuous since any trace realizing globally optimal play is by assumption safe. *PALS* in this regime reduces exactly to Angluin's L* over the Mealy machine induced by $(A, \succeq)$ [2].

3.2. MCTS Oracle

Input : Hypothesis $\mathcal{H}$, NFA A, preference oracle $\succeq$, observation tables A and B, rollout budget K, depth N
Output: Counterexample prefix $\tau\sigma$, or $\perp$ if $\mathcal{H}$ is optimal
for $k \leftarrow 1$ **to** K **do**
  $\tau\sigma \leftarrow$ `SampleSubtrace`$(\mathcal{H})$ `// deviation point in` $\mathcal{H}$
  $\mathcal{S}_{\mathcal{H}} \leftarrow$ `Rollout`$(\mathcal{H}, \tau\sigma, N)$ `// follow` $\mathcal{H}$ `to depth` N
  $\mathcal{S}_{\text{dev}} \leftarrow$ `Rollout`$(B, \tau\sigma, N)$ `// UCB-guided deviation`
  $\pi \leftarrow$ `Compare`$(\mathcal{S}_{\text{dev}}, \mathcal{S}_{\mathcal{H}}, \succeq)$ `// pairwise majority vote`
  $V \leftarrow$ `SMTAssign`$(\mathcal{S}_{\text{dev}} \cup \mathcal{S}_{\mathcal{H}}, \succeq)$ `// leaf values in` $[0, 1]$
  `Backprop`(B, V) `// softmax up the tree, increment visits`
  if $\pi > \frac{1}{2}$ **then**
    $\lambda_{\text{dev}} \leftarrow$ first system output of $\mathcal{S}_{\text{dev}}$ after $\tau\sigma$ Overwrite preference at $\tau\sigma$ to return λ_{dev} Add $\tau\sigma\lambda_{\text{dev}}$ to A; wipe $\lambda_{\mathcal{H}}$-derived traces from $\mathcal{H}$ **return** $\tau\sigma$ `// counterexample`
  end
end
return $\perp$ `// no deviation found in` K `rollouts;` $\mathcal{H}$ `is optimal under` $\succeq$

Algorithm 1: MCTS oracle (For more details on the various components of the MCTS oracle see Appendix A)

An optimal preference oracle $\succeq$ makes recovering policies trivial. While optimal policies are derivable in game theory games, the assumption of a consistent high-quality preference oracle is unrealistic. Instead, PALS has a weaker preference oracle that reasons only on the history of two traces and says which one is better. At each step, PALS selects the $\lambda_i \in \Lambda(\tau\sigma_i)$ whose extended trace is maximally preferred under $\succeq$. This greedy rule is locally optimal but can ignore globally better continuations downstream.

Alongside the L* observation table A, PALS maintains a secondary table B that records every legal output alternative skipped by the preference oracle, ranked by UCB statistics and seeded as A

commits to choices. After each L^* hypothesis $\mathcal{H}$, the MCTS oracle samples deviation points, rolls out depth-N subtrees from both $\mathcal{H}$ and the deviation, and compares the trace sets pairwise under $\succeq$. Successful deviations correct $\mathcal{H}$ by overwriting the local preference and re-running L^* closure; unsuccessful ones still refine B's UCB scores via SMT-derived back-propagation [5]. After K audits without a deviation, $\mathcal{H}$ is deemed locally optimal under $\succeq$ and control passes to equivalence checking.

3.3. *PALS* Sheidling Layer

PALS currently considers safety specifications and assumes that any provided specification is realizable with respect to the environment. Extension to liveness specifications requires Büchi acceptance over the product automaton, which is left to future work.

PALS extends the preference-update mechanism from the MCTS oracle to enforce safety. When the model checker returns a counterexample trace witnessing a violation of the LTL specification, PALS extracts the longest prefix $\tau\sigma$ along the trace at which a different output would have averted the violation, and patches the preference oracle so that future membership queries at $\tau\sigma$ return that corrective output. Because the violation is known, the corrective output is computed by a domain-specific algorithm that returns a safety-abiding continuation from $\tau\sigma$. The hypothesis traces extending $\tau\sigma$ under the original output are wiped, and L^* closure is re-run against the patched preference oracle. PALS trusts that re-running closure yields a hypothesis that respects both the safety specification and the preference oracle, recovering optimal play under $\succeq$ wherever the specification permits. This follows the standard shielding pattern of falling back to a known-safe action whenever the learned controller would otherwise leave the safe region [1, 3].

3.4. Discovery of Optimal Behavior Through *PALS*

We provide a proof sketch for PALS under two simplifying assumptions on the underlying game. Firstly, the system player's objective is stationary across time, in the sense that the criterion under which two traces are compared by $\succeq$ does not itself depend on game history. Games requiring genuinely evolving strategies, such as iterated prisoner's dilemma where the optimal play at one stage is not informative of optimal play later, fall outside this setting. Locally optimal strategies under $\succeq$ may fail to match the globally optimal strategy, but they are nonetheless optimizing the same underlying objective. Secondly, the globally optimal strategy is reachable from the locally optimal strategy by at most N system-output deviations, where N is the MCTS rollout depth. That is, for every prefix $\tau\sigma$ on the locally optimal trace, the globally optimal continuation differs from the locally optimal continuation in at most N output choices somewhere in the hypothesis language.

Under Assumptions 1 and 2, the MCTS audit converges to the globally optimal policy as the rollout budget K grows. The argument follows the standard UCT convergence theorem [8]. By the UCB regret bound, every action sequence of length at most N from a deviation point $\tau\sigma$ is visited infinitely often as $K \to \infty$, and the value estimates v in B converge in probability to the expected preference values induced by $\succeq$. Combined with Assumption 2, which guarantees the globally optimal continuation lies within the depth-N rollout subtree, this implies that any locally suboptimal commitment $\lambda_{\mathcal{H}}$ at $\tau\sigma$ is eventually detected, the pairwise majority vote flips, and PALS overwrites the preference. With probability 1, PALS converges to the globally optimal policy under $\succeq$.

4. Results

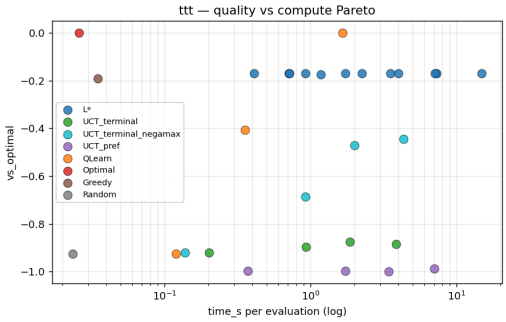

Figure 3: PALS (Denoted L* algorithm) VS. Q Learning VS. Various MCTS algorithms all trained with sub-optimal preference oracles

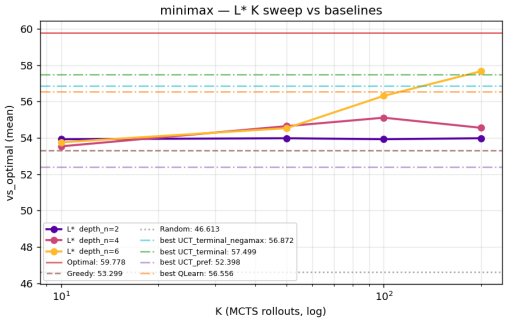

Figure 4: PALS average score as K rollouts increases for deep minimax games with randomized values on nodes: depths of game tree 10,12,14

We evaluate PALS on a suite of standard game-theoretic benchmarks including Minimax with values assigned to every node, Tic-Tac-Toe, Nim, and Dots and Boxes. For each game we construct a suboptimal preference oracle that plays greedily according to a hand-chosen heuristic, and compare *PALS* against several MCTS variants run with varying rollout budgets K and against Q-learning trained with a suboptimal reward function derived from the same heuristic. Since these benchmarks isolate policy quality from correctness, no safety specifications are imposed and the shielding layer is inactive. The charts above report the best score achieved by each method, with full results in Appendix B. Across all four benchmarks *PALS* performs better than or comparably to the strongest baselines. Figure 3 shows that on Tic-Tac-Toe against an optimal opponent, *PALS* consistently learns a strong strategy from a suboptimal preference oracle while UCT-based MCTS variants fail to match its performance at most rollout budgets. Figure 4 reports *PALS*'s average score on deep random-valued Minimax trees with rollout depth $N \ll D$, where it exhibits positive, monotonically increasing performance in K, empirically validating the proof sketch of subsection 3.4 even in regimes where Assumption 2 does not strictly hold.

5. Discussion

We present PALS, an active automata learning framework that learns fully-symbolic policies for goal-directed games from a preference oracle and an LTL safety specification. The hypothesis automaton and the preference oracle co-evolve: MCTS-driven audits surface new prefixes that update the oracle, and automata learning lifts these pointwise updates into a refined symbolic hypothesis whose closure guides the next round of exploration, while a shielding layer enforces the safety specification throughout. PALS recovers the optimal policy where the specification permits and falls back to safe behavior where it does not, with convergence following from the standard UCT guarantees. Empirically, PALS matches or exceeds the strongest MCTS and Q-learning baselines on game-theoretic benchmarks while learning from a suboptimal preference oracle, taking a step toward auditable symbolic RL where policies are inspectable automata rather than opaque networks.

References

[1] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[2] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[3] Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis: Runtime enforcement for reactive systems. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 533–548, 2015.

[4] Jack Clark and Dario Amodei. Faulty reward functions in the wild. OpenAI Blog, 2016. https://openai.com/blog/faulty-reward-functions/.

[5] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[6] Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for switching bandit problems. In *International conference on algorithmic learning theory*, pages 174–188. Springer, 2011.

[7] Malte Isberner. Foundations of active automata learning: an algorithmic perspective. 2015.

[8] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293, Berlin, Heidelberg, 2006. Springer.

[9] Sam Nicholas Kouteili, William Fishell, Christian Scaff, Mark Santolucito, and Ruzica Piskac. Mining beyond the bools: Learning data transformations and temporal specifications. *arXiv preprint arXiv:2603.06710*, 2026.

[10] Tom Murphy, VII. The first level of Super Mario Bros. is easy with lexicographic orderings and time travel...after that it gets a little tricky. SIGBOVIK, 2013. https://www.cs.cmu.edu/~tom7/mario/mario.pdf.

[11] Daniel Neider and Ufuk Topcu. An automaton learning approach to solving safety games over infinite graphs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 204–221. Springer, 2016.

[12] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190, 1989.

[13] Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. *Advances in Neural Information Processing Systems*, 35:9460–9471, 2022.

[14] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Appendix A. MCTS Oracle Details

PALS maintains a secondary observation table B to enable structured exploration of the alternatives passed over by the hypothesis during the MCTS query phase. The main observation table A records only the outputs selected by the preference oracle. B records every legal output encountered during learning along with UCB statistics that guide the MCTS rollouts. Each membership query $\lambda^*(\tau\sigma)$ seeds B with the full set of sibling outputs $\lambda \in \Lambda(\tau\sigma)$, so unvisited alternatives are catalogued the moment A first commits to a choice.

Each entry in B is a tuple $(\tau\sigma\lambda, v, n)$ where $\tau\sigma\lambda$ is a trace prefix ending in a system output, $v \in [0, 1]$ is its current value, and n its visit count. Newly created entries are seeded with optimistic defaults ($v = 0.5$, $n = 0$), ensuring unexplored alternatives are tried at least once before being deprioritized [6]. The UCB score

$$\mathrm{UCB}(\tau\sigma\lambda) \;=\; v \;+\; c\sqrt{\tfrac{\ln N}{n}} \cdot \alpha^{-d}$$

combines the value (exploitation) with an exploration bonus, discounted by depth $d = |\tau\sigma\lambda|$, where N is the total visit count across siblings of $\tau\sigma\lambda$ in B, c is the exploration constant, and $\alpha > 1$ damps exploration at greater depth. Nodes not yet in B are added on demand by querying the NFA for the legal continuations at that state, so B grows lazily as the search expands.

Once L* produces a hypothesis $\mathcal{H}$, the MCTS oracle audits it for greedy suboptimality. From a deviation point $\tau\sigma$ in $\mathcal{H}$, MCTS rolls out subtrees of fixed depth $N \ll D$, where D is the depth of the game tree. Rollouts branch over all environments inputs in Σ and sample system outputs from $\Lambda(\tau\sigma)$ proportional to their UCB scores in B, producing two sets of depth-N traces: $\mathcal{S}_{\mathcal{H}}$ from following $\mathcal{H}$, and $\mathcal{S}_{\mathrm{dev}}$ from the deviation. Every $\tau_{\mathrm{dev}} \in \mathcal{S}_{\mathrm{dev}}$ is compared pairwise under $\succeq$ against every $\tau_{\mathcal{H}} \in \mathcal{S}_{\mathcal{H}}$. If a strict majority prefer $\mathcal{S}_{\mathrm{dev}}$, the greedy choice $\lambda_{\mathcal{H}}$ at $\tau\sigma$ was inconsistent, and PALS overwrites the preference at $\tau\sigma$ to return λ_{dev}, wipes the $\lambda_{\mathcal{H}}$-derived traces from $\mathcal{H}$, adds the λ_{dev} prefix to A, and triggers L* closure.

In parallel, the depth-N leaves of both sets are submitted to an SMT solver with bounded range $[0, 1]$ that assigns each leaf a value consistent with the observed pairwise preferences. These leaf values back-propagate up the rollout tree as a softmax-weighted average, $v(\mathrm{node}) = \sum_c \frac{e^{v(c)}}{\sum_{c'} e^{v(c')}} \cdot v(c)$, with visit counts incremented along the way. The updated (v, n) entries refine the UCB scores governing future rollouts. Unlike A, which is trimmed on every successful deviation, B is append-only and accumulates preference information across all rounds.

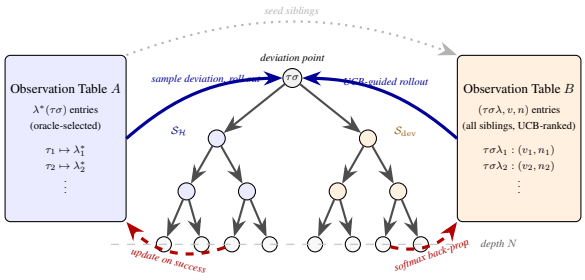

Figure 5: Schematic of the MCTS oracle's interaction with A and B and rolling out of counter strategy.

Appendix B. Game Results

Table 1: Minimax: top 8 strategies by vs_optimal.

strategy	params	vs_rand	vs_grdy	vs_opt	states	time (s)	n
Optimal	–	73.84	69.45	59.78	–	0.013	9
L*	d=6, K=200	73.80	67.93	57.67	38.33	12.249	9
UCT_terminal	K=200	74.84	67.19	57.50	–	4.438	9
UCT_terminal	K=100	74.75	66.77	56.94	–	2.020	9
UCT_terminal_neg.	K=200	74.52	67.68	56.87	–	5.185	9
UCT_terminal_neg.	K=100	74.44	67.34	56.62	–	2.335	9
QLearn	50000 ep.	65.54	59.27	56.56	–	0.784	9
L*	d=6, K=100	72.87	67.42	56.31	38.22	3.906	9

Table 2: Minimax: L* depth_n $\times$ K grid. vs_optimal (top) and time in seconds (bottom).

vs_optimal				
d	K=10	K=50	K=100	K=200
2	53.93	53.98	53.93	53.98
4	53.54	54.65	55.11	54.56
6	53.76	54.54	56.31	57.67
time (s)				
d	K=10	K=50	K=100	K=200
2	0.075	0.149	0.281	0.378
4	0.115	0.621	1.405	2.572
6	0.209	2.247	3.906	12.249

Table 3: Minimax: best variant of each strategy class.

strategy	params	vs_rand	vs_grdy	vs_opt	time (s)	n
Optimal	–	73.84	69.45	59.78	0.013	9
L*	d=6, K=200	73.80	67.93	57.67	12.249	9
UCT_terminal	K=200	74.84	67.19	57.50	4.438	9
UCT_terminal_neg.	K=200	74.52	67.68	56.87	5.185	9
QLearn	50000 ep.	65.54	59.27	56.56	0.784	9
Greedy	–	70.70	66.73	53.30	0.013	9
UCT_pref	K=200	69.74	61.21	52.40	5.299	9
Random	–	60.91	50.58	46.61	0.011	9

Table 4: Dots and Boxes: top 8 strategies by vs_optimal.

strategy	params	vs_rand	vs_grdy	vs_opt	states	time (s)	n
UCT_pref	K=10	0.719	0.086	0.000	–	0.190	4
L*	d=4, K=50	0.880	0.641	0.000	23.50	0.224	4
UCT_pref	K=50	0.715	0.075	0.000	–	0.859	4
L*	d=4, K=200	0.880	0.641	0.000	23.50	0.796	4
L*	d=6, K=100	0.880	0.641	0.000	23.50	0.394	4
L*	d=4, K=100	0.880	0.641	0.000	23.50	0.400	4
L*	d=2, K=200	0.880	0.641	0.000	23.50	0.383	4
L*	d=6, K=50	0.880	0.641	0.000	23.50	0.239	4

Table 5: Dots and Boxes: L* depth_n $\times$ K grid. vs_optimal (top) and time in seconds (bottom).

vs_optimal				
d	K=10	K=50	K=100	K=200
2	0.000	0.000	0.000	0.000
4	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000
time (s)				
d	K=10	K=50	K=100	K=200
2	0.049	0.111	0.203	0.383
4	0.069	0.224	0.400	0.796
6	0.060	0.239	0.394	0.792

Table 6: Dots and Boxes: best variant of each strategy class.

strategy	params	vs_rand	vs_grdy	vs_opt	time (s)	n
UCT_pref	K=10	0.719	0.086	0.000	0.190	4
L*	d=2, K=10	0.880	0.641	0.000	0.049	4
Random	–	0.293	-0.335	-0.500	0.035	2
Greedy	–	0.705	0.115	-0.500	0.031	2
Optimal	–	0.803	0.510	-0.500	0.032	2
UCT_terminal	K=10	0.620	-0.295	-0.500	0.674	2
UCT_terminal_neg.	K=10	0.603	-0.293	-0.500	0.183	2
QLearn	1000 ep.	0.533	-0.010	-0.500	0.093	2

Table 7: Nim: top 8 strategies by vs_optimal.

strategy	params	vs_rand	vs_grdy	vs_opt	states	time (s)	n
Optimal	–	0.965	1.000	-0.500	–	0.017	4
UCT_terminal_neg.	K=100	0.948	0.650	-0.500	–	1.223	4
UCT_terminal_neg.	K=200	0.955	0.568	-0.500	–	2.590	4
QLearn	1000 ep.	0.955	1.000	-0.500	–	0.035	4
QLearn	10000 ep.	0.965	1.000	-0.500	–	0.183	4
QLearn	50000 ep.	0.965	1.000	-0.500	–	0.767	4
L*	d=4, K=10	0.862	0.625	-0.500	15.94	0.099	16
L*	d=4, K=100	0.926	0.625	-0.500	17.00	1.616	16

Table 8: Nim: L* depth_n $\times$ K grid. vs_optimal (top) and time in seconds (bottom).

vs_optimal				
d	K=10	K=50	K=100	K=200
2	-0.548	-0.523	-0.523	-0.525
4	-0.500	-0.525	-0.500	-0.523
time (s)				
d	K=10	K=50	K=100	K=200
2	0.056	0.322	0.500	5.701
4	0.099	0.447	1.616	4.489

Table 9: Nim: best variant of each strategy class.

strategy	params	vs_rand	vs_grdy	vs_opt	time (s)	n
Optimal	–	0.965	1.000	-0.500	0.017	4
UCT_terminal_neg.	K=100	0.948	0.650	-0.500	1.223	4
QLearn	1000 ep.	0.955	1.000	-0.500	0.035	4
L*	d=4, K=10	0.862	0.625	-0.500	0.099	16
UCT_terminal	K=50	0.698	-0.045	-0.670	0.443	4
Greedy	–	0.285	0.500	-0.778	0.022	4
Random	–	-0.048	-0.333	-0.903	0.016	4
UCT_pref	K=50	-0.038	-0.381	-0.924	0.971	16

Table 10: Tic-Tac-Toe: top 8 strategies by vs_optimal.

strategy	params	vs_rand	vs_grdy	vs_opt	states	time (s)	n
Optimal	–	0.785	0.000	0.000	–	0.026	1
QLearn	50000 ep.	0.785	0.000	0.000	–	1.653	1
L*	d=2, K=10	0.840	0.000	-0.169	96.25	0.410	4
L*	d=3, K=10	0.840	0.000	-0.169	96.25	0.712	4
L*	d=4, K=10	0.840	0.000	-0.169	96.25	0.924	4
L*	d=2, K=50	0.840	0.000	-0.169	96.75	0.715	4
L*	d=3, K=50	0.840	0.000	-0.169	96.25	1.723	4
L*	d=4, K=50	0.840	0.000	-0.169	96.25	3.991	4

Table 11: Tic-Tac-Toe: L* depth_n $\times$ K grid. vs_optimal (top) and time in seconds (bottom).

vs_optimal				
d	K=10	K=50	K=100	K=200
2	-0.169	-0.169	-0.174	-0.169
3	-0.169	-0.169	-0.169	-0.169
4	-0.169	-0.169	-0.169	-0.169
time (s)				
d	K=10	K=50	K=100	K=200
2	0.410	0.715	1.174	2.246
3	0.712	1.723	3.530	7.087
4	0.924	3.991	7.298	14.779

Table 12: Tic-Tac-Toe: best variant of each strategy class.

strategy	params	vs_rand	vs_grdy	vs_opt	time (s)	n
Optimal	–	0.785	0.000	0.000	0.026	1
QLearn	50000 ep.	0.785	0.000	0.000	1.653	1
L*	d=2, K=10	0.840	0.000	-0.169	0.410	4
Greedy	–	0.705	0.000	-0.190	0.035	1
UCT_terminal_neg.	K=200	0.905	-0.020	-0.445	4.332	1
UCT_terminal	K=100	0.800	-0.220	-0.875	1.851	1
Random	–	-0.315	-0.960	-0.925	0.024	1
UCT_pref	K=200	0.078	-0.983	-0.988	7.016	4