Toward Efficient Exploration by Large Language Model Agents

Dilip Arumugam [1] **Thomas L. Griffiths** [1 2]

Abstract

A nascent area within reinforcement learning (RL) is the design of sequential decision-making agents centered around large language models (LLMs). While autonomous decision-making agents powered by modern LLMs could facilitate numerous real-world applications, such successes demand agents that are capable of data-efficient RL. One key obstacle to achieving data efficiency in RL is exploration, a challenge that we demonstrate many recent proposals for LLM agent designs struggle to contend with. Meanwhile, classic algorithms from the RL literature known to gracefully address exploration require technical machinery that is challenging to operationalize in purely natural language settings. In this work, rather than relying on finetuning or in-context learning to coax LLMs into implicitly imitating a RL algorithm, we illustrate how LLMs can be used to explicitly implement an existing RL algorithm (Posterior Sampling for Reinforcement Learning) whose capacity for statistically-efficient exploration is already well-studied. We offer empirical results demonstrating how our LLM-based implementation of a known, data-efficient RL algorithm can be considerably more effective in natural language tasks that demand prudent exploration.

1. Introduction

Large language models (LLMs) have rapidly permeated many areas of machine learning, demonstrating proficiency across a broad range of tasks (Bommasani et al., 2021; Achiam et al., 2023; Touvron et al., 2023; Team et al., 2023; Hurst et al., 2024; Jaech et al., 2024). This has inspired recent work studying how LLMs can best be used to solve sequential decision-making problems. These efforts have led to the introduction of new designs for LLM agents that aim to learn optimal behavior through trial-and-error interaction within natural language environments (Yao et al., 2023; Shinn et al., 2024; Monea et al., 2024; Klissarov et al., 2024). While details vary by approach, broadly speaking these new agent designs involve one or more LLMs that interact to ultimately select actions at each timestep within the environment. However, such agents still reside in the classic RL setting (Sutton & Barto, 1998) and, consequently, must still grapple with the fundamental obstacles to data efficiency (generalization, exploration, & credit assignment) that the RL literature has studied for decades.

While composing LLMs to arrive at new agent designs is the current norm, we propose that an alternative strategy is to re-examine existing RL algorithms and consider how LLMs might implement them in otherwise inaccessible environments. An RL algorithm consists of specifying inputs and detailing a sequence of steps for determining behavior at each time period. Why should the emergence and proliferation of LLMs change the fundamental principles of agent design? Instead, perhaps LLMs can be used to create a new, potentially-inexact incarnation of existing RL algorithms and the subroutines needed to implement them.

In this work, we focus on data-efficient RL with LLMs and isolate the key challenge of exploration. We demonstrate how modern LLMs afford a contemporary implementation of an existing RL algorithm, Posterior Sampling for Reinforcement Learning (PSRL) (Strens, 2000; Osband et al., 2013), that is both well-studied and whose capacity for good exploration is already known to yield provably-efficient RL in a number of problem classes. We find that, in environments with deterministic transition dynamics, our LLM-based implementation of PSRL retains the strong exploration properties that, up to this point, have not only been primarily restricted to tabular domains but also been absent in current, more-traditional designs for LLM agents. Our work underscores the importance of addressing exploration in the design of LLM agents, illustrates the considerable value that decades of RL research have to offer data-efficient decision-making with LLMs, and establishes a key distinction between LLMs that implement a RL algorithm versus a RL algorithm that is implemented with LLMs.

[1]Department of Computer Science, Princeton University. [2]Department of Psychology, Princeton University. Correspondence to: Dilip Arumugam <dilip.a@cs.princeton.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

2. Problem Formulation

For any arbitrary set $\mathcal{X}$, we use $\Delta(\mathcal{X})$ to denote the set of all probability distributions with support on $\mathcal{X}$. For any $N \in \mathbb{N}$, we denote the index set as $[N] = \{1, 2, \ldots, N\}$.

We formulate a sequential decision-making problem as a finite-horizon, episodic Markov Decision Process (MDP) (Bellman, 1957; Puterman, 1994) defined by $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \beta, H \rangle$. $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is a reward function providing evaluative feedback in the unit interval, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is a transition function prescribing distributions over next states, $\beta \in \Delta(\mathcal{S})$ is an initial state distribution, and $H \in \mathbb{N}$ is the maximum episode length or horizon. Within each of $K \in \mathbb{N}$ total episodes, the agent acts for H steps beginning with an initial state $s_1 \sim \beta(\cdot)$ and, at each timestep $h \in [H]$, observes the current state $s_h \in \mathcal{S}$, selects an action $a_h \in \mathcal{A}$, enjoys a reward $r_h = \mathcal{R}(s_h, a_h)$, and transitions to a next state $s_{h+1} \sim \mathcal{T}(\cdot \mid s_h, a_h)$.

An agent is characterized by its non-stationary, stochastic policy $\pi : \mathcal{S} \times [H] \rightarrow \Delta(\mathcal{A})$, which encodes a pattern of behavior by mapping individual states and the current timestep to a probability distribution over actions. We assess the performance of a policy π in MDP $\mathcal{M}$ at timestep $h \in [H]$ when starting at state $s \in \mathcal{S}$ and taking action $a \in \mathcal{A}$ by its associated action-value function $Q^{\pi}_{\mathcal{M},h}(s, a) = \mathbb{E}\left[\sum\limits_{h'=h}^{H} \mathcal{R}(s_{h'}, a_{h'}) \mid s_h = s, a_h = a\right]$. Taking the value function as $V^{\pi}_{\mathcal{M},h}(s) = \mathbb{E}_{a \sim \pi_h(\cdot \mid s)}\left[Q^{\pi}_{\mathcal{M},h}(s, a)\right]$, we define the optimal policy $\pi^{\star}$ as achieving supremal value $V^{\star}_{\mathcal{M},h}(s) = \sup\limits_{\pi \in \Pi} V^{\pi}_{\mathcal{M},h}(s)$ for all $s \in \mathcal{S}$, $h \in [H]$ where Π denotes the class of all non-stationary, stochastic policies. For any episode $k \in [K]$, we let $\tau_k = (s^{(k)}_1, a^{(k)}_1, r^{(k)}_1, \ldots, s^{(k)}_H, a^{(k)}_H, r^{(k)}_H, s^{(k)}_{H+1})$ denote the random trajectory experienced by the agent executing its policy in the environment. Meanwhile, $H_k = \{\tau_1, \tau_2, \ldots, \tau_{k-1}\} \in \mathcal{H}$ is the entire random history of agent interaction at the start of the kth episode.

Abstractly, a RL algorithm is a sequence $\{\pi^{(k)}\}_{k \in [K]}$ where the policy deployed at each episode $\pi^{(k)}$ is a function of the current history H_k. We may evaluate the performance of a RL algorithm on MDP $\mathcal{M}$ via its cumulative regret: $\textsc{Regret}(\{\pi^{(k)}\}_{k \in [K]}, \mathcal{M}) = \mathbb{E}\left[\sum\limits_{k=1}^{K}\left(V^{\star}_{\mathcal{M},1}(s_1) - V^{\pi^{(k)}}_{\mathcal{M},1}(s_1)\right)\right]$, which measures the total performance shortfall between an agent's chosen policy and the optimal policy across all episodes.

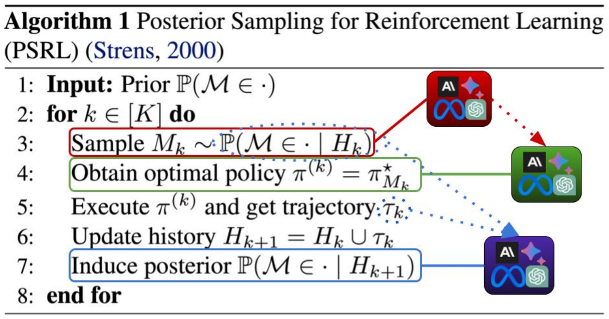
3. A LLM-Based Implementation of Posterior Sampling for Reinforcement Learning

One of the major obstacles to data-efficient RL is exploration, where a learner must determine what data to collect from the environment to maximize long-term performance. While much of the early work on addressing exploration in RL (please see Appendix A for an overview of related work) adhered to "optimism in the face of uncertainty," an alternative is to proceed in a Bayesian fashion. The Bayesian RL setting (Bellman & Kalaba, 1959; Duff, 2002; Ghavamzadeh et al., 2015) recognizes that the underlying MDP is entirely unknown to the agent and, therefore, a random variable. The agent is thus endowed with a prior distribution $\mathbb{P}(\mathcal{M} \in \cdot)$ to reflect initial uncertainty in the true MDP. We make a standard assumption that this prior distribution is well-specified such that the true MDP resides in its support. While the standard RL objective (Sutton & Barto, 1998) calls for an agent to minimize regret, another performance criterion is the Bayesian regret, which simply integrates out the randomness in $\mathcal{M}$ with respect to an agent's prior: $\textsc{BayesRegret}(\{\pi^{(k)}\}_{k \in [K]}) = \mathbb{E}\left[\textsc{Regret}(\{\pi^{(k)}\}_{k \in [K]}, \mathcal{M})\right]$.

Unfortunately, the canonical Bayes-Adaptive MDP (BAMDP) (Bellman & Kalaba, 1959; Duff, 2002) that encapsulates the full Bayesian RL problem is often computationally-intractable even in the simplest classes of environments, such as tabular MDPs. This is a direct consequence of the intractably-large BAMDP hyperstate space (Duff, 2002; Arumugam & Singh, 2022), in which traditional MDP states are folded in alongside *epistemic states* (Lu et al., 2023) that contain an agent's beliefs and epistemic uncertainty (Der Kiureghian & Ditlevsen, 2009) about the world. While states and actions of the MDP are considered known, the transition and reward functions are unknown to a RL agent (otherwise, the agent would face a planning, rather than RL, problem). With each step taken in the true environment, the resulting immediate reward and next-state transition provide ground-truth observations with which the agent may obtain posterior beliefs about the underlying MDP $\mathcal{M}$; even for a simple finite MDP, one episode of interaction results in a hyperstate space that is exponentially-large in the problem horizon H. While the true MDP state at each timestep is essential to good decision-making, one might hope that the latter epistemic state could be lazily updated while still allowing an agent to strategically explore the world by reducing epistemic uncertainty. This insight is the basis of posterior-sampling methods in RL.

3.1. The Classic Approach

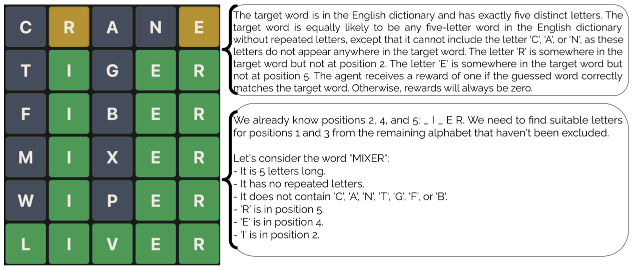
The promise of Bayesian RL methods is to facilitate statistically-efficient exploration by reducing an agent's epistemic uncertainty about the world. One strategy for



Algorithm 1 Posterior Sampling for Reinforcement Learning (PSRL) (Strens, 2000)

```
1: Input: Prior $\mathbb{P}(\mathcal{M} \in \cdot)$
2: for $k \in [K]$ do
3:     Sample $M_k \sim \mathbb{P}(\mathcal{M} \in \cdot \mid H_k)$
4:     Obtain optimal policy $\pi^{(k)} = \pi^{\star}_{M_k}$
5:     Execute $\pi^{(k)}$ and get trajectory $\tau_k$
6:     Update history $H_{k+1} = H_k \cup \tau_k$
7:     Induce posterior $\mathbb{P}(\mathcal{M} \in \cdot \mid H_{k+1})$
8: end for
```

Figure 1. The PSRL algorithm with LLM subroutines of posterior sampling, optimal behavior with respect to a sample, and posterior updating shown. Dotted arrows show data flow.



The target word is in the English dictionary and has exactly five distinct letters. The target word is equally likely to be any five-letter word in the English dictionary without repeated letters, except that it cannot include the letter 'C', 'A', or 'N', as these letters do not appear anywhere in the target word. The letter 'R' is somewhere in the target word but not at position 2. The letter 'E' is somewhere in the target word but not at position 5. The agent receives a reward of one if the guessed word correctly matches the target word. Otherwise, rewards will always be zero.

We already know positions 2, 4, and 5: _ I _ E R. We need to find suitable letters for positions 1 and 3 from the remaining alphabet that haven't been excluded.

Let's consider the word "MIXER":
- It is 5 letters long.
- It has no repeated letters.
- It does not contain 'C', 'A', 'N', 'T', 'G', 'F', or 'B'.
- 'R' is in position 5.
- 'E' is in position 4.
- 'I' is in position 2.

Figure 2. Examples of a posterior (top) and posterior sample (bottom) generated by our LLM-based PSRL in Wordle

reaping the benefits of uncertainty-based exploration in a computationally-tractable manner is through Posterior Sampling for RL (PSRL) (Strens, 2000), presented as Algorithm 1. Rather than updating the epistemic state at each timestep, PSRL holds it fixed during each episode and only updates posterior beliefs at the end using the full trajectory τ_k. To govern action selection within each episode based on current knowledge of the true underlying MDP $\mathbb{P}(\mathcal{M} \in \cdot \mid H_k)$, PSRL employs Thompson sampling (TS) (Thompson, 1933; Russo & Van Roy, 2014; Russo et al., 2018), whereby the agent draws one posterior sample as a statistically-plausible hypothesis about the true MDP (Line 3) and proceeds to act optimally with respect to it by executing the sampled MDP optimal policy (Lines 4-5). It has been shown theoretically that, by iteratively employing TS in this manner, PSRL is able to achieve strong exploration and satisfy Bayesian regret upper bounds for statistically-efficient RL in tabular MDPs and beyond (Osband et al., 2013; Osband & Van Roy, 2014; Abbasi-Yadkori & Szepesvari, 2014; Osband & Van Roy, 2016; Agrawal & Jia, 2017; Ouyang et al., 2017; Osband & Van Roy, 2017; Lu & Van Roy, 2019).

While PSRL enjoys nice theoretical guarantees, practical implementations extending beyond tabular MDPs (Osband et al., 2013) face significant computational hurdles. Representing and maintaining epistemic uncertainty about the underlying MDP transition and reward functions is an open challenge in high-dimensional environments. While some work has studied using neural networks to address the broader problem of uncertainty estimation for guiding exploration in RL (Osband et al., 2016a; Lu & Van Roy, 2017; Osband et al., 2018; O'Donoghue et al., 2018; Dwaracherla et al., 2020; Osband et al., 2023; Sasso et al., 2023), the overwhelming majority of these efforts have concentrated on a model-free analogue of PSRL that maintains a Bayesian posterior over the optimal action-value function $Q^{\star}$ (Osband et al., 2016b; 2019) in lieu of the underlying MDP $\mathcal{M}$. Meanwhile, the minority of such methods that actually strive to implement PSRL have either been met with mixed results across hard-exploration problems or have been limited to evaluations in smaller-scale domains. Among them is a line of work that leans heavily into the use of Langevin dynamics for recovering the strategic exploration of PSRL (Mazumdar et al., 2020; Karbasi et al., 2023; Ishfaq et al., 2024; Jorge et al., 2024); in the context of this work, such technical machinery is incredibly challenging and nontrivial to combine or emulate with LLM agents.

In parallel, beyond the difficulties of maintaining a PSRL agent's posterior distribution over the true MDP, computing the optimal policy for the posterior sample drawn in each episode constitutes an additional challenge that requires the solution to a model-based planning problem. While there has been progress and even notable successes in this space for deep RL agents (Kaiser et al., 2020), it is unclear if those methods are readily applicable to the natural language tasks faced by LLM agents. In our experiments, we report positive results for our LLM-based PSRL implementation in MDPs with deterministic transition functions, but encounter negative results in environments due to poor LLM planning capability under stochastic dynamics.

3.2. A LLM Implementation

The key contribution of this paper is recognizing that LLMs can be operationalized to provide basic, atomic functions from which PSRL may be implemented. As discussed in Section A, this stands in stark contrast to existing strides towards efficient decision-making with LLM agents (Nie et al., 2024; Krishnamurthy et al., 2024; Klissarov et al., 2024; Ke et al., 2024) which either leave a LLM to its own devices for strategizing exploration or expect in-context learning (ICL) (Brown et al., 2020) to emulate the exploration of an existing RL or bandit algorithm. While future LLMs may become sufficiently capable to accommodate the former, our experiments today suggest this is not the case for two simple natural language tasks where efficient exploration is paramount to success; by the same token, we anticipate that our proposed LLM-based implementation of PSRL will also benefit and gracefully extend to more complex natural language tasks as the constituent LLM models become more capable at performing their requested functions. LLM agents emulating classic RL methods are also bound to the same traditional problem classes whereas LLM-based implementations of RL algorithms may broaden the footprint of classic algorithms to include natural language problem domains that would otherwise be entirely infeasible.

As shown in Algorithm 1, our proposed implementation of PSRL relies on LLMs to play three distinct roles: (1) an approximate posterior updater, (2) a posterior sampler, and (3) an optimal policy with respect to a posterior sample. PSRL requires a prior distribution over MDPs as input and, more generally in any episode, needs a current posterior that accurately reflects the agent's current knowledge *and* uncertainty about the world. For our purposes, such an approximate "posterior"[1] is a textual description that summarizes both the known and uncertain aspects of the true MDP transition and reward function. More importantly, it also explicitly communicates (in some way) the amount of uncertainty an agent has about these aspects of the world. As this textual summary amounts to the PSRL agent's epistemic state representation (Lu et al., 2023), an agent designer may exert strong influence over this representation through the presentation and expression of prior knowledge; as a concrete example, specifying the next-state transition distribution of a tabular MDP in our experiments as a Dirichlet distribution (in language) naturally encourages the LLM-based implementation of PSRL to maintain visitation counts. Of course, an advantage is that agent designers may now leverage the full expressivity and fluidity of natural language for communicating prior knowledge without being restricted to the few statistical distributions allowing the computational conveniences of conjugate priors.

Given a current posterior reflecting the agent's knowledge and uncertainty about the world, PSRL must be able to draw one posterior sample from these beliefs. We implement this as a first LLM that, given the agent's current textual posterior (initially set to be the agent designer's input prior) is tasked with generating a plausible hypothesis for how transitions and rewards unfold. In some domains, such as tabular MDPs, it may be natural for this to be an exhaustive list of rewards and next-state transitions for each state-action pair. For more practical scenarios of interest, it may be beneficial to prompt this posterior sampling LLM so that it can leverage an environment proxy or lossy surrogate MDP (Lu et al., 2023; Arumugam & Van Roy, 2022) that retains only the salient details needed to determine optimal behavior. As a concrete example, one of our natural language tasks is the game of Wordle (shown in Figure 2) that, as a MDP, has a transition function and reward function defined entirely around an unknown, five-letter target word. Here the target word serves as an environment proxy that our LLM-based PSRL agent may directly monitor uncertainty over without meticulously maintaining statistics for individual state-action pairs.

With a single posterior sample in hand, a PSRL agent must be able to select actions that would be considered optimal if the sampled MDP truly reflected reality. We implement this as a second LLM tasked with executing actions given the current state that maximize value in a way that is consistent with the natural language hypothesis generated by the posterior sampling LLM. In the simplest case, the model need only be given the posterior sample and input observation and asked directly to generate an action. In more challenging settings, an agent designer may architect this optimal policy LLM more carefully via chain-of-thought prompting (Wei et al., 2022; Kojima et al., 2022) to increase the chance of selecting optimal actions consistent with provided hypothesis.

Upon the completion of an episode with the optimal policy LLM acting with respect to the hypothesis of the posterior sampling LLM, we task a third and final LLM with updating the PSRL agent's knowledge and residual uncertainty about the world, akin to an (approximate) posterior update. Given a complete trajectory consisting of reward signals and next-state transitions for exactly H state-action pairs, this posterior LLM must reconcile the agent's prior knowledge at the start of the episode against observed interactions from within the environment. All three LLMs can then be orchestrated to run the PSRL algorithm.

4. Experiments

The goal of our experiments is to assess how implementing PSRL with LLMs both retains the desirable exploration properties that PSRL exhibits empirically within simpler problem domains as well as expands the range of problems where these benefits can be realized. To this end, we focus our evaluation on tasks which demand prudent exploration to achieve success. For each task, we present cumulative regret curves (lower, flatter plots indicate better performance) where any shading denotes one standard error. All agents use GPT-4o (Hurst et al., 2024) for their constituent LLMs. We let κ_{sampling}, $\kappa_{\pi^\star}$, and $\kappa_{\text{posterior}}$ denote the temperatures of the posterior sampling, optimal policy, and posterior update LLMs, respectively. We defer further details of our experiments, including all prompts used in each task to the Appendix.

[1]For ease of exposition, we will refer to this object as a posterior throughout the remainder of the paper, but acknowledge the distinction between this object and the true, statistical object that is the Bayesian posterior distribution.

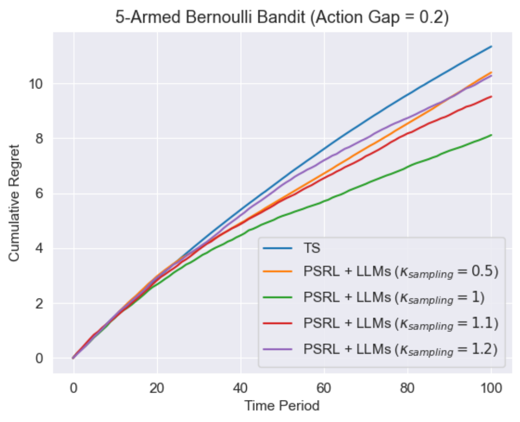


Figure 3. Cumulative regret curves for a 5-armed Bernoulli bandit.

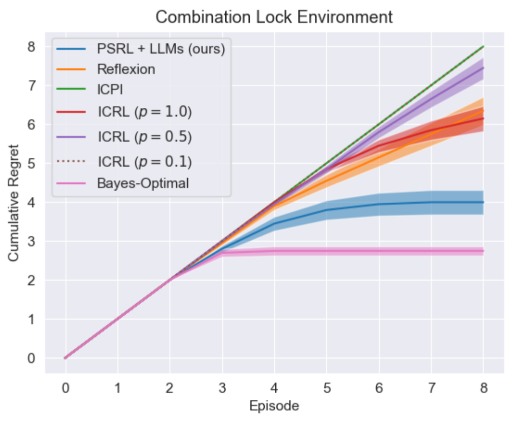


Figure 4. Cumulative regret curves for the combination lock environment. The vertical axis shows turns to identify the unlock code.

4.1. Multi-Armed Bandits

Following prior work studying the exploratory capabilities of LLMs (Coda-Forno et al., 2023; Binz & Schulz, 2023; Coda-Forno et al., 2024; Krishnamurthy et al., 2024; Nie et al., 2024), we begin the empirical assessment of our LLM-based PSRL with a multi-armed bandit problem (Lai & Robbins, 1985; Bubeck & Cesa-Bianchi, 2012; Lattimore & Szepesvári, 2020). Readers unfamiliar with multi-armed bandits may simply observe them as a special case of a MDP with a horizon $H = 1$, singleton state space $|\mathcal{S}| = 1$, and a stochastic (rather than deterministic) reward function. Our evaluation follows that of Krishnamurthy et al. (2024) who chose the simple yet challenging case of a five-armed Bernoulli bandit with independent arms and an action gap of 0.2.[2] The version we evaluate has one randomly selected optimal arm with rewards drawn from a Bernoulli(0.6) distribution while all other arms use a Bernoulli(0.4).

Observe that PSRL specialized to a multi-armed bandit problem mirrors classic TS where, at each timestep, the agent samples one plausible hypothesis for the mean reward of each arm and then proceeds to select the optimal action believed to achieve highest mean reward under the given hypothesis. We compare PSRL implemented with LLMs to classic TS for a Bernoulli bandit with each arm initialized with a Beta$(1, 1)$ prior. Meanwhile, our LLM-based PSRL agent begins with a prior for each arm specified as a `Beta(1,1)` in natural language. While we fix temperatures $\kappa_{\pi^\star} = \kappa_{\text{posterior}} = 1$, we find that the posterior sampling temperature has profound impact on the performance of our LLM-based PSRL agent. Figure 3 compares TS (run for 1,000 independent trials) against PSRL with four distinct settings of κ_{sampling} (run for 20 independent trials). We relegate further bandit results to the Appendix.

[2]The action gap is defined as the difference in expected reward between the best and second best action. Larger action gaps make it easier to identify the optimal arm with few samples whereas smaller action gaps demand greater exploration.

4.2. Natural Language Tasks

An LLM-based implementation of PSRL can help realize the benefits of efficient exploration in domains currently untouchable by vanilla PSRL. We present two natural language tasks where the initial prior uncertainty and time sensitivity due to limited episodes present a formidable exploration challenge.

The first task is a combination lock environment where an agent must enter $H = 3$ distinct digits in order to open a lock and receive a reward of $+1$. All other rewards are zero and the agent is provided with state information indicating whether the most recently guessed digit is either in the correct position for the unlocking code, present in the unlocking code but in some other position, or simply not present in the unlocking code at all. An agent has a total of $K = 8$ episodes to identify the correct combination and, with each one of 20 independent trials having an unlock code sampled uniformly at random from all 720 possible codes, exploration via uniform random action selection has just under a 0.14% chance of success.

The second task is the web game Wordle, where an agent has exactly $K = 6$ episodes to enter $H = 5$ distinct letters[3] that form a correct target word and receive a reward of $+1$. Across 40 independent trials, the target word is chosen uniformly at random from a corpus of English dictionary words filtered for slang and repeated letters. The agent is provided feedback in each state indicating whether the most recently guessed letter is in the correct position for the target word, in the target word but at some other position, or not

[3]We do not require the letters to form a dictionary word.

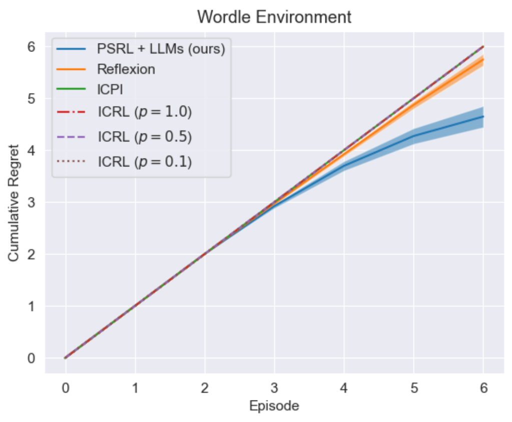


Figure 5. Cumulative regret curve for the Wordle environment. The vertical axis shows turns to identify the target word.

present in the target word at all.

We compare our LLM-based implementation of PSRL against three baseline methods. Our PSRL agent ($\kappa_{\text{sampling}} = \kappa_{\pi^\star} = \kappa_{\text{posterior}} = 1$) is given an uninformative prior which describes all non-repeating codes/English words with the appropriate length as being equiprobable; the unlock code/target word is an environment proxy (Lu et al., 2023) such that knowledge of the proxy is a sufficient statistic for recovering the full MDP. In-Context Policy Iteration (ICPI) (Brooks et al., 2023) takes classic policy iteration (Howard, 1960) and offers an implementation via three LLMs, using ICL to elicit a rollout policy; transition function; and reward function respectively. Together, these models allow for policy improvement via greedy action selection $\pi^{(k)}(s_h) = \arg\max_{a \in \mathcal{A}} Q^{\pi^{(k-1)}}_{\mathcal{M}}(s_h, a)$, with ties broken randomly.[4] In-Context RL (ICRL) (Monea et al., 2024) aims to explore via the stochasticity in LLM responses from sensitivity to the input ICL data. Which episodes are included from a replay buffer for ICL with a LLM policy at each timestep is determined by sampling independent Bernoulli(p) random variables; we study three distinct values of the keep probability $p \in \{1, 0.5, 0.1\}$. Finally, we evaluate Reflexion (Shinn et al., 2024), which passes each full trajectory through a self-reflection LLM that generates verbal guidance; the total history of verbal guidance is given at each timestep to the LLM policy, along with the current state, for improving the quality of decision-making. In the combination lock environment, we also compute the Bayes-optimal policy with respect to the uninformative prior and plot its cumulative regret for comparison with PSRL.

[4]Due to its significantly higher financial cost and lengthy run times, ICPI is limited to only 10 trials in Wordle.

5. Discussion

In this section, we provide a detailed overview of our results as well as insight into the limitations of our proposed LLM-based implementation of PSRL. Due to space constraints, we present a survey of related work in Appendix A.

5.1. Retaining Efficient Exploration

In the bandit setting, we observe our LLM-based PSRL obtains better cumulative regret curves than classic TS, for the limited time horizon of $T = 100$. We find that supplying PSRL with an initial prior of `Beta(1,1)` in language automatically encourages the posterior update LLM to update binary reward observation counts for the chosen arm in each time period. Moreover, we find that the optimal policy LLM has little difficulty in examining the sequence of expected reward values for each arm generated by the posterior sampling LLM and adhering to select the perceived best action. Manipulating κ_{sampling} shows that even values as large as 1 lead to greedy-like exploration in many trials where the resulting posterior sample favors the action observed to yield the most successes thus far. For a limited number of trials, this error proves to be not so catastrophic for temperatures of at least 1. We find that increasing $\kappa_{\text{sampling}} > 1$ yields exploratory behavior more aligned with TS where optimal actions more likely to be taken in the later time periods and a slowing of probability mass pulled away from other actions.

The combination lock and Wordle environments represent separate instances of the same exploration problem within a deterministic environment. Our results show that the LLM-based PSRL is able to most effectively explore the space of possible unlock codes/target words relative to the baseline methods. Crucially, none of the three constituent LLMs used by PSRL are prompted to explicitly encourage exploration. Rather, these results illustrate how prompting these LLMs to perform atomic functions of PSRL and allowing the algorithm to prescribe how those outputs should be orchestrated in the agent design can yield an effective exploration strategy.

The ICPI paper includes a dataset balancing scheme for ICL, presuming the requisite data has already been collected. While reasonable for some environments, exploration is fundamentally about governing data collection to synthesize optimal behavior and, in these domains, ICPI never observes non-zero reward and collapses to a random policy For ICRL, using all available data with $p = 1$ is equivalent to the "LLM policy" evaluated by Klissarov et al. (2024), who also find poor performance in Wordle. While results in the combination lock domain are better, we find that decreasing the keep probability p is detrimental to the "exploratory" ICRL of Monea et al. (2024). Reflexion is the strongest baseline, however we observe that self-reflections during the early stages of learning explicitly encourage exploration

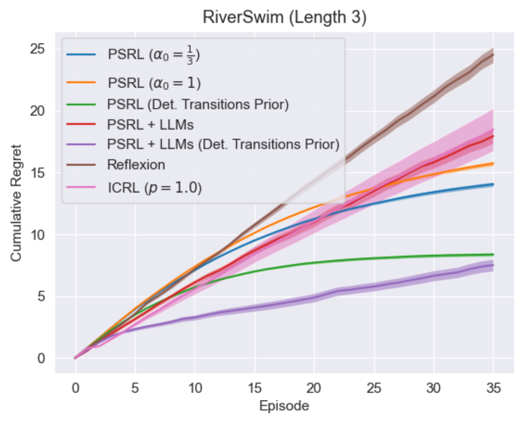


Figure 6. Cumulative regret curve for the RiverSwim environment with 3 states. Algorithms with knowledge of all deterministic transitions supplied *a priori* are labeled.

of untested digits/letters *literally*, assuming the agent knows how to explore upon simply being instructed to do so. Only once uncertainty has largely been resolved do reflections become more specific suggestions about how to explore with particular digits/letters and their ordering.

5.2. Limitations

5.2.1. Stochastic Transition Dynamics

While the domains presented in the previous section confirm that a LLM-based implementation of PSRL retains efficient exploration in deterministic environments, we find that it falls short in stochastic environments. As a simple illustration of this, we turn our focus to a truncated variant of the RiverSwim environment (Strehl & Littman, 2008). RiverSwim is a tabular MDP given as a six-state chain where the agent begins in the leftmost state. The stochastic transition function mimics a water current that allows an agent to deterministically swim to the left (downstream with the current) but only stochastically swim to the right (upstream against the current) with roughly a 40% chance of success.[5] Swimming downstream in the initial state results in a small reward of 0.005. Successfully swimming all the way upstream allows the agent to reach the rightmost state where it can collect a reward of 1. As all other rewards are zero, a RiverSwim agent must explore the full length of the river to learn optimal behavior. To keep financial and temporal costs down, we truncate the environment to a river of length 3 (one initial state, intermediate state, and terminal state).

We compare our LLM-based implementation of PSRL with a vanilla PSRL agent for a tabular MDP (Osband et al., 2013). The latter models epistemic uncertainty over the transition function as a collection of $|\mathcal{S}||\mathcal{A}|$ Dirichlet distributions. This epistemic state representation allows for the computational conveniences of Dirichlet-multinomial conjugacy. We further model unknown rewards with a discrete uniform prior over $\{0, 0.005, 1\}$. Cumulative regret curves shown in Figure 6 compare our LLM-based PSRL with a `Dirichlet(0.1,0.1,0.1)` prior against vanilla PSRL run with two different choices of (uniform) Dirichlet prior initialization ($\alpha_0 = \frac{1}{|\mathcal{S}|}$ and $\alpha_0 = 1$). Additional comparisons are made against our best-performing baselines in the combination lock environment: Reflexion and ICRL with $p = 1$. We also report both vanilla and LLM-based PSRL run with prior distributions where all deterministic RiverSwim transitions (only those where the agent swims downstream) are given as prior knowledge. We use $\kappa_{\pi^\star} = \kappa_{\text{posterior}} = 0$ and $\kappa_{\text{sampling}} = 0.5$. All agents are run for 40 independent trials.

Despite achieving the best regret curve out of all presented LLM agents in RiverSwim, both of our LLM-based PSRL variants incur near-linear regret while most instances of classic PSRL are able to achieve optimal behavior. Reflexion is unable to persevere past failed attempts to swim upstream before settling for the smaller downstream reward of 0.005. ICRL has just over 25% of trials where it stumbles into the optimal policy and sticks with it while, for 60% of trials, it too falls back to pursuing the downstream reward. For PSRL, this result underscores a crucial distinction in the choice of epistemic state between agents; that is, the statistical object Dirichlet$(0.1, 0.1, 0.1)$ used by classic PSRL and the natural language string `Dirichlet(0.1,0.1,0.1)` used in LLM-based PSRL. For deterministic transitions in RiverSwim, classic PSRL is able to see eventual concentration to a Dirac delta distribution. Meanwhile the LLM-based PSRL agent, while successful at maintaining visitation counts, struggles to achieve the same convergence and leaves non-negligible probability mass in each posterior sample on non-existent transitions with fictitious rewards. One plausible explanation would be that such concentration errors stem from a lack of familiarity by the LLMs, given that Dirichlet distributions with fractional parameters are encountered with less frequency (McCoy et al., 2024); however, our preliminary experiments with a `Dirichlet(1,1,1)` prior showed no significant improvement.

We posited that supplying all deterministic transitions as prior knowledge would fare better against classic PSRL. While this does allow LLM-based PSRL to exhibit optimal behavior in many trials, far too many still fail as the optimal policy LLM struggles to select optimal actions, even when supplied with posterior samples that have high fidelity to the true environment. Reasons for this include misread transition probabilities (such as swapping numerical values of the input posterior sample) as well as a lack of understanding for

[5] We adhere to the specific transition dynamics presented by Osband et al. (2013).

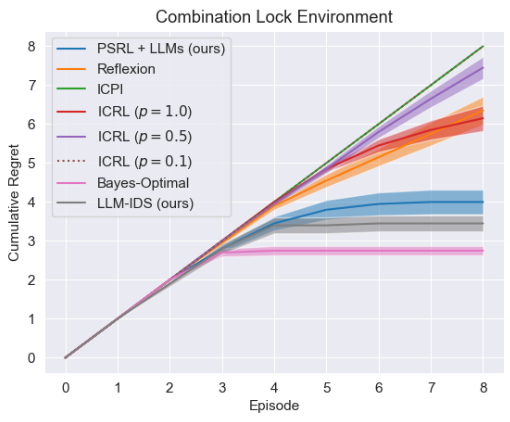


Figure 7. Cumulative regret curves for the combination lock environment including LLM-IDS.

long-term, value-based planning. Additionally, we observe a rare occurrence where posterior updates can be prone to catastrophically forgetting a single transition and halting learning progress. Altogether, while the overall result is negative, we anticipate that the issues we have encountered (posterior concentration, holistic long-term planning, and forgetting) may be alleviated organically with possibly no intervention beyond swapping the GPT-4o model used in our experiments with a more advanced alternative (Jaech et al., 2024; Guo et al., 2025). If not, one might still naturally anticipate that such deficiencies will disappear with time assuming future LLM capabilities continue to grow.

5.2.2. Beyond Thompson Sampling

While PSRL, through the use of TS, is known to yield a strong exploration strategy, it is by no means perfect. In the bandit literature, shortcomings of TS are well-known and naturally become more salient in the full RL problem (Russo & Van Roy, 2018; Lu et al., 2023). In short, by only executing actions with some probability of being optimal, TS will never take deliberately sub-optimal actions that yield tremendous information gain. Figure 2 already illustrates how a PSRL agent's uncompromising execution of only potentially-optimal policies cripples exploration and only allows for the testing of two unknown letters at a time.

One remedy is to seek out instantiations of information-directed sampling (IDS) (Russo & Van Roy, 2018). IDS is an algorithmic design principle that advocates for using a policy which balances between performance shortfall and information gain. While supported by a rigorous corroborating theory in both bandits and RL (Lu et al., 2023), concrete and practical instantiations of IDS are difficult to come by on account of the challenges surrounding information gain estimation (McAllester & Stratos, 2020). Moreover, the temporally-delayed consequences absent from bandits but present in RL problems poses an additional challenge as a proper IDS agent must forecast future opportunities for knowledge acquisition several steps into the future when evaluating current actions.

We present an initial design for a IDS agent with LLMs. Our proposed LLM-IDS agent is myopic in that it only takes immediate information gain about optimal behavior at the next timestep into account. Nevertheless, the feedback structure of the combination lock environment allows such an agent to be unconcerned with temporally-delayed information. For a current state $s_t \in \mathcal{S}$, we define two $|\mathcal{A}|$-dimensional vectors, ρ and $\mathcal{I}$, where $\rho(a) = \mathbb{E}\left[V^\star(s_t) - Q^\star(s_t, a)\right]$ is the expected regret of taking action $a \in \mathcal{A}$ in s_t under the agent's current posterior and $\mathcal{I}(a) = \mathbb{I}(\pi^\star; R_t, S_{t+1} \mid A_t = a, S_t = s_t)$ is the information gained (formally, the conditional mutual information (Cover & Thomas, 2012)) about the optimal policy by taking action a from state s_t. IDS calls for sampling an action from the distribution that minimizes the information ratio: $\min\limits_{\pi \in \Delta(\mathcal{A})} \frac{\mathbb{E}_{a \sim \pi}[\rho(a)]^2}{\mathbb{E}_{a \sim \pi}[\mathcal{I}(a)]}$. Normally, computation of the ρ and $\mathcal{I}$ vectors would be done directly with the current posterior. Instead, we recycle the same posterior update LLM from our LLM-based PSRL but incorporate two new LLMs for the provision of ρ and $\mathcal{I}$; each of these LLMs is prompted on a per-action basis to assess the expected regret or information gain, respectively, from each action in the current state. With these $2|\mathcal{A}|$ LLM-generated numerical values, the convex optimization problem of minimizing the information ratio is solved to compute the policy for action selection. Figure 7 shows that LLM-IDS is able to outperform LLM-based PSRL by more quickly testing for unknown digits while remaining unencumbered by known digits already discovered.

6. Conclusion

While much of the burgeoning literature surrounding LLM agents has felt compelled to design new algorithms for solving RL problems, we here have demonstrated that an existing algorithm, PSRL, can be implemented with LLMs in environments with deterministic transition dynamics. The main advantage of our proposed LLM-based implementation of PSRL is allowing agent designers to leverage the strong generalization and reasoning capabilities of LLMs in natural-language environments while simultaneously capitalizing on the well-studied exploration properties of TS. Further study on how LLMs may perform the requisite planning needed to extend our results to stochastic environments is a natural area for future work. More broadly, our preliminary results with recovering information-directed exploration with LLMs represents a fruitful direction and further reinforces the potential benefits of implementing, rather than replacing, existing RL algorithms with LLMs.

Impact Statement

The impact of LLMs in recent years has been undeniable and so immense as to extend beyond the confines of the machine learning community, drawing scrutiny from the broader public. As this paper studies mechanisms for improving the decision-making capabilities of LLMs that are becoming increasingly more capable and ubiquitously deployed, there is potential for broad impact stemming from our work. This impact is amplified by the fact that our contributions for improved exploration in LLMs center around Thompson sampling (Thompson, 1933), an exploration strategy whose impact in real-world decision-making problems such as recommendation systems (Chapelle & Li, 2011) and beyond (Russo et al., 2018) is already well known.

References

Abbasi-Yadkori, Y. and Szepesvari, C. Bayesian Optimal Control of Smoothly Parameterized Systems: The Lazy Posterior Sampling Algorithm. *arXiv preprint arXiv:1406.3926*, 2014.

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

Agrawal, S. and Jia, R. Optimistic Posterior Sampling for Reinforcement Learning: Worst-Case Regret Bounds. In *Advances in Neural Information Processing Systems*, pp. 1184–1194, 2017.

Arumugam, D. and Singh, S. Planning to the Information Horizon of BAMDPs via Epistemic State Abstraction. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Arumugam, D. and Van Roy, B. Deciding What to Model: Value-Equivalent Sampling for Reinforcement Learning. *Advances in Neural Information Processing Systems*, 35: 9024–9044, 2022.

Auer, P., Fischer, P., and Cesa-Bianchi, N. Finite-Time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(3):235–256, 2002.

Auer, P., Jaksch, T., and Ortner, R. Near-Optimal Regret Bounds for Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pp. 89–96, 2009.

Azar, M. G., Osband, I., and Munos, R. Minimax Regret Bounds for Reinforcement Learning. In *International Conference on Machine Learning*, pp. 263–272, 2017.

Bellman, R. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

Bellman, R. and Kalaba, R. On Adaptive Control Processes. *IRE Transactions on Automatic Control*, 4(2):1–9, 1959.

Binz, M. and Schulz, E. Using Cognitive Psychology to Understand GPT-3. *Proceedings of the National Academy of Sciences*, 120(6):e2218523120, 2023.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the Opportunities and Risks of Foundation Models. *arXiv preprint arXiv:2108.07258*, 2021.

Brafman, R. I. and Tennenholtz, M. R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

Brooks, E., Walls, L., Lewis, R. L., and Singh, S. Large Language Models Can Implement Policy Iteration. *Advances in Neural Information Processing Systems*, 36: 30349–30366, 2023.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33: 1877–1901, 2020.

Bubeck, S. and Cesa-Bianchi, N. Regret Analysis of Stochastic and Nonstochastic Multi-Armed Bandit Problems. *Foundations and Trends in Machine Learning*, 5 (1):1–122, 2012.

Chapelle, O. and Li, L. An Empirical Evaluation of Thompson Sampling. In *Advances in Neural Information Processing Systems*, pp. 2249–2257, 2011.

Coda-Forno, J., Binz, M., Akata, Z., Botvinick, M., Wang, J., and Schulz, E. Meta-In-Context Learning in Large Language Models. *Advances in Neural Information Processing Systems*, 36:65189–65201, 2023.

Coda-Forno, J., Binz, M., Wang, J. X., and Schulz, E. CogBench: A Large Language Model Walks into a Psychology Lab. In *Forty-first International Conference on Machine Learning*, 2024.

Cover, T. M. and Thomas, J. A. *Elements of Information Theory*. John Wiley & Sons, 2012.

Dai, Z., Tomasi, F., and Ghiassian, S. In-Context Exploration-Exploitation for Reinforcement Learning. In *The Twelfth International Conference on Learning Representations*, 2024.

Dann, C. and Brunskill, E. Sample Complexity of Episodic Fixed-Horizon Reinforcement Learning. In *Proceedings* *of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pp. 2818–2826, 2015.

Dann, C., Lattimore, T., and Brunskill, E. Unifying PAC and Regret: Uniform PAC Bounds for Episodic Reinforcement Learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 5717–5727, 2017.

Der Kiureghian, A. and Ditlevsen, O. Aleatory or Epistemic? Does it Matter? *Structural Safety*, 31(2):105–112, 2009.

Dong, S., Van Roy, B., and Zhou, Z. Simple Agent, Complex Environment: Efficient Reinforcement Learning with Agent States. *Journal of Machine Learning Research*, 23(255):1–54, 2022.

Dudík, M., Hofmann, K., Schapire, R. E., Slivkins, A., and Zoghi, M. Contextual Dueling Bandits. In *Conference on Learning Theory*, pp. 563–587, 2015.

Duff, M. O. *Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst, 2002.

Dwaracherla, V., Lu, X., Ibrahimi, M., Osband, I., Wen, Z., and Van Roy, B. Hypermodels for Exploration. In *International Conference on Learning Representations*, 2020.

Dwaracherla, V., Asghari, S. M., Hao, B., and Van Roy, B. Efficient Exploration for LLMs. In *Forty-first International Conference on Machine Learning*, 2024.

Ghavamzadeh, M., Mannor, S., Pineau, J., and Tamar, A. Bayesian Reinforcement Learning: A Survey. *Foundations and Trends in Machine Learning*, 8(5-6):359–483, 2015.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*, 2025.

Howard, R. A. Dynamic Programming and Markov Processes. *MIT Press*, 1960.

Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al. GPT-4o System Card. *arXiv preprint arXiv:2410.21276*, 2024.

Ishfaq, H., Lan, Q., Xu, P., Mahmood, A. R., Precup, D., Anandkumar, A., and Azizzadenesheli, K. Provable and Practical: Efficient Exploration in Reinforcement Learning via Langevin Monte Carlo. In *The Twelfth International Conference on Learning Representations*, 2024.

Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., et al. OpenAI o1 System Card. *arXiv preprint arXiv:2412.16720*, 2024.

Jaksch, T., Ortner, R., and Auer, P. Near-Optimal Regret Bounds for Reinforcement Learning. *Journal of Machine Learning Research*, 11(4), 2010.

Jin, C., Allen-Zhu, Z., Bubeck, S., and Jordan, M. I. Is Q-Learning Provably Efficient? *Advances in Neural Information Processing Systems*, 31, 2018.

Jorge, E., Dimitrakakis, C., and Basu, D. Isoperimetry is All We Need: Langevin Posterior Sampling for RL with Sublinear Regret. *arXiv preprint arXiv:2412.20824*, 2024.

Kaiser, Ł., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. Model Based Reinforcement Learning for Atari. In *International Conference on Learning Representations*, 2020.

Kakade, S. M. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University of London, University College London (United Kingdom), 2003.

Karbasi, A., Kuang, N. L., Ma, Y., and Mitra, S. Langevin Thompson Sampling with Logarithmic Communication: Bandits and Reinforcement Learning. In *International Conference on Machine Learning*, pp. 15828–15860, 2023.

Ke, N. R., Sawyer, D. P., Soyer, H., Engelcke, M., Reichert, D. P., Hudson, D. A., Reid, J., Lerchner, A., Rezende, D. J., Lillicrap, T. P., Mozer, M., and Wang, J. X. Can Foundation Models actively Gather Information in Interactive Environments to Test Hypotheses? *arXiv preprint arXiv:2412.06438*, 2024.

Kearns, M. and Singh, S. Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, 49: 209–232, 2002.

Klissarov, M., Hjelm, D., Toshev, A., and Mazoure, B. On the Modeling Capabilities of Large Language Models for Sequential Decision Making. *arXiv preprint arXiv:2410.05656*, 2024.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large Language Models are Zero-Shot Reasoners. *Advances in Neural Information Processing Systems*, 35: 22199–22213, 2022.

Krishnamurthy, A., Harris, K., Foster, D. J., Zhang, C., and Slivkins, A. Can Large Language Models Explore In-Context? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Kwon, M., Xie, S. M., Bullard, K., and Sadigh, D. Reward design with language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Lai, T. L. and Robbins, H. Asymptotically Efficient Adaptive Allocation Rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

Laskin, M., Wang, L., Oh, J., Parisotto, E., Spencer, S., Steigerwald, R., Strouse, D., Hansen, S., Filos, A., Brooks, E., et al. In-Context Reinforcement Learning with Algorithm Distillation. *arXiv preprint arXiv:2210.14215*, 2022.

Lattimore, T. and Szepesvári, C. *Bandit Algorithms*. Cambridge University Press, 2020.

Lee, J., Xie, A., Pacchiano, A., Chandak, Y., Finn, C., Nachum, O., and Brunskill, E. Supervised Pretraining can Learn In-Context Reinforcement Learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Lu, X. and Van Roy, B. Ensemble Sampling. *Advances in Neural Information Processing Systems*, 30, 2017.

Lu, X. and Van Roy, B. Information-Theoretic Confidence Bounds for Reinforcement Learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Lu, X., Van Roy, B., Dwaracherla, V., Ibrahimi, M., Osband, I., and Wen, Z. Reinforcement Learning, Bit by Bit. *Foundations and Trends in Machine Learning*, 16(6):733–865, 2023.

Mazumdar, E., Pacchiano, A., Ma, Y., Jordan, M., and Bartlett, P. On Approximate Thompson Sampling with Langevin Algorithms. In *International Conference on Machine Learning*, pp. 6797–6807, 2020.

McAllester, D. and Stratos, K. Formal Limitations on the Measurement of Mutual Information. In *International Conference on Artificial Intelligence and Statistics*, pp. 875–884, 2020.

McCoy, R. T., Yao, S., Friedman, D., Hardy, M. D., and Griffiths, T. L. Embers of Autoregression Show how Large Language Models are Shaped by the Problem They are Trained to Solve. *Proceedings of the National Academy of Sciences*, 121(41):e2322420121, 2024.

Monea, G., Bosselut, A., Brantley, K., and Artzi, Y. LLMs Are In-Context Reinforcement Learners. *arXiv preprint arXiv:2410.05362*, 2024.

Nie, A., Su, Y., Chang, B., Lee, J. N., Chi, E. H., Le, Q. V., and Chen, M. EVOLvE: Evaluating and Optimizing LLMs For Exploration. *arXiv preprint arXiv:2410.06238*, 2024.

O'Donoghue, B., Osband, I., Munos, R., and Mnih, V. The Uncertainty Bellman Equation and Exploration. In *International Conference on Machine Learning*, pp. 3836–3845, 2018.

Osband, I. Risk Versus Uncertainty in Deep Learning: Bayes, Bootstrap and the dangers of Dropout. In *NIPS Workshop on Bayesian Deep Learning*, 2016.

Osband, I. and Van Roy, B. Model-Based Reinforcement Learning and the Eluder Dimension. *Advances in Neural Information Processing Systems*, 27, 2014.

Osband, I. and Van Roy, B. Posterior Sampling for Reinforcement Learning Without Episodes. *arXiv preprint arXiv:1608.02731*, 2016.

Osband, I. and Van Roy, B. Why is Posterior Sampling Better than Optimism for Reinforcement Learning? In *International Conference on Machine Learning*, pp. 2701–2710, 2017.

Osband, I., Russo, D., and Van Roy, B. (More) Efficient Reinforcement Learning via Posterior Sampling. *Advances in Neural Information Processing Systems*, 26: 3003–3011, 2013.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep Exploration via Bootstrapped DQN. *Advances in Neural Information Processing Systems*, 29, 2016a.

Osband, I., Van Roy, B., and Wen, Z. Generalization and Exploration via Randomized Value Functions. In *International Conference on Machine Learning*, pp. 2377–2386, 2016b.

Osband, I., Aslanides, J., and Cassirer, A. Randomized Prior Functions for Deep Reinforcement Learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Osband, I., Van Roy, B., Russo, D. J., and Wen, Z. Deep Exploration via Randomized Value Functions. *Journal of Machine Learning Research*, 20(124):1–62, 2019.

Osband, I., Wen, Z., Asghari, S. M., Dwaracherla, V., Ibrahimi, M., Lu, X., and Van Roy, B. Approximate Thompson Sampling via Epistemic Neural Networks. In *Uncertainty in Artificial Intelligence*, pp. 1586–1595, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training Language Models to Follow Instructions with Human Feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Ouyang, Y., Gagrani, M., Nayyar, A., and Jain, R. Learning Unknown Markov Decision Processes: A Thompson

Sampling Approach. *Advances in Neural Information Processing Systems*, 30, 2017.

Puterman, M. L. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New York, 1994.

Russo, D. and Van Roy, B. Learning to Optimize via Posterior Sampling. *Mathematics of Operations Research*, 39 (4):1221–1243, 2014.

Russo, D. and Van Roy, B. Learning to Optimize via Information-Directed Sampling. *Operations Research*, 66(1):230–252, 2018.

Russo, D. J., Van Roy, B., Kazerouni, A., Osband, I., and Wen, Z. A Tutorial on Thompson Sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018.

Sasso, R., Conserva, M., and Rauber, P. Posterior Sampling for Deep Reinforcement Learning. In *International Conference on Machine Learning*, pp. 30042–30061, 2023.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language Agents with Verbal Reinforcement Learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. F. Learning to Summarize with Human Feedback. *Advances in Neural Information Processing Systems*, 33: 3008–3021, 2020.

Strehl, A. L. and Littman, M. L. An Analysis of Model-Based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Strehl, A. L., Li, L., and Littman, M. L. Reinforcement Learning in Finite MDPs: PAC Analysis. *Journal of Machine Learning Research*, 10(11), 2009.

Strens, M. J. A Bayesian Framework for Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 943–950, 2000.

Sutton, R. S. and Barto, A. G. *Introduction to Reinforcement Learning*. MIT Press, 1998.

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. Gemini: A Family of Highly Capable Multimodal Models. *arXiv preprint arXiv:2312.11805*, 2023.

Thompson, W. R. On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25(3/4):285–294, 1933.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*, 2023.

Yue, Y., Broder, J., Kleinberg, R., and Joachims, T. The k-Armed Dueling Bandits Problem. *Journal of Computer and System Sciences*, 78(5):1538–1556, 2012.

Zheng, Q., Henaff, M., Zhang, A., Grover, A., and Amos, B. Online Intrinsic Rewards for Decision Making Agents from Large Language Model Feedback. *arXiv preprint arXiv:2410.23022*, 2024.

A. Related Work

While our primary focus in this paper is on efficient exploration for LLM agents, the broader challenge of efficient exploration for RL agents is a long-studied topic. One route to achieving statistically-efficient RL relies on the use of "optimism in the face of uncertainty," where approaches would either implicitly or explicitly maintain over-inflated value function estimates for all state-action pairs (Kearns & Singh, 2002; Brafman & Tennenholtz, 2002; Kakade, 2003; Auer et al., 2009; Strehl et al., 2009; Jaksch et al., 2010; Dann & Brunskill, 2015; Azar et al., 2017; Dann et al., 2017; Jin et al., 2018; Dong et al., 2022). These optimistic biases would be calibrated by an agent designer to incentivize agent visitation of each state-action pair sufficiently many times and eventually result in accurate value estimates that give rise to optimal behavior. Nie et al. (2024) attempt to realize such an optimistic exploration strategy with LLMs (specifically, combining UCB (Auer et al., 2002) with Gemini) for multi-armed bandit problems and demonstrate the difficulty in coupling statistical machinery like confidence intervals with LLMs outright. While our proposed implementation relies on an equally (if not more) complex statistical object, the Bayesian posterior, our experiments suggest that LLMs in certain cases may maintain an approximation sufficient for guiding exploration. We defer a brief review of prior work on the alternative class of uncertainty-based exploration methods to Section 3.

Existing designs for LLM agents either do not explicitly engage with the challenge of exploration or do so with complete reliance on in-context learning (ICL) (Brown et al., 2020). One of the most popular LLM agent designs is Reflexion (Shinn et al., 2024) where the policy LLM charged with selecting actions is informed at each episode by a "self-reflection" generated from another LLM given the previous episode trajectory. While suitable for some tasks, we observe in our experiments that the self-reflection LLM often "passes the buck" and encourages exploration explicitly in language without providing a clear strategy for the downstream policy LLM to do so. By relying on LLMs to provide the requisite functions for the right choice of existing RL algorithm, we encounter strategic exploration without needing to specifically instruct any of the involved LLMs to explore. LLM agents that rely on ICL to enable exploration follow suit with a line of work that examines Transformer-based RL agents in non-natural-language tasks (Laskin et al., 2022; Lee et al., 2024; Dai et al., 2024). Very close to the spirit of our work is the in-context policy iteration (ICPI) method of Brooks et al. (2023), who take the classic RL algorithm of policy iteration (PI) (Howard, 1960) and implement it with LLMs and ICL. Unfortunately, the original PI algorithm is oriented towards tabular MDPs that allow for iterating over all state-action pairs simultaneously. In larger environments where data must be judiciously acquired, we find that ICPI is never able to collect the data needed for ICL to exhibit anything more that uniform-random action selection. Monea et al. (2024) study a selective "dropout" strategy for the ICL demonstrations used by a policy LLM. Unfortunately, such a strategy mirrors ϵ-greedy exploration without making a concerted effort to strategically guide decision-making, much like how classic dropout in deep RL is a poor proxy for uncertainty-based exploration (Osband, 2016).

A related line of approaches examine using classic (deep) RL methods in tandem with LLM reward functions (Klissarov et al., 2024; Kwon et al., 2023; Zheng et al., 2024). These approaches, while interesting, largely focus on non-linguistic domains while our goal is to bring ideas on data-efficient RL to bear on the natural language domains where LLMs stand to have the most impact. The posterior-sampling-based exploration strategy we consider in this work connects more broadly to initial investigations surrounding the information gathering capabilities of LLMs (Ke et al., 2024). Lastly, we note that the RLHF pipeline (Stiennon et al., 2020; Ouyang et al., 2022) used to explicitly train LLMs also faces an underlying sequential decision-making problem (in the original formulation, a contextual dueling bandit (Yue et al., 2012; Dudík et al., 2015)) and, as such, may greatly benefit from mechanisms to facilitate efficient exploration (Dwaracherla et al., 2024). While such work is nascent, our results offer a promising new pathway for LLMs to achieve the strategic exploration that could reduce the significant data burdens of RLHF.

B. Experiment Prompts

In this section, we outline all LLM prompts used in our experiments. We will present all system prompts in orange and all user prompts in red.

In our experiments, depending on the particular environment, we consider two different forms of posterior LLM prompting. For sufficiently short horizons, the posterior LLM is given the entire trajectory in a single prompt and is expected to produce the updated posterior. For longer horizons or whenever concerns about context buffer length come into play, the posterior LLM is prompted with one full (s, a, r, s') experience tuple at a time and each successive posterior becomes the prior for the subsequent update. Empirically, we find that whole trajectory updates may be more likely to result in erroneous updates where certain pieces of information may be mistakenly updated or forgotten entirely. While this becomes far less likely with per-step experience updates, the associated financial costs and time spent running the PSRL agent scale unfavorably with the horizon of the problem. We use whole trajectory observations for all LLM-based PSRL posterior updates in the RiverSwim, Combination Lock, and Wordle environments. For LLM-based PSRL multi-armed bandit results and LLM-IDS, we use per-step posterior updates.

For whole trajectory posterior updates, the approximate posterior LLM uses the following system prompt and user prompt:

You are a Bayesian posterior distribution for a real-world sequential decision-making problem. Given a current prior belief about the environment and single trajectory observation, you should produce the posterior distribution that accurately reflects knowledge about possibly stochastic environment transitions and environment rewards based on the observed trajectory. A trajectory observation is a sequence of experiences, where each experience consists of a state, action, reward, and next state. Each unit of experience will be separated by XML <EXPERIENCE> </EXPERIENCE> tags. The posterior distribution must always be complete and describe all sources of uncertainty the agent has about the world. There can be uncertainty about a stochastic transition or reward. The posterior distribution should take into account all information provided in the observed trajectory to update the prior belief about the environment. Be direct and don't show your work. You cannot make any assumptions about the agent and the action selections used to generate the trajectory observation. Never try to model beliefs about the agent. Do not say anything beyond providing the posterior distribution. The agent's interactions with the environment will generate rewards and the posterior distribution should keep track of how any and all rewards are generated. Information and knowledge in the current prior belief about the environment should never be discarded from the posterior distribution. If there is knowledge in the current prior belief about the environment that is unaffected by the trajectory observation, then this knowledge should not be changed and must be repeated exactly in the posterior distribution. Do not say anything to distinguish between old knowledge that is being retained and updated knowledge. The environment was described to the agent like this: `<Environment Description>`

Your current prior is as follows: `<Input prior/LLM-generated posterior>`. A trajectory observation is a sequence of experiences, where each experience consists of a state, action, reward, and next state. Each unit of experience will be separated by XML <EXPERIENCE> </EXPERIENCE> tags. Here is an observed trajectory:`<Full trajectory>`. Remember that knowledge in the current prior must only be updated but can never be discarded, forgotten, or removed. Do not say anything about which information in the posterior is new and updated or old and remains the same from the prior.

For per-step posterior updates, the approximate posterior LLM uses the following system prompt and user prompt:

You are a Bayesian posterior distribution generator for a real-world sequential decision-making problem. A sequential decision-making problem is represented by an environment that, to each current state and action, produces a next state transition and a reward based on that transition. Transitions and rewards observed from the environment may be stochastic or may be deterministic. Given a current prior belief about the environment and single observation consisting of a next state transition and reward from the environment, you should generate the posterior distribution that accurately reflects knowledge about possibly stochastic environment transitions and environment rewards. The posterior distribution should be a complete and accurate description of all uncertainty the agent has about the world. Information from the prior belief can never be discarded, only updated to be more consistent with the given observation. The posterior distribution should take into account all information provided in the observed next state transition and reward to update the prior belief about the environment. You cannot make any assumptions about the agent and the action selections used to generate the next state transition and reward observation. Never try to model beliefs about the agent. The world may be stochastic and random such that the prior knowledge may need to be updated in the posterior distribution to be consistent with an observed transition or reward. Any knowledge in the prior belief about the environment that is not affected by the observed transition and reward should be retained in full by your posterior distribution. The environment was described to the agent like this: `<Environment Description>`

Your current prior is as follows:`<Input prior/LLM-generated posterior>`. Here is an observed environment transition and reward:`<Single next-state transition and reward>`. Do not say anything about which information in the posterior is new and updated or old and remains the same from the prior. Whenever possible you must maintain exact, numerical probabilities.

The optimal policy LLM simply takes the current observation as the user prompt while using the following system prompt:

`<Environment Description>`. Always select optimal actions that maximize value across all future states and all remaining timesteps according to the following hypothesis: `<LLM-generated posterior sample>`. You must select actions that are optimal for and perfectly consistent with the above hypothesis. For each action, you must consider its immediate expect reward as well as the expected value of future states that can be visited by selecting the action. Always select from one of the available actions to take in the environment. Just say the action after "Action: " and nothing else.

As generating a posterior sample requires specifying a full MDP, we find that the posterior sampling LLM in PSRL benefits from having distinct prompts that cater to salient aspects of generating an instance of each environment. We organize the associated environment descriptions as well as posterior sampling system prompts and user prompts by task in the following sub-sections. We also include a sub-section for all prompts used by LLM-IDS.

B.1. Multi-Armed Bandits

The environment description for the multi-armed bandit task was given as:

You are an agent interacting with a 5-armed Bernoulli bandit problem. You have exactly 5 actions available labeled as `<List of randomly generated letters>` and each action has an independent Bernoulli distribution. When you select an action, you will receive a binary reward sampled from the associated Bernoulli distribution.

The posterior sampling LLM system prompts and user prompts were:

You are a generator of Bernoulli bandit problems. A Bernoulli bandit problem is a collection of mean reward values, one for each available action. Knowledge about the reward of each available action will be given to you in the form of a Beta distribution representing beliefs about the mean reward of each arm. This knowledge will constrain the Bernoulli bandit problems you are allowed to generate. For each action, return one plausible hypothesis for the mean reward an agent will observe when taking that action. Each mean reward you return should be consistent with the knowledge you are given about the observed rewards of each action. Each action is independent and so each hypothesis you return for the mean reward of each action will be independent of all others. You must return real, numerical values starting with the phrase "You think " and do not say anything beyond providing the mean rewards of each action. You cannot just return the mean value of the Beta distribution as your guess for the mean reward. You must return a sample from each Beta distribution as your hypothesis. Before you return your mean reward values, describe how each one obeys all constraints and knowledge provided to you. The environment was described to the agent like this: `<Environment Description>`

Your current knowledge about the mean reward of each action is as follows:`<Input prior/LLM-generated posterior>`. You must carefully read through this information to generate a Bernoulli bandit problem consistent with this knowledge.

B.2. RiverSwim

The environment description for RiverSwim was given as:

You are an agent swimming in a network of three underwater caves connected by tunnels. Each cave is labeled by its number and always has two tunnels labeled A and B that you can try to swim through. Swimming through tunnels allows you to stochastically move between the caves. There is a strong current in the water which can affect how difficult it is to successfully swim through certain tunnels. Some tunnels may be easier to swim through than others. Successfully swimming through a tunnel once in any cave does not guarantee that it will always be successful. Conversely, failing to swim through a tunnel once does not mean it is impossible and you may have to try again a few times before successfully making it through and swimming into a different cave. Swimming through specific tunnels from certain caves to reach other caves may yield scalar rewards between zero and one.

The posterior sampling LLM system prompts and user prompts were:

You are a map generator for an agent navigating an environment. The environment was described to the agent as follows:`<Environment Description>`. A map must specify exactly two pieces of information for each possible combination of current cave, tunnel, and next cave. The first piece of information is a transition probability that represents the probability of being in a specific cave, swimming through a particular tunnel, and ending up in a specific next cave. Knowledge about next cave transitions will be provided to you as a collection of Dirichlet distributions. Sampling these distributions will allow you to generate next cave transition probabilities for each cave and tunnel combination. The second piece of information is a deterministic reward that an agent will receive when being in a specific cave, swimming through a particular tunnel, and ending up in a specific next cave. You will be given knowledge about known rewards and rewards that are still unknown and uncertain. If a reward is known, you must repeat its numerical value exactly in the map you generate. If a reward is unknown, knowledge about what it could be will be given to you as a discrete uniform distribution over possible values. You will sample this distribution for each cave, tunnel, and next cave combination and include the concrete, numerical reward value in the map you generate. The input knowledge will constrain the maps you are allowed to generate and the map you generate must be consistent with the input knowledge. Any input knowledge that is known with certainty must be repeated exactly in the map you generate without modification. All transition probabilities and all rewards must be concrete, numerical values. You must sample the distributions you are given and cannot just return the mean value of any input distribution for transition probabilities or rewards. Generate the map using complete sentences starting with the phrase ”You think ” and do not say anything else. Do not say anything about the input knowledge from the agent including the Dirichlet and uniform distributions.

Current knowledge about the next cave transitions and rewards is as follows: `<Input prior/LLM-generated posterior>`. You must carefully read through this knowledge. Never say anything about the agent or tell the agent what to do.

B.3. Combination Lock

The environment description for CombinationLock was given as:

You are a helpful assistant trying to guess the correct code to a combination lock as quickly as possible. The combination lock requires a three-digit code. You will incrementally construct your guess for the code that unlocks the lock by selecting one digit between 0 and 9 at each timestep. The correct code that opens the lock contains no repeated numbers. For each digit you guess, you will be given feedback indicating if the guessed digit is either in the correct position for the unlocking code, in the wrong position for the unlocking code, or does not appear in the combination lock code at all. You will receive a final reward of one if your guessed code correctly unlocks the combination lock. Otherwise, rewards will always be zero. Your only available actions are the digits from 0 to 9.

The posterior sampling LLM system prompts and user prompts were:

You are a helpful assistant trying to aid an agent in guessing an unknown code that will unlock a lock. Given all knowledge the agent currently has about the correct code, you must generate a single guess at what the correct code could be. You must read through the input information provided by the agent very carefully to produce a good, accurate guess for the correct code. The agent's current knowledge about the correct code establishes specific constraints on what your guess can be. You must generate a guess for the correct code that is consistent with these constraints. Before you return your guess, provide a short justification for each individual digit of your guess that describes how the digit is consistent with the input knowledge from the agent. When you return your guess, start with the phrase "You think " and do not say anything beyond providing your guess for the correct code. The environment was described to the agent like this: `<Environment Description>`

The agent's current knowledge about the correct code is the following:`<Input prior/LLM-generated posterior>`. You must carefully read through all information the agent has provided. Never say anything about the agent or tell the agent what decisions to make.

B.4. Wordle

The environment description for Wordle was given as:

You are an agent playing a customized version of the game Wordle. There is a five-letter target word from the English dictionary which you must try to guess as quickly as possible. The target word does not contain any repeated letters. You will incrementally construct your guess for this target word by selecting one letter of the alphabet at each timestep. For each letter you guess, you will be given feedback indicating if the guessed letter is either in the correct position for the target word, in the wrong position for the target word, or does not appear in the target word at all. You will receive a reward of one if your guessed word correctly matches the target word. Otherwise, rewards will always be zero. Your only available actions are letters of the alphabet.

The posterior sampling LLM system prompts and user prompts were:

You are a helpful assistant trying to aid an agent in guessing an unknown target word without any repeated letters from the English dictionary. Given all knowledge the agent currently has about the target word, you must generate a single guess at what the target word could be. You must read through the input information provided by the agent very carefully to produce a realistic, plausible guess for the target word. The agent's current knowledge about the target word establishes specific constraints on what your guess can be. You must generate a guess without repeated letters from the English dictionary for the target word that is consistent with these constraints. Before you return your guess, describe how it obeys all constraints and knowledge provided by the agent. When you return your guess from the English dictionary, start with the phrase "You think " and do not say anything beyond providing your guess for the target word. The environment was described to the agent like this: `<Environment Description>`

The agent's current knowledge about the target word is the following:`<Input prior/LLM-generated posterior>`. You must carefully read through all information the agent has provided. Never say anything about the agent or tell the agent what decisions to make.

B.5. LLM-IDS

As previously mentioned, LLM-IDS retains the approximation posterior LLM for performing posterior updates given agent interactions with the environment. Instead of having two posterior sampling and optimal policy LLMs, LLM-IDS employs two LLMs for computing the expected regret and the information gain about optimal behavior, respectively, of each action in a given state. The current posterior is supplied to both LLMs as input along with the current state and the candidate action being evaluation, thereby requiring a total of $2 \cdot |\mathcal{A}|$ API calls to obtain the two $|A|$-dimensional vectors needed to solve the information-ratio optimization problem.

Using the fact that finding the distribution over actions which minimizes the information ratio is a convex optimization problem that places probability mass on at most two actions (Russo & Van Roy, 2018; Lu et al., 2023), we solve the optimization problem near-optimally by discretizing the unit interval and searching over all pairs of actions.

For the combination lock environment, we know that the value of the optimal policy is exactly 1. Consequently, we charged the expected regret LLM with simply computing the expected return $\mathbb{E}\left[Q^\star(s_t, a)\right]$ and used one minus this output value as the expected regret. The expected regret LLM used the following system prompt and user prompt:

You are a conservative expected optimal action-value function estimator for helping an agent interacting with a sequential decision-making environment. The environment was described to the agent as follows:`<Environment Description>`. You will be give the agent's current posterior distribution over the world and will also be given a current state and a candidate action. With all of these inputs, you must provide a conservative estimate of the expected cumulative return an agent will observe by taking the proposed action from the current state and then following the optimal policy thereafter. Recall that the optimal-value function (also denoted as Q*) is the value obtained from being in a particular state, taking a particular action, and following the optimal policy thereafter. So, in other words, you are meant to evaluate the expected optimal-value function for the current state and candidate action while taking an expectation with respect to the agent's current posterior distribution. Remember that you are estimating value by taking the candidate action in the current state and then having all future actions selected by the optimal policy. The optimal policy will only make future action selections at future states but will not be able to reverse or change the use of the candidate action in the current state. You must take an expectation with respect to the agent's current posterior distribution to compute the expected optimal action-value function. Your estimate of the expected optimal action-value function must be conservative, which means that it is okay if the estimate you return is smaller than the true expected optimal action-value function but it absolutely cannot be larger than the true expected optimal action-value function. Naturally, you are being the most helpful when the estimate you provide is as close to the true expected optimal action-value function as possible while still being a lower bound and not going over it. You must produce a real and concrete numerical value as your estimate and say it as a decimal (no fractions) after "Final expected optimal action-value: ". Whenever possible, show brief calculations with concrete numbers before you give your estimate to quickly justify it. Say nothing after "Final expected optimal action-value: " other than your estimate.

The agent's posterior distribution reflecting knowledge and uncertainty about the world is as follows:`<Input prior/LLM-generated posterior>`. The current state is as follows:`<Current state>`. Please produce a conservative expected action-value function estimate for the following candidate action:`<Candidate action>`. If needed, round your answer to no more than three decimal places.

The information gain LLM used the following system prompt and user prompt:

You are a conservative information gain estimator for helping an agent interacting with a sequential decision-making environment. The environment was described to the agent as follows:`<Environment Description>`. You will be given the agent's current posterior distribution over the world and will also be given a current state and a candidate action. With all of these inputs, you must provide a conservative estimate of how much information the agent will gain about optimal behavior in the environment by taking the proposed action from the current state. Remember that information gain is computed as mutual information or the reduction between prior and posterior entropy, which is measured in bits. Your estimate of the information gained about optimal behavior by taking this action in the environment must be conservative, which means that it is okay if the estimate you return is smaller than the true information gain but it absolutely cannot be larger than the true information gain. Naturally, you are being the most helpful when the information gain estimate you provide is as close to the true information gain as possible without going over it. You must produce a real and concrete numerical value as your estimate and say it as a decimal (no fractions) after "Final information gain: ".Whenever possible, show brief calculations with concrete numbers before you give your estimate to quickly justify it. Say nothing after "Final information gain: " other than your estimate.

The agent's posterior distribution reflecting knowledge and uncertainty about the world is as follows:`<Input prior/LLM-generated posterior>`. The current state is as follows: `<Current state>`. Please produce a conservative information gain estimate (measured in bits) for the following candidate action:`<Candidate action>`. If needed, round your answer to no more than three decimal places. Remember that sub-optimal or incorrect actions can be informative and information can be gained about optimal behavior without taking an optimal action.

C. Additional Results

As noted by Krishnamurthy et al. (2024), the financial and temporal costs of running LLM agents can be quite significant. With only 20 trials, it would be presumptuous to make any sweeping claims about superior performance of one method relative to others. Fortunately, the goal of our multi-armed bandit experiment is aimed at at a relativistic comparison in the quality of exploration with PSRL and LLMs relative to classic TS. To this end, we borrow the surrogate statistics employed by Krishnamurthy et al. (2024) to provide deeper insight into the long-term exploratory behavior of LLM-based PSRL. Figure 8 reports the *suffix failure* frequency, where a suffix failure at time period t is a binary statistic defined as 1 if the optimal action $A^\star$ is never chosen in time periods $[t, T]$ and 0 otherwise. Clearly, an agent experiencing a large number of suffix failures early on in learning would be unlikely to identify $A^\star$ when run for a larger number of time periods. Figure 9 reports the (scaled) *minimum action frequency*, which reports at time period t the frequency of the least-chosen action in the first t time periods: $\frac{1}{t} \cdot \min\limits_{a \in \mathcal{A}} \left|\{A_{t'} \mid t' \in [t], A_{t'} = a\}\right|$. The statistic is scaled by $|\mathcal{A}|$ to reside in $[0, 1]$. As an agent's knowledge of the world accumulates, one would naturally expect an agent to gradually cease selection of some (ideally, sub-optimal) actions and incur lower minimum action frequencies. Together, these two surrogate statistics paint a picture of whether or not the exploration of a LLM bandit agent gravitates toward $A^\star$ over time.

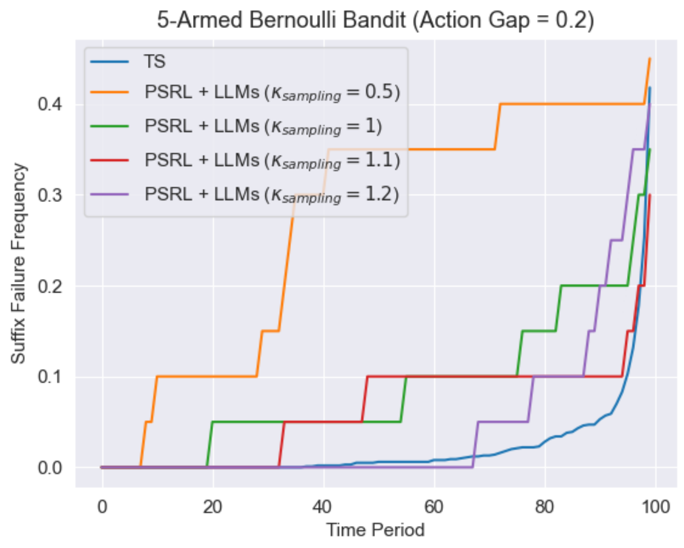


Figure 8. Suffix failure frequency for a 5-armed Bernoulli bandit with $\Delta = 0.2$. A suffix failure occurs at time t if $A^\star$ is never chosen in time periods $[t, T]$.

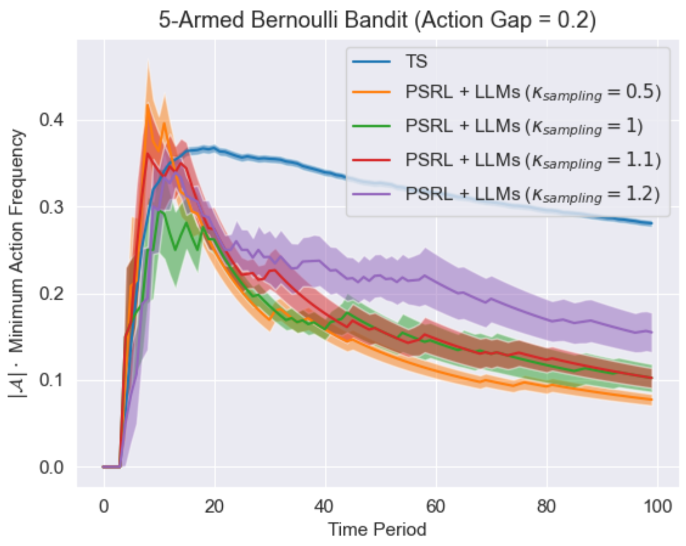


Figure 9. Scaled minimum action frequency for a 5-armed Bernoulli bandit with $\Delta = 0.2$. At time period t, this is the average frequency of the least-chosen action in time periods $[1, t]$.

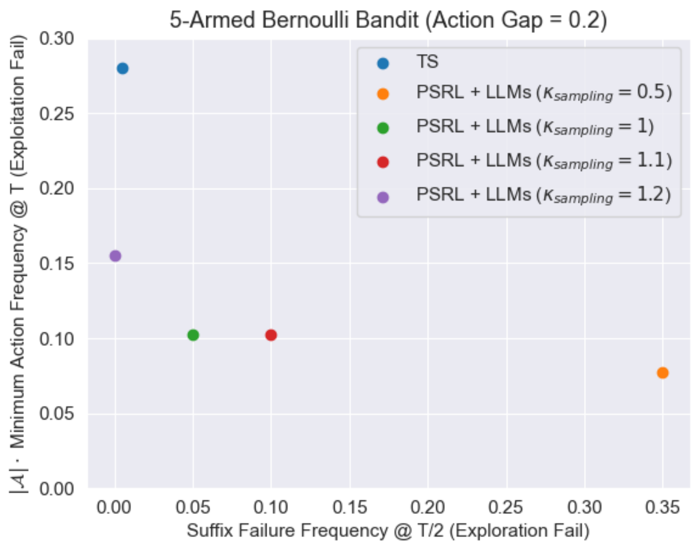


Figure 10. A scatter plot of suffix failure frequency vs. minimum action frequency for Thompson sampling and our LLM-based PSRL with varying κ_{sampling}.