

# PARL: Prompt-based Agents for Reinforcement Learning

Yarik Menchaca Resendiz<sup>1,2</sup>, Roman Klinger<sup>2</sup>

<sup>1</sup>Leibniz-Institut für Psychologie (ZPID), Trier, Germany

<sup>2</sup>Fundamentals of Natural Language Processing, University of Bamberg, Germany  
ymr@leibniz-psychology.org, roman.klinger@uni-bamberg.de

## Abstract

Large language models (LLMs) have demonstrated high performance on tasks expressed in natural language, particularly in zero- or few-shot settings. These are typically framed as supervised (e.g., classification) or unsupervised (e.g., clustering) problems. However, limited work evaluates LLMs as agents in reinforcement learning (RL) tasks (e.g., playing games), where learning occurs through interaction with an environment and a reward system. While prior work focused on representing tasks that rely on a language representation, we study structured, non-linguistic reasoning – such as interpreting positions in a grid world. We therefore introduce PARL (Prompt-based Agent for Reinforcement Learning), a method that uses LLMs as RL agents through prompting, without any fine-tuning. PARL encodes actions, states, and rewards in the prompt, enabling the model to learn through trial-and-error interaction. We evaluate PARL on three standard RL tasks that do not entirely rely on natural language. We show that it can match or outperform traditional RL agents in simple environments by leveraging pretrained knowledge. However, we identify performance limitations in tasks that require complex mathematical operations or decoding states and actions.

**Keywords:** Reinforcement learning, LLM agents, few-shot learning.

## 1. Introduction

Large language models (LLMs; e.g., OpenAI, 2024; Jiang et al., 2024; Devlin et al., 2019) have demonstrated strong performance across a wide range of tasks. While originally developed for text-based applications, they are being applied to other modalities such as vision (Dosovitskiy et al., 2021; Radford et al., 2021) and audio (Borsos et al., 2022; Rubenstein et al., 2023). In natural language processing (NLP), tasks are often framed as supervised learning problems, relying on labeled data, such as classification (Sun et al., 2023b; Schick and Schütze, 2021) and translation (Zhang et al., 2023a; Moslem et al., 2023; Brants et al., 2007), or as unsupervised learning problems, such as text clustering (Zhang et al., 2023b; Viswanathan et al., 2024). Although LLMs have addressed many such tasks using (zero- and few-shot) prompting, they are predominantly based on supervised or unsupervised learning frameworks.

Many real-world problems are, however, more appropriately modeled as reinforcement learning (RL; Kreutzer et al., 2021). In RL, agents learn by interacting with an environment, and take actions and receive feedback in the form of rewards. Unlike supervised learning, which relies on labeled data, RL involves trial-and-error learning with delayed rewards and requires balancing exploration and exploitation. Common applications include learning to play games (e.g., Go (Silver et al., 2016), Atari games (Schrittwieser et al., 2020), Chess (David et al., 2016), BlackJack (Zha et al., 2019)), robotics (Kober et al., 2013)), and improving large models with human feedback (Christiano

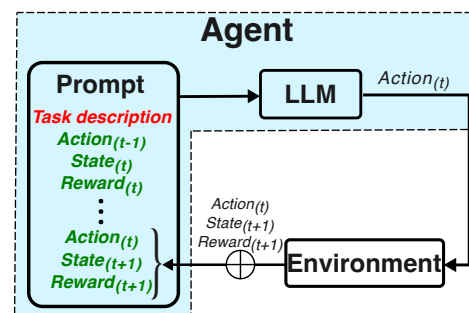


Figure 1: Prompt-based Agent Optimization for Reinforcement Learning (PARL) uses a large language model to make decisions through in-context learning. The optimization begins with a prompt that contains only the **task description**. As the agent interacts with the environment, the prompt is updated by concatenating ( $\oplus$ ) the history of interactions of the **states, actions, and rewards** generated at each step  $t$ .

et al., 2017; Kaufmann et al., 2023).

While recent work has explored LLM-based agents for decision-making via techniques like chain-of-thought prompting or self-reflection (Yao et al., 2023; Zhao et al., 2023; Sun et al., 2023a; Shinn et al., 2023), these methods treat LLMs as text-based agents that rely on natural language for both input and output. Such designs are effective for language-centered tasks (e.g., planning), but their applicability to structured, non-linguistic environments remains underexplored.

In this paper, we investigate and evaluate whether LLMs can act as traditional RL agents in tasks where input and output are not naturally ex-

pressed in natural language – such as navigating a grid world. We introduce PARL (Prompt-based Agent for Reinforcement Learning), a method that keeps LLM weights frozen and uses prompting to enable in-context learning through iterative interaction with an environment. At each step, PARL encodes states, actions, and rewards into a cumulative text prompt, allowing the model to improve decision-making via in-context learning (Figure 1).

We evaluate PARL against three state-of-the-art RL policies across three established RL tasks (Blackjack, Frozen Lake, and Taxi) to address: (RQ1) *Can a PARL agent learn from interactions with the environment similarly to other RL policies?*; (RQ2) *Can a PARL agent benefit from pre-trained knowledge from the LLMs?*; (RQ3) *Does a PARL agent explore and exploit similarly to a standard RL agent?* Our findings show that PARL can match or outperform established policies in knowledge-intensive tasks (e.g., blackjack), where the agent benefits from the LLM’s pre-trained information. However, in more complex tasks, PARL faces challenges due to LLMs’ limitations in symbolic computation and state decoding.

## 2. Related Work

In Section 2.1, we review state-of-the-art reinforcement learning methods, whereas in Section 2.2, we discuss the use of LLMs as agents in RL.

### 2.1. Reinforcement Learning

Reinforcement learning aims at optimizing agents’ actions in sequential decision-making tasks through trial-and-error learning guided by a reward system. The problem is typically formalized as a Markov decision process, defined by  $(S, A, P, R, \gamma)$ , where  $S$  represents the state space,  $A$  the action space,  $P(s'|s, a)$  the state transition probability,  $R(s, a)$  the reward function, and  $\gamma \in [0, 1]$  the discount factor. The goal is to find an optimal policy  $\pi(a|s)$  that maximizes the expected cumulative return:

$$G = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \quad (1)$$

Q-learning and Deep Q-Networks (DQN; Mnih et al., 2013, 2015) set foundational work by integrating neural networks to approximate the action-value function  $Q(s, a)$ , which represents the expected reward for taking action  $a$  in state  $s$ . In DQN, this function is parameterized as  $Q(s, a; \theta)$ , where  $\theta$  denotes the weights of the neural network. The model learns these parameters by minimizing a sequence of loss functions  $\mathcal{L}_i(\theta_i)$  that changes at each iteration  $i$ :

$$\mathcal{L}^{\text{DQN}}(\theta) = \mathbb{E}_{s, a \sim p(\cdot)} \left[ (y_i - Q(s, a; \theta))^2 \right], \quad (2)$$

where  $y_i = \mathbb{E}_{s' \sim \mathcal{E}} [r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a]$  is the target for iteration  $i$  and  $p(s, a)$  is a probability distribution over sequences  $s$  and actions  $a$ .

Proximal Policy Optimization (PPO; Schulman et al., 2017) balances exploration and exploitation while maintaining a stable policy updates by taking small policy updates – big updates may guide the policy in a suboptimal direction. PPO uses a clipped objective function to ensure small policy updates, defined as:

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 \pm \epsilon) \hat{A}_t \right) \right], \quad (3)$$

where  $r_t(\theta)$  is the probability ratio,  $\hat{A}_t$  the advantage estimate, and  $\epsilon$  the clipping parameter.

Advantage Actor-Critic (A2C; Mnih et al., 2016) combines value- and policy-based methods in an actor-critic framework, where the actor  $\pi_\theta(a | s)$  selects actions and the critic  $V(s)$  estimates the value function to reduce variance, improving learning stability. The A2C loss function is defined as:

$$\mathcal{L}^{\text{A2C}}(\theta) = \mathbb{E}_t \left[ \log \pi_\theta(a_t | s_t) \hat{A}_t \right], \quad (4)$$

where  $\hat{A}_t = r_t + \gamma V(S_{t+1}) - V(s)$  is the advantage function computed from the critic’s value estimated.

### 2.2. Prompting in Reinforcement Learning

LLMs, such as GPT-4 (OpenAI, 2024), LLaMa 3 (Team, 2024), and Mixtral (Jiang et al., 2024), have demonstrated strong performance on zero- and few-shot tasks using prompts (Semnani et al., 2023; Lin et al., 2022). These prompting strategies are widely used in traditional NLP tasks. For example, text classification prompts combine instructions with labels (e.g., “Tag the text as positive or negative...”; Hu et al., 2022; Gu et al., 2022), while summarization include keywords like “TL; DR” or “summarize” (Radford et al., 2019; Narayan et al., 2021). Translation prompts specify source and target languages (e.g., “Translate English to German”; Raffel et al., 2020).

While these prompting strategies have primarily been applied to traditional (un)supervised tasks, recent work has explored how LLMs can be integrated into RL as support decision-making algorithms. For instance, Zhao et al. (2023) propose LLM-MCTS, an algorithm that uses LLMs as a commonsense-informed world model and a heuristic policy within the Monte Carlo Tree Search (MCTS) framework. Their approach outperforms standard MCTS and purely LLM-based policies in complex planning tasks. Similarly, AdaPlanner (Sun et al., 2023a) proposes a method that incrementally refines plans

– represented as prompts containing a sequence of high-level steps (e.g., clean lettuce on the dining table) – based on feedback from the environment. This feedback is evaluated by comparing predicted outcomes against actual outcomes, allowing the model to adjust its plan when misalignment occurs.

Other works aim to improve the LLM-based agents through reasoning (e.g., chain-of-thought) and learning from experience (Zhang et al., 2024). ReAct (Yao et al., 2023) interleaves reasoning with actions (interactions with tools or environments), which improves task performance and interpretability while reducing hallucinations. Reflexion (Shinn et al., 2023) extends this by enabling agents to self-critique (e.g., critique their own decisions) and update their behavior over time through memory-based adaptation, improving performance without additional fine-tuning. This self-improving behavior is effective in domains like ALFWorld (an environment for instruction-following tasks) and HotPotQA (question answering across documents).

To evaluate LLMs’ capabilities, recent benchmarks use structured, game-like environments. Clembench (Chalamalasetti et al., 2023) exposes models to constrained decision-making tasks to test instruction-following and consistency. Results indicate that chat-optimized LLMs struggle with simple interactive scenarios, revealing limitations in current agent designs. Similarly, Topsakal and Harper (2024) use games like Tic-Tac-Toe for evaluation, benefiting from well-defined rules and outcomes.

Despite this progress, most prior work uses LLMs as text-based agents, where both input and output are in natural language. While this is effective for tasks like planning or instruction following, it has not been explored or evaluated in settings with structured, non-linguistic inputs – such as grid positions or card values in games like blackjack.

### 3. Methods

In the following section, we introduce PARL (in Section 3.1). PARL adopts the standard RL framework, learning through interaction with the environment to achieve task-specific goals (in Section 3.2)<sup>1</sup>.

#### 3.1. PARL: Prompt Base Agent for Reinforcement Learning

PARL represents a reinforcement learning policy optimization method that uses LLMs as decision-making agents via prompting techniques. It combines the task description  $\mathcal{T}$  with the concatenated interaction history ( $h$ ) with the environment. The

<sup>1</sup>Code and resources available at <https://github.com/YarikMR/PARL>

PARL policy is defined by the prompt:

$$\mathcal{P}^{\text{PARL}} = \mathcal{T} \bigoplus_{t=0}^n h_t, \quad (5)$$

where  $\bigoplus$  indicates concatenation and  $\mathcal{T}$  – the *task description* – is defined as:

$$\mathcal{T} = (\mathcal{G}, \mathcal{A}, \mathcal{S}, \mathcal{R}),$$

where  $\mathcal{G}$  denotes the task goal. For example, in the context of Blackjack: “*Blackjack is a card game where the goal is to get as close to 21 as possible without exceeding it.*”.  $\mathcal{A}$  represents the set of actions available  $\{a_1, \dots, a_k\}$  to the agent (e.g., “0: *Stick (Stop)*, 1: *Hit (Draw)*”).  $\mathcal{S}$  specifies the state representation, describing the information observed by the agent (e.g., “*The observation is a tuple: (player’s value, dealer’s value)*”). Finally,  $\mathcal{R}$  is the set of rewards  $\{r_1, \dots, r_k\}$  (e.g., “+1: *Win*, -1: *Lose*, 0: *Draw*”).  $h_t$  represents the history of interactions with the environment, defined as:

$$h_t = (s_t, a_t, r_t),$$

where  $s_t$  denotes the state at time  $t$ ,  $a_t$  represents the action taken at time  $t$ , and  $r_t$  is the reward received as a result of the action. This iterative log captures the sequence of interactions, recording the state, action, and reward for each time step  $t = 0, \dots, n$ . For example: “*Action (taken): Stick; (new) State: [10, 6, 6], [7]; Reward: 0*”. Figure 2 presents a simplified depiction of the agent, while full examples are provided in Appendix B. State representations – provided by the environment – are often encoded in specific formats (e.g., data structures or numerical values such as hash values calculated out of structured representations). Therefore, decoding state representations can either be handled directly by the LLM – requiring it to interpret the encoded state – or delegated to an external function (e.g., a Python script)<sup>2</sup>. Such script may include world knowledge on how to represent a particular state in a meaningful way in language.

#### 3.2. PARL Training

Training a PARL agent requires creating the history  $h$  between the agent and the environment to enable in-context learning for the LLM. Since LLMs do not retain the memory of previous outputs during inference, the prompt must explicitly accumulate information from prior steps and episodes.

Figure 2 illustrates a simplified depiction of the prompt used for the blackjack RL task. In the first iteration (Step 1 Prompt), the input prompt contains only the task description  $\mathcal{T}$  (goal, action space, observation space, set of rewards), and the initial

<sup>2</sup>Notably, the LLM can generate these scripts itself.

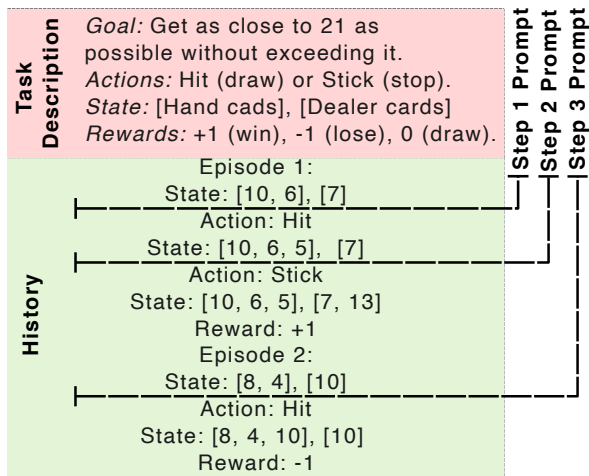


Figure 2: Example of a simplified prompt interaction for the Blackjack task, over three steps with the environment, equivalent to two episodes.

state. The LLM is prompted to generate an action from the specified action space (as described in Section 3.1). This action (e.g., “Hit”) is then fed to the environment, to produce a new state and rewards (e.g., “[10,6,5] and [7]”, representing the three user’s and one dealer’s cards). These updates are concatenated with the  $\mathcal{P}^{\text{PARL}}$  prompt to be used in the next step (e.g., Step 2 Prompt). This process iterates over  $n$  steps, each time appending the log of interactions to the prompt (Eq. 5). With each new episode, additional logs are concatenated to the prompt, gradually increasing the context – with more examples – to enhance decision-making in future iterations through in-context learning.

### 3.3. PARL Inference

The inference process of a PARL agent uses the trained policy, represented as a prompt, which consists of two main components: (1) the task description ( $\mathcal{T}$ , Section 3.1), which includes the task goal, action space, state representation, and reward space, and (2) the history ( $h = \{(s_1, a_1, r_1), \dots, (s_n, a_n, r_n)\}$ ), which records the sequence of states, actions, and rewards with the environment. During inference, only the history from the current episode is concatenated at each step and removed once the episode ends. The PARL agent uses the LLMs to generate an action from the defined action space.

## 4. Experiments

To address our research questions (Section 1), we evaluate the PARL agent under three configurations (Section 4.1): *full context*, *context with random rewards*, and *no context*. Each configuration is tested in two state-decoding variants (Section 4.3). GPT-

4o<sup>3</sup> is used as the underlying language model for all experiments.

The experiments are conducted on three RL tasks: Blackjack, Frozen Lake, and Taxi (Section 4.2). These environments are non-linguistic, offering a clear contrast to typical LLM applications, where natural language is central.

### 4.1. PARL Agent Configurations

**Full History.** This setup defines our standard approach for training PARL agents. We train a PARL agent for 100 episodes for each task. The history  $h$  is gradually concatenated, meaning that episode  $n$  includes context from all previous  $n - 1$  episodes. For example, Episode 1 has no context, while Episode 100 has the history of all previous 99 episodes ( $h = \{(s_1, a_1, r_1), \dots, (s_{99}, a_{99}, r_{99})\}$ ).

**History with Random Rewards.** To assess whether the LLM learns from reward signals, we adopt a similar setup to the *Full History*. The prompt includes the task description ( $\mathcal{T}$ ) and interaction history ( $h$ ), but with true rewards ( $r_t$ ) replaced by random ones ( $r_t^{\text{rand}}$ ), sampled uniformly from the reward set ( $\mathcal{R}$ ). If performance remains comparable to setups with true rewards, it suggests the LLM is not learning from rewards but instead leveraging pre-trained knowledge.

**No History.** In this configuration, the PARL agent has no access to prior episode history. The prompt includes only the task description ( $\mathcal{T}$ ) and the interaction history ( $h$ ) from the current episode. Specifically,  $h = \{(s_t, a_t, r_t)\}$  contains only the states, actions, and rewards from the current episode ( $t$ ). Rewards ( $r_t$ ) are included in the prompt only if provided by the environment at that step (e.g., per-step vs. end-of-episode rewards).

### 4.2. Reinforcement Learning Tasks

**Blackjack.** The *task* is to play a card game aiming to get closer to 21 than the dealer without exceeding it. The *state* is represented as a tuple: the player’s current sum, the value of the dealer’s visible card (1–10, where 1 represents an Ace), and whether the player holds a usable Ace (0 or 1). *Actions* are “Hit” (draw a card) or “Stick” (end turn). *Rewards* are +1 for winning, 0 for a tie, and –1 for losing. The episode *ends* if the player hits and the sum of the hand exceeds 21, or if the player sticks. The *objective* is to learn an optimal strategy for deciding when to “Hit” or “Stick”.<sup>4</sup>

<sup>3</sup>The total cost of the experiments was 145 USD. They have been performed in October 2024.

<sup>4</sup>Gymnasium Blackjack description: [https://gymnasium.farama.org/environments/toy\\_text/](https://gymnasium.farama.org/environments/toy_text/)

**Frozen Lake.** The *task* consists of crossing a frozen lake from start to goal without falling into any holes by walking over the frozen lake. The *state* represents the player’s position as a single integer, given by  $\text{row}_t \cdot \text{ncols} + \text{col}_t$ , where  $\text{row}_t$  and  $\text{col}_t$  are the current row and column, and  $\text{ncols}$  is the total number of columns. *Actions* are of moving *Up*, *Down*, *Left*, or *Right*. *Rewards* are +1 for reaching the goal and 0 otherwise. The episode *ends* if the player moves into a hole, reaches the goal, or exceeds a maximum number of movements. The *objective* is to learn an optimal path to the goal while minimizing the risk of falling into holes.<sup>5</sup>

**Taxi.** The *task* involves driving a passenger in a grid world, picking them up, and dropping them at one of the four locations on the map, one at a time. The *state* is encoded as a single integer that tracks the taxi’s position, the passenger’s location, and the destination:  $((\text{taxi}_{\text{row}} \cdot 5 + \text{taxi}_{\text{col}}) \cdot 5 + \text{passenger}_{\text{location}}) \cdot 4 + \text{passenger}_{\text{destination}}$ . *Actions* consist of *Down*, *Up*, *Left*, *Right*, *Pickup* or *Drop-off*. *Rewards* are +20 for *successfully dropping off* a passenger at the correct destination, −1 for each *step taken*, and −10 for attempting illegal *Pickup* or *Drop-off* actions. The episode *ends* when the taxi successfully drops off the passenger at the correct destination or when the maximum number of steps is reached. The *objective* is to learn efficient pick-up and drop-off strategies to minimize steps and maximize rewards.<sup>6</sup>

### 4.3. State-decoding Variants

Two configurations are used for the state ( $s$ ) in the history  $h$ : (1) the environment state is passed directly to the prompt without preprocessing, requiring the LLM to interpret the raw symbolic or numeric form. (2) An external decoding function (a Python script unique to each environment) converts the state into a natural-language description. In **Blackjack**, the raw state is used as produced by the environment, e.g., “State: [10, 6, 4], [7]”, while the decoded version expresses it in natural language, e.g., “The player’s hand totals 20, and the dealer shows a 7.”. In **Frozen Lake**, the raw state is a grid index, e.g., “State: 6”, and the decoded version describes the position, e.g., “The player is currently located at row 1, column 2 in a 4x4 grid.”. In **Taxi**, the raw state is a tuple encoding position and passenger data, e.g., “State: 6”, and the decoded version reformulates it as “The taxi is at row

blackjack/.

<sup>5</sup>Gymnasium Frozen Lake description: [https://gymnasium.farama.org/environments/toy\\_text/frozen\\_lake/](https://gymnasium.farama.org/environments/toy_text/frozen_lake/).

<sup>6</sup>Gymnasium Taxi task description: [https://gymnasium.farama.org/environments/toy\\_text/taxi/](https://gymnasium.farama.org/environments/toy_text/taxi/).

0, column 0. The passenger is at location Green, and the destination is Yellow.”.

## 5. Results

Before discussing the three configurations of the PARL agent used to address our research questions, we first examine the two state-decoding variants by comparing their training curves: LLM self-decoding (Figure 3) and script-based decoding (Figure 4). Note that reward values are only comparable between the two decoding variants within the same environment, as each task defines its own reward scale and penalty structure (see Section 4.2 and Appendix A for detailed task descriptions).

In Blackjack, the script-based decoding reaches an average reward of 0.6 after only 23 episodes, while the self-decoding only reaches 0.3 after 50 episodes. A similar pattern can be seen in Frozen Lake, where the script variant achieves 1 after 32 episodes, whereas the self-decoding only reaches 0.55 at the end of training. The Taxi environment is more challenging due to its more complex encoding. Here, the LLM is unable to reliably decode positions on its own, leading to random actions. In contrast, the script provides accurate state representations, allowing the LLM to learn low-penalty actions.

Overall, the script-based decoding shows faster learning and higher performance, with steeper learning curves. Based on these results, the following section focuses only on the script-based decoding method.

### 5.1. RQ1: Can a PARL Agent Learn from Environment Interactions?

To evaluate whether the PARL agent learns to take better actions based on previous episodes, we investigate three subquestions: *RQ1.1: Does the PromptRL-Agent benefit from contextual examples during training and inference?* *RQ1.2: What happens when random or incorrect rewards are provided?*; and *RQ1.3: How does the PARL policy compare to state-of-the-art policies?*

#### RQ 1.1: Does a PARL agent benefit from contextual examples during training and inference?

We start by analyzing the role of context during training and inference (Exp. 1; PARL with full history). Figure 4 shows the learning curves for the PARL agent across the three RL tasks. The agent benefits from the contextual examples provided, suggesting that the LLM can learn to make better action choices using contextual examples. In Blackjack, the model’s average reward increases from −0.8 to 0.6 (human performance is around 0.48; Carlin and Robinson, 2009).

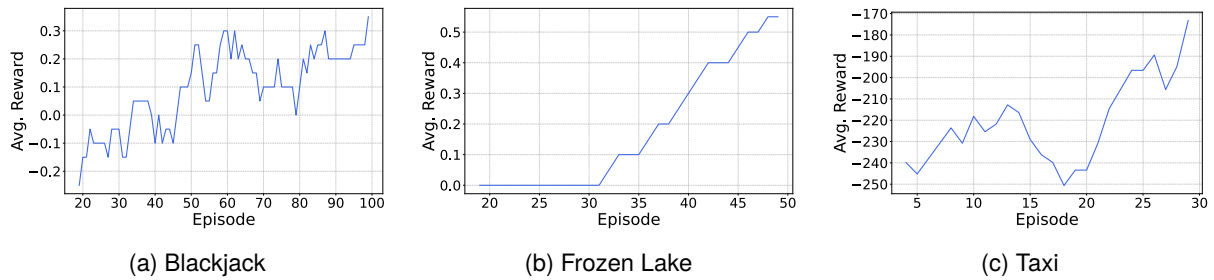


Figure 3: Average reward during training with the PARL agent using LLM self-decoding of states. A smoothing window of five episodes is applied.

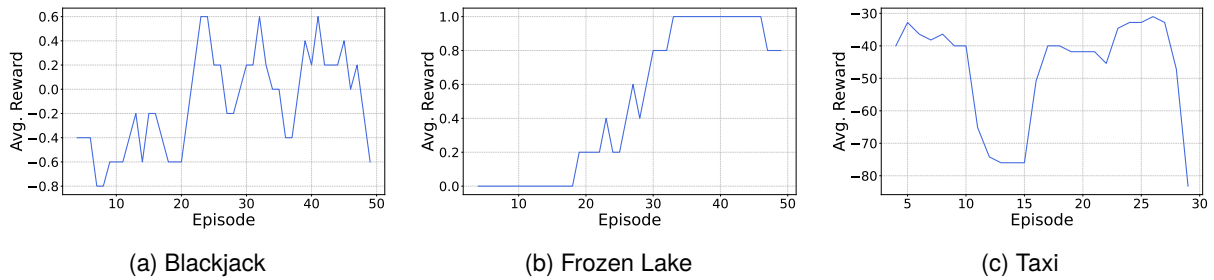


Figure 4: Average reward during training with the PARL agent using script-based state decoding. A smoothing window of five episodes is applied.

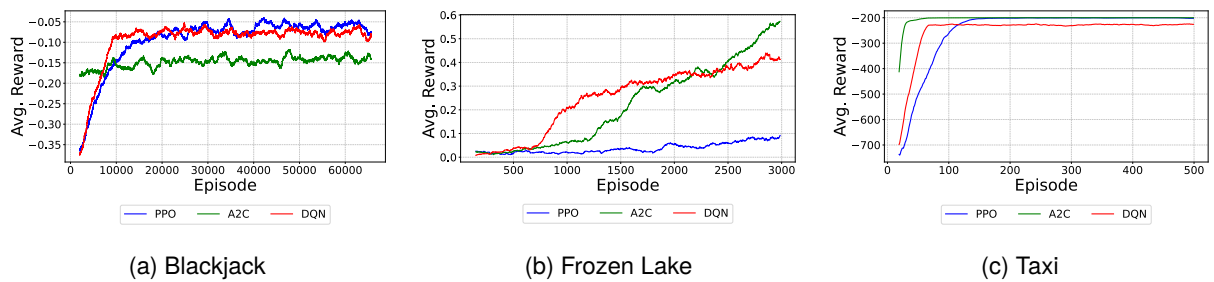


Figure 5: Comparison of average training rewards for SOTA agents (smoothing window = 200).

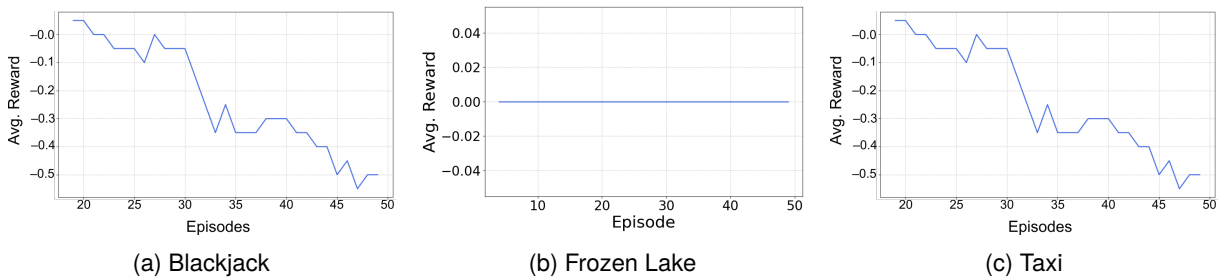


Figure 6: Comparison of the episode real reward during PARL agent training when informing the agent with random or incorrect rewards.

A similar behavior is observed in the Frozen Lake task, where the model initially explores to find the correct path. This training improvement shows the agent's ability to learn better action policies based on the history  $h$ . The Taxi task presents a greater challenge with its larger action space. The LLM is unable to learn to solve the task.

Table 1 reports the average rewards obtained

by the three PARL agent configurations during inference. It presents the average rewards over 100 evaluation episodes for the three configurations of the PARL agent (top) and three state-of-the-art agents (bottom). When comparing the *Full History* configuration to the *History with Random Rewards* and *No History* setups, we observe that history improves the PARL agent's performance. For in-

stance, in Blackjack, the PARL agent achieves an average reward of 0.2 with *Full History*, outperforming the *History with Random Rewards* (−0.12) and *No History* (−0.09) setups. Similarly, in the Frozen Lake task, the *Full History* setup achieves an average reward of 0.66, compared to the 0.0 from the other configurations. On the Taxi task, the PARL agent with *Full History* presents a similar behavior to the training. The agent fails to solve the task but tries to minimize negative rewards (−696.6 compared to −807.5 for *History with Random Rewards* and −568.95 for *No History*). The agent frequently selects sub-optimal actions, such as illegal pick-up or drop-off, which incur a significant penalty (−10).

**RQ1.2: What happens when random or incorrect rewards are provided?** In the configuration where random rewards are included as context, the model fails to learn effectively across all tasks. Figure 6 illustrates that average rewards remain low regardless of training. In Blackjack, the agent’s average reward decreases to −0.5. In Frozen Lake, the agent fails to learn a policy, achieving an average reward of 0.0 (not succeeding in the task even once). In the Taxi task, the agent selects random actions, accumulating high penalties, likely due to the large state and action space. This confirms that irrelevant or misleading context introduces noise, preventing the model from identifying optimal actions.

**RQ1.3: How does the PARL policy compare to state-of-the-art policies?** We compare our policy to three state-of-the-art methods (PPO, DQN, and A2C; Section 2.1). We start by comparing the training setups. Figure 4 presents the average reward during training for the PARL agent and Figure 5 for the SOTA agents. The two figures show similar patterns for Blackjack and Frozen Lake, where average rewards improve with more training iterations.

During training on the more challenging Taxi task, the SOTA agents learn to minimize negative rewards by avoiding high-penalty actions, such as illegal “pick-up” or “drop-off” actions, but they still fail to successfully complete the task. PARL agent initially shows a similar behavior, it then accumulates higher negative rewards per episode, suggesting that adding more examples may introduce noise during training. We can observe a similar result when comparing the episode length from both the PARL (Figure 7) and the SOTA policies (Figure 8). Notably, a PARL agent learns faster, requiring only a small fraction of the training examples needed by the SOTA agents – 100 vs. 100,000.

At inference time, Table 1 shows the average rewards. In Blackjack, the PARL agent (*Full History*) outperforms the SOTA agents by 16 pp – 0.2

vs. 0.04 for PPO, −0.04 for DQN, and −0.02 for A2C. In Frozen Lake, the performance is similar to the other agents – 0.66 vs. 0.67 for PPO, 0.61 for DQN, and 0.62 for A2C. However, in the Taxi task, the PARL agent underperforms compared to all SOTA agents. In addition, none of the agents successfully completed the task, as indicated by the average episode length of 100 – the maximum number of actions allowed in the environment, resulting in the truncation/ending of the episode. The SOTA agents learn to minimize negative rewards by only selecting movement actions (−1) and avoiding high-penalty actions. In contrast, the PARL shows a random behavior, choosing among all possible actions, including illegal *pick-up* and *drop-off* actions (−10).

## 5.2. RQ2: Can a PARL policy benefit from pre-trained knowledge from the LLMs?

To assess the impact of LLM pre-trained knowledge on PARL, we focus on the *No History* configuration in Table 1, where the agent has only access to the current episode’s steps, actions, and rewards. In Blackjack, the agent performs slightly worse than the SOTA models, achieving an average reward of −0.7 vs. 0.04 (PPO), −0.04 (DQN), and −0.02 (A2C). This suggests that the LLM’s prior knowledge of card games aids its decision-making process.

However, in Frozen Lake, the absence of context prevents the agent from discovering the correct path, resulting in failure to accomplish the task. Interestingly, in the Taxi task, the *No History* configuration achieves the highest average reward among all PARL configurations. This suggests that long context introduces noise, complicating the decision-making process more challenging in complex environments. The Taxi tasks environment allows up to 100 steps per episode, compared to an average of only 1.5 steps per episode in Blackjack (Table 1). This extended episode length may provide a PARL agent with more information to learn the task and to prioritize actions with lower penalties – similar to the behavior of SOTA agents. In conclusion, a PARL agent benefits from pre-trained knowledge from the LLMs, but this only applies to tasks that are presumably frequent in the training data (e.g., popular card games and video games).

## 5.3. RQ3: Does a PARL agent explore and exploit similarly to a standard RL agent?

Figure 7 examines whether a PARL agent can explore its environment by reporting average episode lengths across three tasks. We focus on Frozen

Setup	Blackjack		Frozen Lake		Taxi	
	Avg. Reward	Avg. Length	Avg. Reward	Avg. Length	Avg. Reward	Avg. Length
PARL Full History	0.20 (.94)	1.5 (.66)	0.66 (.47)	6.37 (.89)	- 696.6 (28.23)	100 (0.0)
PARL Random Rewards	- 0.12 (.94)	1.5 (.69)	0 (0)	3.86 (.46)	- 807.5 (103.11)	100 (0.0)
PARL No History	- 0.07 (.93)	1.7 (.80)	0 (0)	3.34 (.97)	- 568.9 (64.95)	100 (0.0)
PPO	0.04 (.96)	1.47 (.64)	0.67 (.47)	34.69 (23.69)	- 100 (0.0)	100 (0.0)
DQN	- 0.04 (.95)	1.56 (.74)	0.61 (.48)	36.84 (26.06)	- 100 (0.0)	100 (0.0)
A2C	- 0.02 (.92)	1.24 (.58)	0.62 (.48)	34.96 (23.56)	- 100 (0.0)	100 (0.0)

Table 1: Average (Avg.) reward and episode length across 100 episodes for the Blackjack, Frozen Lake, and Taxi tasks, comparing three PARL agent configurations (top) with state-of-the-art (SOTA; bottom) agents. Standard deviation is in parentheses.

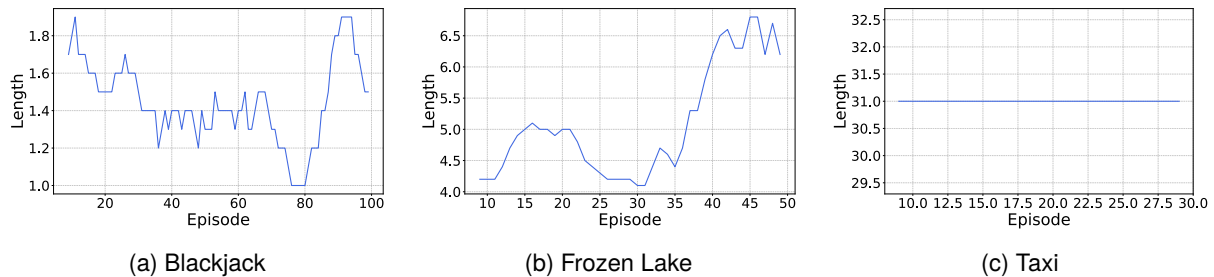


Figure 7: Average episode length during training from PARL agent, with a smoothing window size of five.

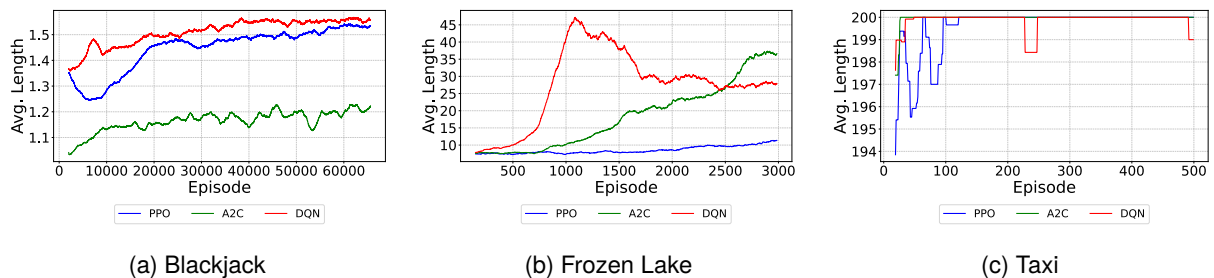


Figure 8: Average episode length during training for SOTA agents (smoothing window = 200).

Lake, which requires exploration, unlike Blackjack (where episode length depends on initial cards) and Taxi (where agents reach the maximum actions per episode without solving the task, as discussed in Section 5.1). In Figure 7b, we observe an initial increase in episode length from 4 to 5, followed by a drop and later stabilization around an average of 7 steps. A closer analysis reveals that the agent initially follows a repeated action sequence (path), which fails to reach the goal (noted by the drop between episodes 20 and 30). It later discovers a successful path (set of actions) and obtains positive rewards. Occasionally, drops in episode length are due to the task’s “slippery” nature, where unintended moves may alter the next state (e.g., moving down might result in ending to the left). This analysis shows that the PARL agent is capable of exploration, but it is limited to exploitation. Once it successfully completes the task, it tends to follow the same set of actions repeatedly, without searching for a more optimal solution.

## 6. Conclusion and Future Work

We introduced PARL, a prompting-based agent that leverages large language models for reinforcement learning in non-linguistic environments. Unlike prior work that focuses on natural language-centered tasks, PARL shows that LLMs can be adapted to basic mathematical decision-making tasks by encoding the environment as textual prompts. Our experiments demonstrate that LLMs can learn from reward-based interactions in simple environments and outperform standard RL baselines in tasks involving common knowledge (e.g., Blackjack). Notably, PARL achieves competitive performance with significantly fewer iterations (100 vs. 100,000), highlighting its sample efficiency. While the PARL exhibits some capacity for exploration, it has limited exploitation behavior. Moreover, it struggles with more complex tasks that require precise computation or abstract state interpretation – such as decoding grid positions – underscoring current limitations

of LLMs in handling raw numeric values.

This work leads to important future research. One promising direction is the use of retrieval-based methods to select relevant past episodes, as long histories ( $h$ ) may introduce noise to the LLM. In addition, extending PARL to a multimodal framework incorporating vision or structured graphs may improve its performance in visual tasks (e.g., Atari and card games).

## 7. Ethical Considerations

The proposed methodology, Prompt-based Agent for Reinforcement Learning (PARL), introduces an alternative optimization policy by using a large language model through prompting techniques. However, the reliance on LLMs raises several ethical concerns that must be addressed.

Firstly, LLMs are trained on vast and diverse datasets, which can include societal biases or generate inappropriate content. When such biases are integrated into reinforcement learning tasks, they may influence the learning process, potentially leading to harmful or unfair outcomes. This issue is particularly critical when applying LLM-based agents in sensitive or real-world domains, such as healthcare, education, or autonomous systems, where biased or erroneous decisions can have significant consequences.

Secondly, the transparency of LLM decision-making poses a challenge. These models function as black boxes, making it difficult to interpret their reasoning processes and understand how decisions are reached. This lack of interpretability complicates efforts to identify and mitigate biases or errors.

To address these challenges, future research should prioritize the development of techniques to reduce biases in LLMs and improve the interpretability of their decision-making processes. It is important to note that such risks are not inherent to PARL methodology but stem from the base LLMs used.

## 8. Limitations

The effectiveness of the proposed PARL method is significantly influenced by the capabilities of the underlying language models (e.g., GPT, LLaMa-7B-Chat, and Mistral-7B). While the approach shows potential in using pre-trained LLMs for reinforcement learning tasks, several limitations must be acknowledged.

Firstly, the method struggles with complex tasks that require advanced mathematical reasoning or precise mappings of states and actions. These limitations present a challenge for the model to

accomplish the tasks that require the encoding of information.

Secondly, the dependence on text-based prompts introduces scalability challenges. Longer training leads to larger interaction histories, which can result in prompts exceeding the input token limits of LLMs. This may lead to the truncation of important information or limit the number of training episodes. As a result, the agent may struggle to learn in longer tasks or more complicated environments.

Third, environments with large state-action spaces may introduce potential noise. This noise can complicate decision-making and reduce PARL's performance in dynamic or highly variable environments.

Fourth, there is an imbalance in model scale. PARL uses large language models (e.g., GPT-4o), while the RL baselines use much smaller networks. This raises fairness concerns. However, the PARL is trained on only a fraction of the episodes compared to the SOTA baselines, which helps balance the comparison to some extent. Still, we acknowledge that model size can affect performance and encourage future work to explore more size-aligned setups.

Despite these limitations, PARL is the first method to demonstrate an optimization policy using an LLM through prompting techniques. The method has proven effective in generating text-based reinforcement learning strategies, particularly for simpler or knowledge-intensive tasks. However, users should be aware of these limitations when evaluating the method's capabilities and potential applications.

## Acknowledgements

This work has been supported by a CONACYT scholarship(2020-000009-01EXTF-00195) and by the project INPROMPT (Interactive Prompt Optimization with the Human in the Loop for Natural Language Understanding Model Development and Intervention, funded by the German Research Foundation, KL 2869/13-1, project number 521755488).

## 9. Bibliographical References

Zalán Borsos, Raphaël Marinier, Damien Vincent, Eugene Kharitonov, Olivier Pietquin, Matthew Sharifi, Dominik Roblek, Olivier Teboul, David Grangier, Marco Tagliasacchi, and Neil Zeghidour. 2022. [Audiolm: A language modeling approach to audio generation](#). *IEEE/ACM Transac-*

- tions on Audio, Speech, and Language Processing, 31:2523–2533.
- Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. [Large language models in machine translation](#). In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, Prague, Czech Republic. Association for Computational Linguistics.
- Bruce I. Carlin and David T. Robinson. 2009. [Fear and loathing in las vegas: Evidence from blackjack tables](#). *Judgment and Decision Making*, 4(5):385–396.
- Kranti Chalamalasetti, Jana Götze, Sherzod Hakimov, Brielen Madureira, Philipp Sadler, and David Schlangen. 2023. [clembench: Using game play to evaluate chat-optimized language models as conversational agents](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 11174–11219, Singapore. Association for Computational Linguistics.
- Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. [Deep reinforcement learning from human preferences](#). *Advances in neural information processing systems*, 30.
- Omid E David, Nathan S Netanyahu, and Lior Wolf. 2016. [Deepchess: End-to-end deep neural network for automatic learning in chess](#). In *Artificial Neural Networks and Machine Learning—ICANN 2016: 25th International Conference on Artificial Neural Networks, Barcelona, Spain, September 6-9, 2016, Proceedings, Part II 25*, pages 88–96. Springer.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. [BERT: Pre-training of deep bidirectional transformers for language understanding](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. [An image is worth 16x16 words: Transformers for image recognition at scale](#). In *International Conference on Learning Representations*.
- Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. 2022. [PPT: Pre-trained prompt tuning for few-shot learning](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8410–8423, Dublin, Ireland. Association for Computational Linguistics.
- Shengding Hu, Ning Ding, Huadong Wang, Zhiyuan Liu, Jingang Wang, Juanzi Li, Wei Wu, and Maosong Sun. 2022. [Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2225–2240, Dublin, Ireland. Association for Computational Linguistics.
- Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. [Mixtral of experts](#). *arXiv preprint arXiv:2401.04088*.
- Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. 2023. [A Survey of Reinforcement Learning from Human Feedback](#).
- Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. [Reinforcement learning in robotics: A survey](#). *The International Journal of Robotics Research*, 32(11):1238–1274.
- Julia Kreutzer, Stefan Riezler, and Carolin Lawrence. 2021. [Offline reinforcement learning from human feedback in real-world sequence-to-sequence tasks](#). In *Proceedings of the 5th Workshop on Structured Prediction for NLP (SPNLP 2021)*, pages 37–43, Online. Association for Computational Linguistics.
- Xi Victoria Lin, Todor Mihaylov, Mikel Artetxe, Tianlu Wang, Shuohui Chen, Daniel Simig, Myle Ott, Naman Goyal, Shruti Bhosale, Jingfei Du, Ramakanth Pasunuru, Sam Shleifer, Punit Singh Koura, Vishrav Chaudhary, Brian O’Horo, Jeff Wang, Luke Zettlemoyer, Zornitsa Kozareva, Mona Diab, Veselin Stoyanov, and Xian Li. 2022. [Few-shot learning with multilingual generative language models](#). In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9019–9052, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. [Asynchronous methods for deep reinforcement learning](#). In *Proceedings of The 33rd*

- International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA. PMLR.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. [Playing atari with deep reinforcement learning](#). *ArXiv*, abs/1312.5602.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. [Human-level control through deep reinforcement learning](#). *nature*, 518(7540):529–533.
- Yasmin Moslem, Rejwanul Haque, John D. Kelleher, and Andy Way. 2023. [Adaptive machine translation with large language models](#). In *Proceedings of the 24th Annual Conference of the European Association for Machine Translation*, pages 227–237, Tampere, Finland. European Association for Machine Translation.
- Shashi Narayan, Yao Zhao, Joshua Maynez, Gonçalo Simões, Vitaly Nikolaev, and Ryan McDonald. 2021. [Planning with learned entity prompts for abstractive summarization](#). *Transactions of the Association for Computational Linguistics*, 9:1475–1492.
- OpenAI. 2024. Gpt-4o model. <https://www.openai.com>. Accessed: [30.03.2024].
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. [Learning transferable visual models from natural language supervision](#). In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. [Language models are unsupervised multitask learners](#). *OpenAI blog*, 1(8):9.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. [Exploring the limits of transfer learning with a unified text-to-text transformer](#). *Journal of Machine Learning Research*, 21(140):1–67.
- Paul K Rubenstein, Chulayuth Asawaroengchai, Duc Dung Nguyen, Ankur Bapna, Zalán Borsoš, Félix de Chaumont Quitry, Peter Chen, Dalia El Badawy, Wei Han, Eugene Kharitonov, et al. 2023. [Audiopalm: A large language model that can speak and listen](#). *arXiv preprint arXiv:2306.12925*.
- Timo Schick and Hinrich Schütze. 2021. [Exploiting cloze-questions for few-shot text classification and natural language inference](#). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online. Association for Computational Linguistics.
- Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. [Mastering atari, go, chess and shogi by planning with a learned model](#). *Nature*, 588(7839):604–609.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. [Proximal policy optimization algorithms](#). *arXiv preprint arXiv:1707.06347*.
- Sina Semnani, Violet Yao, Heidi Zhang, and Monica Lam. 2023. [WikiChat: Stopping the hallucination of large language model chatbots by few-shot grounding on Wikipedia](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2387–2413, Singapore. Association for Computational Linguistics.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. [Reflection: Language agents with verbal reinforcement learning](#). *Advances in Neural Information Processing Systems*, 36:8634–8652.
- David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. [Mastering the game of go with deep neural networks and tree search](#). *nature*, 529(7587):484–489.
- Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023a. [Adaplaner: Adaptive planning from feedback with language models](#). In *Advances in Neural Information Processing Systems*, volume 36, pages 58202–58245. Curran Associates, Inc.
- Xiaofei Sun, Xiaoya Li, Jiwei Li, Fei Wu, Shangwei Guo, Tianwei Zhang, and Guoyin Wang. 2023b. [Text classification via large language models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8990–9005, Singapore. Association for Computational Linguistics.
- Llama Team. 2024. [The llama 3 herd of models](#).

- Oguzhan Topsakal and Jackson B. Harper. 2024. [Benchmarking large language model \(llm\) performance for game playing via tic-tac-toe](#). *Electronics*, 13(8).
- Vijay Viswanathan, Kiril Gashteovski, Kiril Gash-teovski, Carolin Lawrence, Tongshuang Wu, and Graham Neubig. 2024. [Large language models enable few-shot clustering](#). *Transactions of the Association for Computational Linguistics*, 12:321–333.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. [React: Synergizing reasoning and acting in language models](#). In *International Conference on Learning Representations (ICLR)*.
- Daochen Zha, Kwei-Herng Lai, Yuanpu Cao, Songyi Huang, Ruzhe Wei, Junyu Guo, and Xia Hu. 2019. Rlcard: A toolkit for reinforcement learning in card games. *arXiv preprint arXiv:1910.04376*.
- Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. 2024. [Agent-pro: Learning to evolve via policy-level reflection and optimization](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5348–5375, Bangkok, Thailand. Association for Computational Linguistics.
- Xuan Zhang, Navid Rajabi, Kevin Duh, and Philipp Koehn. 2023a. [Machine translation with large language models: Prompting, few-shot learning, and fine-tuning with QLoRA](#). In *Proceedings of the Eighth Conference on Machine Translation*, pages 468–481, Singapore. Association for Computational Linguistics.
- Yuwei Zhang, Zihan Wang, and Jingbo Shang. 2023b. [ClusterLLM: Large language models as a guide for text clustering](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13903–13920, Singapore. Association for Computational Linguistics.
- Zirui Zhao, Wee Sun Lee, and David Hsu. 2023. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems*, 36:31967–31987.

## A. Reinforcement learning environments

The “Toy text” environments from the gymnasium library<sup>7</sup> are designed to be simple and easy to understand, clearly defined states and actions. This simplicity makes them ideal for testing and debugging reinforcement learning algorithms.

### A.1. Blackjack

The Blackjack environment simulates the classic card game Blackjack. The player’s objective is to accumulate a hand of cards whose total value is as close to 21 as possible without exceeding it, while simultaneously outscoring the dealer.

**Description.** The game begins with the dealer holding one card face-up and one face-down, while the player is dealt two face-up cards. Cards are drawn from an infinite deck, meaning cards are replaced after each draw. The player’s total is the sum of their cards. They can choose to draw more cards (hit) or stop drawing (stick). If their total exceeds 21, they go bust and lose immediately. Once the player sticks, the dealer reveals their face-down card and continues drawing until their total is at least 17. If the dealer goes bust, the player wins. If neither the player nor the dealer busts, the winner is determined by whose total is closer to 21. A tie results in a draw. Card values are as follows:

- Face cards (Jack, Queen, King) are worth 10 points.
- Aces can count as either 11 (a “usable ace”) or 1 point.
- Numbered cards (2–9) are worth their face value.

**Action Space.** The action space is discrete, consisting of two possible actions:

- 0: Stick (end the player’s turn).
- 1: Hit (draw another card).

**Observation Space.** The observation space is a tuple of three discrete integers in the format (int, int, int):

- Player’s sum: The total value of the player’s cards.
- Dealer’s visible card: The value of the dealer’s face-up card.
- Usable Ace: A binary indicator (0 or 1) showing whether the player has an Ace that can be used as 11 without exceeding 21.

<sup>7</sup>[https://gymnasium.farama.org/environments/toy\\_text](https://gymnasium.farama.org/environments/toy_text)

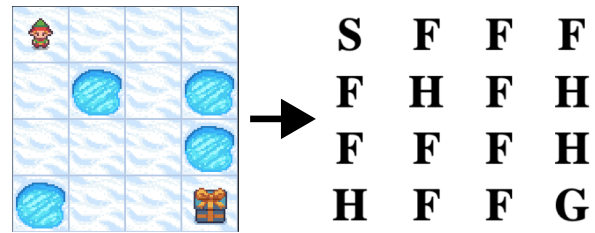


Figure 9: Example of a  $4 \times 4$  map layout for the FrozenLake environment.

**Initial State.** The environment begins with the player’s and dealer’s card values being randomly drawn from the deck. The player is handed two cards, while the dealer receives one face-up card and one face-down card.

**Rewards.**

- +1: The player wins the game.
- -1: The player loses the game.
- 0: The game results in a draw.

**Episode Termination or Truncation.** The episode ends in any of the following scenarios:

- The player’s total hand value exceeds 21.
- The player chooses to stick.

### A.2. Frozen Lake

The Frozen Lake environment simulates a navigation task where the player must traverse a slippery frozen lake to reach a goal without falling into holes.

**Description.** The environment consists of a grid representing a frozen lake (Figure 9 shows an example), where each cell corresponds to a specific type of terrain:

- **S**: Start point (safe).
- **F**: Frozen surface (safe but slippery).
- **H**: Hole (falling in ends the episode).
- **G**: Goal (reaching this ends the episode successfully).

Movement is not always deterministic due to the slippery nature of the lake, which can cause the player to move perpendicular to the intended direction sometimes. Randomly generated worlds will always have a path to the goal.

**Action Space.** The action space is discrete, with four possible actions:

- 0: Move left.
- 1: Move down.
- 2: Move right.
- 3: Move up.

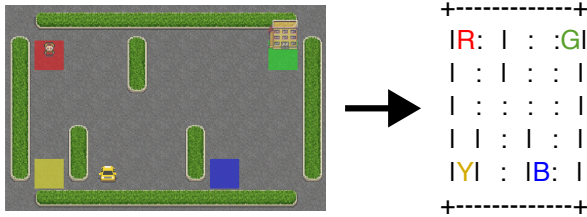


Figure 10: Example the grid map drop-off destinations are Red, Blue, Yellow, and Green.

**Observation Space.** The observation space is a single discrete integer – `int()` – representing the player’s current position on the grid, where both the row and col start at 0. The position is calculated as:

$$\text{current\_row} \times \text{ncols} + \text{current\_col} \quad (6)$$

For example, in a  $4 \times 4$  map, the position of the start cell at the top-left corner can be calculated as:  $0 \times 4 + 0 = 0$ .

**Initial State.** The player starts at the **S** ( $0 \times 0$ ) cell in the grid.

#### Rewards.

- +1: The player reaches the goal (**G**).
- 0: The player falls into a hole (**H**) or moves across frozen surfaces (**F**).

**Episode Termination or Truncation.** The episode ends in any of the following scenarios:

- The player reaches the goal (**G**).
- The player falls into a hole (**H**).
- The player reaches the maximum movements (e.g., 100 for  $4 \times 4$  environment (when using the `time_limit` wrapper)).

### A.3. Taxi

The Taxi environment simulates a grid-world navigation task where a taxi must pick up a passenger at one location and drop them at one of four locations.

**Description.** The environment consists of a  $5 \times 5$  grid with specific locations designated by colors Red, Green, Blue, and Yellow (Figure 10 show an example of the map). At the start of the game, the taxi, the passenger, and the destination are randomly initialized. The goal is to navigate the taxi to the passenger’s location, pick them up, and transport them to the destination. Movement in the grid is deterministic, with walls that act as obstacles between certain cells. The agent must navigate around these walls to complete the task. Actions such as attempting to pick up or drop off a passenger in the wrong location result in penalties.

**Action Space.** The action space is discrete, consisting of six possible actions:

- 0: Move south (down).
- 1: Move north (up).
- 2: Move east (right).
- 3: Move west (left).
- 4: Pick up the passenger.
- 5: Drop off the passenger.

**Observation Space.** The environment contains 500 discrete states, determined by 25 possible taxi positions, 5 possible passenger locations (including the scenario where the passenger is already in the taxi), and 4 destination points. Destinations on the map are identified by the first letter of their corresponding color. Passenger Locations:

- 0: Red
- 1: Green
- 2: Yellow
- 3: Blue
- 4: In the taxi

Destinations:

- 0: Red
- 1: Green
- 2: Yellow
- 3: Blue

An observation is represented as an integer that encodes the state using the formula:

$$((\text{taxi\_row} \times 5 + \text{taxi\_col}) \times 5 + \text{passenger\_location}) \times 4 + \text{destination} \quad (7)$$

There are 400 states that can be accessed during an episode. The missing states occur when the passenger’s location matches their destination, which typically marks the end of the episode. Additionally, four extra states are observed immediately after a successful episode, where both the taxi and the passenger are at the destination. This results in a total of 404 discrete reachable states.

**Initial State.** The taxi’s initial position, the passenger’s location, and the destination are all randomly assigned at the start of the episode.

#### Rewards.

- +20: Successfully dropping off the passenger at the destination.
- –10: Attempting an illegal pickup or drop-off.
- –1: Each time step to encourage efficiency.

**Episode Termination or Truncation.** The episode ends in either of the following scenarios:

- The passenger is successfully dropped off at the destination.
- The maximum number of steps is reached, truncating the episode.

## B. PARL: Prompt-based Agent Optimization for Reinforcement Learning

PARL introduces a novel reinforcement learning method leveraging large language models (LLMs) as decision-making agents through prompting techniques. The method integrates the task description  $\mathcal{T}$  with the agent's interaction history ( $h_t$ ) with the environment to define the policy. The PARL policy is defined as:

$$\mathcal{P}^{\text{PARL}} = \mathcal{T} \bigoplus_{t=0}^n h_t,$$

where  $\mathcal{T}$  represents the task description and  $h_t$  denotes the interaction history.

The combination of  $\mathcal{T}$  and  $h_t$  enables LLMs to make decisions based on interactions with the environment. Examples of PARL agent prompts are presented in Section B.1 for Blackjack, Section B.2 for Frozen Lake, and Section B.3 for Taxi. Detailed explanations for each task can be found in their respective sections.

### B.1. BlackJack Prompt

#### Task description:

Imagine you are a RL agent for a black Jack Agent. The task is the following.

#### Description

The game starts with the dealer having one face up and one face down card, while the player has two face up cards. All cards are drawn from an infinite deck (i.e. with replacement).

The card values are:

- \* Face cards (Jack, Queen, King) have a point value of 10.
- \* Aces can either count as 11 (called a 'usable ace') or 1.
- \* Numerical cards (2-9) have a value equal to their number.

The player has the sum of cards held. The player can request additional cards (hit) until they decide to stop (stick) or exceed 21 (bust, immediate loss).

After the player sticks, the dealer reveals their facedown card, and draws cards until their sum is 17 or greater. If the dealer goes bust, the player wins.

If neither the player nor the dealer busts, the outcome (win, lose, draw) is decided by whose sum is closer to 21.

#### Action Space

The action shape is (1,) in the range {0, 1} indicating whether to stick or hit.

- \* 0: Stick
- \* 1: Hit

#### Observation Space

The observation consists of a 3-tuple containing: the player's current sum, the value of the dealer's one showing card (1-10 where 1 is ace), and whether the player holds a usable ace (0 or 1). The observation is returned as (int(), int(), int()).

#### Rewards

- \* win game: +1
- \* lose game: -1
- \* draw game: 0
- \* win game with natural blackjack: +1.5 (if natural is True) +1 (if natural is False)

#### Episode End

The episode ends if the following happens:

\* Termination:

1. The player hits and the sum of hand exceeds 21.
2. The player sticks.

An ace will always be counted as usable (11) unless it busts the player.

Example of previous episodes:

#### History:

--- Episode 0 ---

Previous observation within the episode:

---Step: 0---

observations: (20, 10, 0)

action taken: 0

Result:

observations: (20, 10, 0)

reward: 1.0

terminated: True

truncated: False

Episode 0 end: Episode reward 1.0

--- Episode 1 ---

Previous observation within the episode:

---Step: 0---

observations: (19, 5, 0)

action taken: 0

Result:

observations: (19, 5, 0)

reward: -1.0

terminated: True

truncated: False

Episode 1 end: Episode reward -1.0

--- Episode 2 ---

Previous observation within the episode:

---Step: 0---

observations: (7, 9, 0)

```

action taken: 1
Result:
observations: (14, 9, 0)
reward: 0.0
terminated: False
truncated: False
---Step: 1---
observations: (14, 9, 0)
action taken: 1
Result:
observations: (18, 9, 0)
reward: 0.0
terminated: False
truncated: False
---Step: 2---
observations: (18, 9, 0)
action taken: 0
Result:
observations: (18, 9, 0)
reward: 1.0
terminated: True
truncated: False
Episode 2 end: Episode reward 1.0

```

```

--- Episode 3 ---
Previous observation within the episode:
---Step: 0---
observations: (18, 10, 0)
action taken: 0
Result:
observations: (18, 10, 0)
reward: 1.0
terminated: True
truncated: False
Episode 3 end: Episode reward 1.0

```

```

--- Episode 4 ---
Previous observation within the episode:
---Step: 0---
observations: (18, 9, 0)
action taken: 0
Result:
observations: (18, 9, 0)
reward: -1.0
terminated: True
truncated: False
Episode 4 end: Episode reward -1.0

```

```

--- Episode 5 ---
Previous observation within the episode:
---Step: 0---
observations: (7, 6, 0)
action taken: 1
Result:
observations: (17, 6, 0)
reward: 0.0
terminated: False
truncated: False
---Step: 1---
observations: (17, 6, 0)
action taken: 0
Result:
observations: (17, 6, 0)

```

```

reward: 1.0
terminated: True
truncated: False
Episode 5 end: Episode reward 1.0

```

## B.2. Frozen Lake Prompt

### Task description:

Imagine you are a Reinforcement Learning Agent for the following task.

Frozen lake involves crossing a frozen lake from start to goal without falling into any holes by walking over the frozen lake. The player may not always move in the intended direction due to the slippery nature of the frozen lake.

### Description

The game starts with the player at location  $[0,0]$  of the frozen lake grid world with the goal located at far extent of the world e.g.  $[3,3]$  for the 4x4 environment.

Holes in the ice are distributed in set locations when using a pre-determined map or in random locations when a random map is generated.

The player makes moves until they reach the goal or fall in a hole.

The lake is slippery (unless disabled) so the player may move perpendicular to the intended direction sometimes (see `is_slippery`).

Randomly generated worlds will always have a path to the goal.

### Action Space

The action shape is  $(1,)$  in the range  $\{0, 3\}$  indicating which direction to move the player.

```

0: Move left
1: Move down
2: Move right
3: Move up

```

### Observation Space

The observation is a value representing the player's current position as `current_row * n_rows + current_col` (where both the row and col start at 0).

For example, the goal position in the 4x4 map can be calculated as follows:  $3 * 4 + 3 = 15$ . The number of possible observations is dependent on the size of the map.

The observation is returned as an `int()`.

### Starting State

The episode starts with the player in state  $[0]$  (location  $[0, 0]$ ).

### Rewards

Reach goal: +1  
Reach hole: 0  
Reach frozen: 0

Episode End

The episode ends if the following happens:

Termination:

The player moves into a hole.

The player reaches the goal at  $\max(\text{nrow}) * \max(\text{ncol}) - 1$  (location  $[\max(\text{nrow}) - 1, \max(\text{ncol}) - 1]$ ).

Truncation (when using the `time_limit` wrapper):

The length of the episode is 100 for 4x4 environment, 200 for FrozenLake8x8-v1 environment.

Example of previous episodes:

### History:

--- Episode 0 ---

Previous position within the episode:

---Step: 0---

Position: 0

action taken: right

Result:

Position: 1

reward: 0.0

terminated: False

truncated: False

---Step: 1---

Position: 1

action taken: right

Result:

Position: 2

reward: 0.0

terminated: False

truncated: False

---Step: 2---

Position: 2

action taken: right

Result:

Position: 3

reward: 0.0

terminated: False

truncated: False

---Step: 3---

Position: 3

action taken: down

Result:

Position: 7

reward: 0.0

terminated: True

truncated: False

Episode 0 end: Episode reward 0.0

--- Episode 1 ---

Previous position within the episode:

---Step: 0---

Position: 0

action taken: right

Result:

Position: 1

reward: 0.0

terminated: False

truncated: False

---Step: 1---

Position: 1

action taken: right

Result:

Position: 2

reward: 0.0

terminated: False

truncated: False

---Step: 2---

Position: 2

action taken: right

Result:

Position: 3

reward: 0.0

terminated: False

truncated: False

---Step: 3---

Position: 3

action taken: down

Result:

Position: 7

reward: 0.0

terminated: True

truncated: False

Episode 1 end: Episode reward 0.0

--- Episode 2 ---

Previous position within the episode:

---Step: 0---

Position: 0

action taken: right

Result:

Position: 1

reward: 0.0

terminated: False

truncated: False

---Step: 1---

Position: 1

action taken: right

Result:

Position: 2

reward: 0.0

terminated: False

truncated: False

---Step: 2---

Position: 2

action taken: right

Result:

Position: 3

reward: 0.0

terminated: False

truncated: False

---Step: 3---

Position: 3

action taken: down

Result:

Position: 7

reward: 0.0

terminated: True

truncated: False

Episode 2 end: Episode reward 0.0

### B.3. Taxi Prompt

#### Task description:

Imagine you are a Reinforcement Learning Agent for the following task:

The Taxi Problem involves navigating to passengers in a grid world, picking them up and dropping them off at one of four locations.

#### Description

There are four designated pick-up and drop-off locations (Red, Green, Yellow and Blue) in the 5x5 grid world. The taxi starts off at a random square and the passenger at one of the designated locations.

The goal is move the taxi to the passenger's location, pick up the passenger, move to the passenger's desired destination, and drop off the passenger. Once the passenger is dropped off, the episode ends. The player receives positive rewards for successfully dropping-off the passenger at the correct location. Negative rewards for incorrect attempts to pick-up/drop-off passenger and for each step where another reward is not received.

#### Observation Space

There are 500 discrete states since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is in the taxi), and 4 destination locations.

Destination on the map are represented with the first letter of the color.

#### Passenger locations:

0: Red  
1: Green  
2: Yellow  
3: Blue  
4: In taxi

#### Destinations:

0: Red  
1: Green  
2: Yellow  
3: Blue

An observation is returned as an int() that encodes the corresponding state, calculated by  $((\text{taxi\_row} * 5 + \text{taxi\_col}) * 5 + \text{passenger\_location}) * 4 + \text{dest}$  ination Note that there are 400 states that can actually be reached during an episode. The missing states correspond to situations in which the passenger is at the same location as their destination, as this typically signals the end of an episode. Four additional states can be observed right after a successful episodes, when both the

passenger and the taxi are at the destination. This gives a total of 404 reachable discrete states.

#### Starting State

The episode starts with the player in a random state.

#### Rewards

-1 per step unless other reward is triggered.  
+20 delivering passenger.  
-10 executing 'pickup' and 'drop-off' actions illegally.

An action that results a noop, like moving into a wall, will incur the time step penalty. Noops can be avoided by sampling the action\_mask returned in info.

#### Episode End

The episode ends if the following happens:  
Termination: 1. The taxi drops off the passenger.

#### Truncation (when using the time\_limit wrapper):

1. The length of the episode is 200.

#### Map:

```
+-----+
|R: | : :G|
| : | : : |
| : : : : |
| | : | : |
|Y| : |B: |
+-----+
```

#### Action Space

The action shape is (1,) in the range {0, 5} indicating which direction to move the taxi or to pickup/drop off passengers.

0: Move south (down)  
1: Move north (up)  
2: Move east (right)  
3: Move west (left)  
4: Pickup passenger (pickup)  
5: Drop off passenger (drop\_off)

#### Example of previous episodes:

##### History:

```
--- Episode 0 ---
Previous position within the episode:
---Step: 0---
Observation: 201
action taken: right
Result:
Observation: 221
reward: -1
terminated: False
truncated: False
Episode accumulative reward -1
---Step: 1---
Observation: 221
action taken: pickup
Result:
```

Observation: 221  
reward: -10  
terminated: False  
truncated: False  
Episode accumulative reward -11  
---Step: 2---  
Observation: 221  
action taken: pickup  
Result:  
Observation: 221  
reward: -10  
terminated: False  
truncated: False  
Episode accumulative reward -21  
---Step: 3---  
Observation: 221  
action taken: down  
Result:  
Observation: 321  
reward: -1  
terminated: False  
truncated: False  
Episode accumulative reward -22  
---Step: 4---  
Observation: 321  
action taken: right  
Result:  
Observation: 341  
reward: -1  
terminated: False  
truncated: False  
Episode accumulative reward -23  
---Step: 5---  
Observation: 341  
action taken: right  
Result:  
Observation: 341  
reward: -1  
terminated: False  
truncated: False  
Episode accumulative reward -24  
---Step: 6---  
Observation: 341  
action taken: left  
Result:  
Observation: 321  
reward: -1  
terminated: False  
truncated: False  
Episode accumulative reward -25  
---Step: 7---  
Observation: 321  
action taken: pickup  
Result:  
Observation: 321  
reward: -10  
terminated: False  
truncated: False  
Episode accumulative reward -35  
---Step: 8---  
Observation: 321  
action taken: right  
Result:

Observation: 341  
reward: -1  
terminated: False  
truncated: False  
Episode accumulative reward -36  
---Step: 9---  
Observation: 341  
action taken: left  
Result:  
Observation: 321  
reward: -1  
terminated: False  
truncated: False  
Episode accumulative reward -37  
---Step: 10---  
Observation: 321  
action taken: pickup  
Result:  
Observation: 321  
reward: -10  
terminated: False  
truncated: False  
Episode accumulative reward -47