LLM-Guided Decision Making in Reinforcement Learning

Mohammad Zaid, **Aman Raj**, and **Srija Mukhopadhyay**

8th May 2025

Abstract

Reinforcement Learning (RL) algorithms frequently encounter challenges with inefficient exploration, particularly in complex environments characterized by sparse rewards. This paper investigates the potential of leveraging the semantic knowledge inherent in Large Language Models (LLMs) to augment the decision-making processes within RL. We introduce a modified ε-greedy strategy wherein, during the exploration phase, action suggestions are solicited from an LLM, rather than relying on conventional uniform random choices. This LLM-guided ε-greedy approach is systematically evaluated on the MiniGrid DoorKey-5x5 and ALFWorld environments, utilizing Deep Q-Network (DQN), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC) algorithms. Our experimental results demonstrate that LLM guidance substantially improves both the convergence rate and the quality of the learned policy for DQN, especially in settings with sparse rewards. Conversely, for policy gradient methods such as PPO and SAC, which depend on nuanced probability distributions over actions, the direct integration of LLM for action selection resulted in a degradation of performance. These findings suggest that while LLMs can effectively guide value-based RL methods, their application to probabilistic policy-based methods necessitates more sophisticated integration techniques.

1 Introduction

1.1 Problem Formulation

This research aims to explore whether the integration of semantic knowledge from Large Language Models (LLMs) can enhance the convergence rate and policy-making capabilities of Reinforcement Learning (RL) algorithms. A common challenge in RL is the inefficiency of standard exploration strategies, such as the ε-greedy method, which employs uniform random exploration. This approach can be particularly detrimental in large or complex state spaces as it fails to leverage any domain-specific or semantic knowledge, potentially leading to prolonged and unfocused exploration.

1.2 RL Environment 1: MiniGrid DoorKey

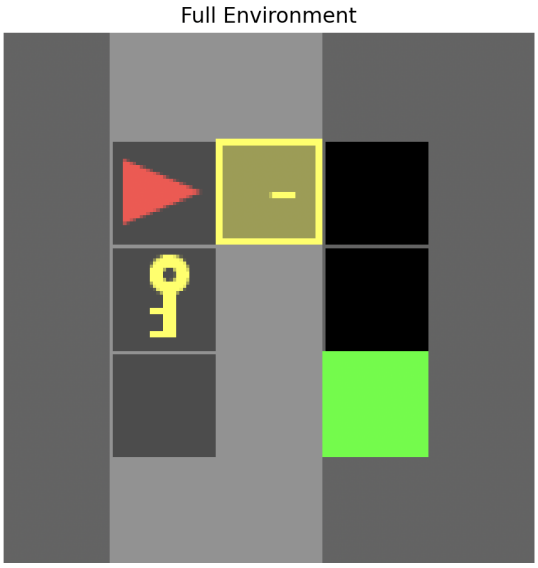

Figure 1: RL Environment 1: MiniGrid DoorKey-5x5 environment

The MiniGrid DoorKey-5x5 environment, depicted in Figure 1, serves as one of our primary testbeds. In this environment, the agent commences its task in a randomly assigned cell, oriented towards one of the four cardinal directions (north, south, east, or west). The environment is characterized by partial observability; the agent's perception is restricted to the cells directly in its line of sight, contingent upon its current position and orientation. The overarching objective for the agent is to locate and retrieve a key, utilize this key to unlock a door, and subsequently navigate to a designated goal cell, thereby successfully completing its mission.

1.3 RL Environment 2: ALFWorld

The ALFWorld environment, illustrated in Figure 2, presents a different set of challenges. Here, the agent is tasked with accomplishing high-level objectives, typically analogous to common household chores. Interaction with this environment is facilitated through text-based commands typed by the agent. Despite its text-based interface, ALFWorld is built upon the THOR (The House Of inteRactions) 3D visual environment simulator, which imbues the interactions with a greater degree of realism. The primary goal for an AI agent within ALFWorld is to successfully execute a given high-level instruction by generating an appropriate sequence of valid, low-level text actions.

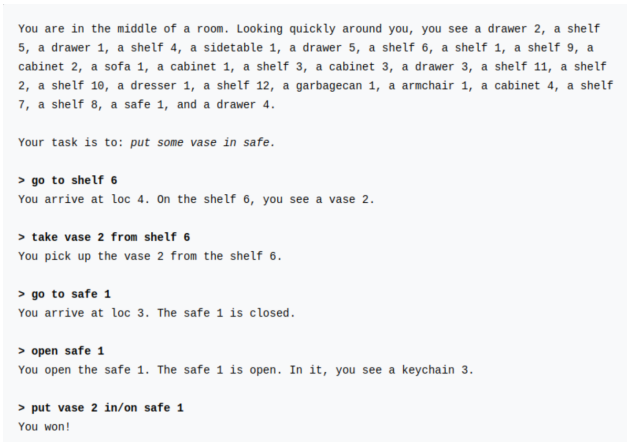

Figure 2: RL Environment 2: ALFWorld environment

2 Our Approach

2.1 ε-greedy method in RL

In Reinforcement Learning, a fundamental challenge lies in balancing exploration (discovering new states and actions) and exploitation (utilizing existing knowledge to maximize rewards). An excess of exploration can lead to unstable learning, as the agent may continually try random actions without consolidating effective strategies. Conversely, excessive exploitation can hinder learning, as the agent might prematurely converge to suboptimal policies without discovering better alternatives. The ε-greedy method addresses this trade-off by introducing a parameter ε, representing the probability with which the agent chooses to explore. With probability ε, the agent selects a random action, and with probability 1-ε, it selects the action deemed optimal by its current policy (exploitation). Typically, ε is initialized to a high value (e.g., 1, encouraging full exploration) and is progressively decayed over time. This progressive reduction in ε ensures that exploration is prominent in the early stages of learning, while exploitation becomes increasingly dominant as the agent gains experience, aiming for stable and effective policy learning.

2.2 Problems with ε-greedy method in RL

While the ε-greedy method provides a straightforward mechanism for balancing exploration and exploitation, its reliance on uniform random action selection during the exploration phase presents notable limitations. This randomness can lead to the selection of actions that are contextually inappropriate or "absurd," potentially resulting in inefficient learning trajectories and poor policy convergence. Furthermore, as ε is progressively decreased, the agent might have explored only a limited, potentially suboptimal, subset of the state-action space if the random choices were not sufficiently diverse or strategic. This can impede the RL algorithm's ability to discover truly optimal policies.

2.3 Problems with ε-greedy method in MiniGrid DoorKey

In the MiniGrid DoorKey environment, the shortcomings of random exploration inherent in the ε-greedy method become particularly apparent. For instance, during the initial exploration phase, the agent might randomly choose to move forward even if it has not yet acquired the necessary key to unlock a door, an action that yields no progress towards the goal. Similarly, it might attempt to pick up the key without being correctly oriented towards it, resulting in a null operation. Such unproductive actions highlight the need for incorporating semantic knowledge to guide the agent towards more meaningful and strategically sound decisions, rather than relying purely on chance.

2.4 Problems with ε-greedy method in ALFWorld

The ALFWorld environment, with its complex, language-grounded tasks, further exacerbates the inefficiencies of ε-greedy exploration. Random actions in this context are highly likely to be semantically meaningless or unproductive, leading to a significant waste of computational steps. The probability of discovering a correct, often long, sequence of actions through purely random exploration is exceedingly small; a single random deviation can invalidate an entire plan. Compounding this difficulty are the sparse rewards, typically granted only upon task completion. Consequently, an agent employing random exploration would rarely receive positive feedback, rendering learning extremely slow or even impossible. The "blind" nature of ε-greedy exploration, devoid of common sense, struggles to identify plausible or useful actions within a given context.

2.5 Proposed: LLM-Guided ε-greedy Exploration

Large Language Models (LLMs) possess extensive semantic knowledge, derived from vast text corpora, which enables them to make contextually informed choices when presented with situational information. To harness this capability for improving RL exploration, we propose a modification to the standard ε-greedy method. Our approach, termed LLM-Guided ε-Greedy, replaces the random action selection component of the exploration phase with an action suggested by an LLM. This aims to inject semantic understanding into the exploration process, guiding the agent towards more promising state-action pairs.

2.6 Mechanism of LLM-Guided Exploration

As illustrated in Figure 3, our LLM-Guided ε-Greedy approach operates by modifying the exploration step of the ε-greedy strategy. When the agent decides to explore (which occurs with probability ε), instead of selecting an action uniformly at random from the available action space, it queries an LLM for an action suggestion. To facilitate this, the current state of the environment is formatted appropriately (e.g., converted to a textual description) and provided as input to the LLM. The LLM then processes this state information and outputs a recommended action. During the initial stages of training, when ε is typically high, the RL algorithm relies more heavily on these LLM-suggested actions for exploration. As training progresses and ε decays, the agent's dependence on the LLM decreases, and it

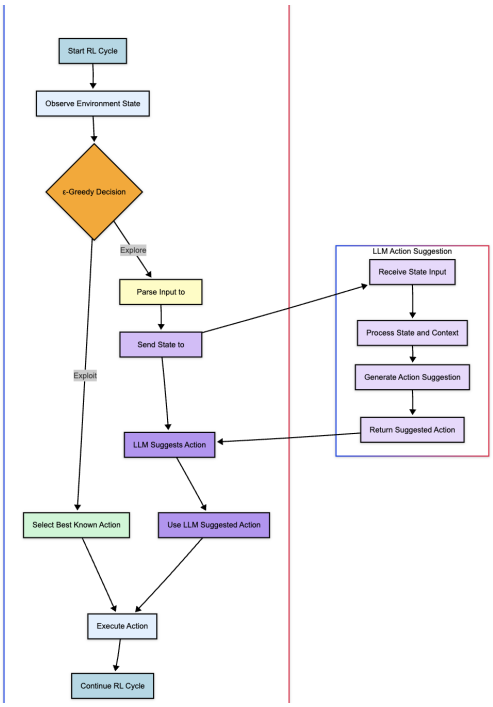

Figure 3: Flow Chart of Our LLM-Guided ε-Greedy Approach

increasingly exploits its own learned policy. This LLM-guided exploration is hypothesized to lead to faster convergence and more stable learning by making the exploration process more intelligent and less arbitrary.

3 Implementation Details

3.1 State to Text for MiniGrid

To enable an LLM to process states from the MiniGrid environment, our implementation includes a crucial step of converting the environment's object-based representation into a textual format. Cells that are not visible to the agent from its current perspective are represented by a placeholder token, such as '?'. Conversely, visible objects, including walls, doors, and keys, are translated into distinct labeled tokens (e.g., 'WALL', 'DOOR', 'KEY'). The agent's orientation within the grid is also accounted for, typically by rotating the grid representation so that the LLM receives a view consistent with the agent's forward direction. This state-to-text conversion is essential for querying the LLM with relevant contextual information. For the ALFWorld environment, this step was unnecessary as it is inherently text-based.

3.2 Implementation - LLM Suggestion for DQN

The integration of LLM suggestions into our Deep Q-Network (DQN) agent is conceptually illustrated in the code snippet shown in Figure 4. The process begins with the conversion of

```python
# Initialize the Ollama client
client = ollama.Client()
# Define the model
model = 'llama3.2'

def suggest_llm_action(env_name, action_dim, state):
    # Convert the state to an LLM readable format
    llm_readable_format = convert_state_to_text(env_name, state)

    # Prompt to the Ollama model
    prompt = f"""
    You are given an RL environment {env_name}.
    The current state is
    '{llm_readable_format}'
    You are an RL expert. Using your knowledge of the environment, suggest a possible action for
    this scenario.
    Strictly, give your answer only in range 0 to {action_dim} and nothing else.
    Description of actions is as follows:

    - 0: turn left
    - 1: turn right
    - 2: move forward
    - 3: pickup
    - 4: drop
    - 5: toggle (to open/close doors)
    - 6: done/noop

    """

    # Run the model
    stream = client.chat(model = model, messages=[{'role': 'user', 'content': prompt}])

    # Return the answer
    suggested_action = int(stream['message']['content'])
    return suggested_action
```

Figure 4: Conceptual code snippet for LLM Suggestion in DQN

the current environment state into a textual format, specifically for the MiniGrid environment. This textual representation of the state is then passed as a prompt to an LLM, along with contextual information such as the environment's name and a description of the actions available to the agent in that particular state. Leveraging its pre-trained knowledge, the LLM infers a suitable action and returns this suggestion. This direct action suggestion approach aligns well with Q-Learning methods like DQN, which learn action-values (Q-values) for state-action pairs. However, this form of direct action output is less straightforwardly applicable to policy gradient methods such as SAC or PPO, which learn a policy that maps states to a probability distribution over actions.

3.3 Implementation - LLM Suggestion for SAC/PPO

For policy gradient algorithms like Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO), which operate on probability distributions over actions rather than deterministic action choices, we adapted the LLM interaction mechanism. Instead of prompting the LLM for a single, discrete action, we requested it to return a probability distribution across the set of available actions. The LLM's output in this scenario is typically a tuple or list, where each element corresponds to the probability of selecting the respective action. This probabilistic output from the LLM is then utilized by the SAC or PPO agent during its exploration steps, replacing or biasing the standard exploration mechanism.

3.4 Implementation - LLM Reflection for ALFWorld

Specifically for the ALFWorld environment, we incorporated an LLM reflection mechanism to enhance decision-making. This involves allowing the LLM to reflect upon a sequence of its previous actions and their observed outcomes, maintained in a short-term memory. Based on this reflection, the LLM is prompted to generate not just a single action, but a

sequence of plausible future actions along with their associated probabilities. For a DQN agent, the action with the highest probability or value derived from these suggestions can be selected. For SAC and PPO agents, the generated probability distributions can be directly incorporated into their action selection process during exploration.

3.5 Implementation - Memory Update for ALFWorld

To support the LLM reflection process effectively in the ALFWorld environment, we implemented a mechanism for continuously updating a short-term memory buffer. Due to the inherent context length limitations of current LLMs, this memory typically stores information from the most recent interactions, such as the last few (e.g., three) actions taken, the LLM's reflections on those actions, and their observed results or outcomes in the environment. This concise history provides the LLM with immediate context for its reflective reasoning.

3.6 Implementation - DQN

We implemented the Deep Q-Network (DQN) algorithm, encapsulating its core logic within a 'QLearningAgent' class. A key modification was made to the 'getAction()' function within this class. As conceptually depicted in Figure 4, this function was adapted to incorporate LLM-generated action suggestions during the exploration phase, which occurs with probability ε. In our experimental setup, the value of ε is typically annealed over time, commonly by multiplying it with a decay factor at the end of each training step or episode, to gradually shift from exploration to exploitation.

3.7 Implementation - PPO and SAC

In addition to DQN, we implemented the Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) algorithms. These actor-critic methods are characterized by their use of a stochastic policy, meaning they learn a probability distribution over actions for any given state. To integrate LLM guidance, we modified their respective 'getAction()' (or equivalent policy sampling) functions. During exploration phases, determined by the ε parameter, these functions were adapted to utilize the probabilistic action distributions obtained from the LLM, instead of relying solely on sampling from the agent's current learned policy or a uniform random distribution.

3.8 Implementation - Training Loop

A unified training loop was developed to accommodate all implemented RL algorithms (DQN, PPO, and SAC), leveraging their abstracted 'getAction()' methods which internally handle LLM queries when applicable. This loop orchestrates the agent-environment interaction over a series of episodes. In each episode, the agent observes states, selects actions (potentially guided by the LLM based on the current ε value), receives rewards, and gathers experiences. These experiences are then used to update the agent's policy (for PPO/SAC) or value functions (for DQN), facilitating the learning process.

4 Results and Analysis

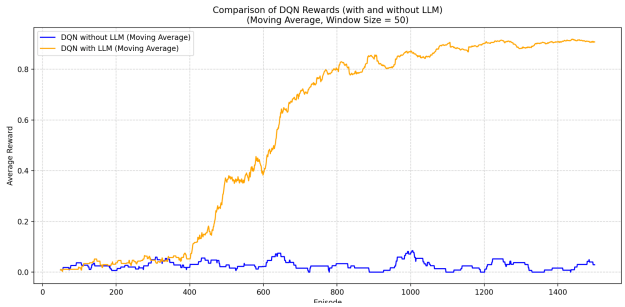

Figure 5: Illustrative Comparison: DQN with and without LLM (MiniGrid DoorKey)

4.1 Experiments Without LLM - Results

The baseline performance of standard RL algorithms, without any LLM guidance, is summarized in Table 1. Notably, the DQN algorithm failed to converge within 10,000 iterations in either the MiniGrid DoorKey or the ALFWorld environment. In contrast, for the MiniGrid DoorKey environment, PPO and SAC demonstrated convergence, achieving it in approximately 1,500 and 1,300 iterations, respectively. However, similar to DQN, neither PPO nor SAC converged in the more complex ALFWorld environment within the 10,000 iteration limit.

Algorithm	**Convergence Result (Iterations)**	**Comments**
DQN	No convergence (10,000+) for MiniGrid DoorKey. No convergence (10,000+) for ALFWorld.	Struggles with sparse rewards.
PPO	Converges 1,500 for MiniGrid DoorKey. No convergence (10,000+) for ALFWorld.	Better exploration than DQN.
SAC	Converges 1,300 for MiniGrid DoorKey. No convergence (10,000+) for ALFWorld.	Often efficient in discrete action spaces too.

Table 1: Convergence results without LLM integration.

4.2 Experiments Without LLM - Analysis I (DQN)

The lack of convergence for DQN in the baseline experiments can be primarily attributed to the challenge of sparse rewards. In environments where feedback is infrequent, agents often engage in prolonged periods of random or uninformative behavior before stumbling upon a rewarding state. Specifically, in the MiniGrid DoorKey environment, the DQN agent frequently receives no reward for its actions, leading to minimal or no meaningful updates to its Q-values, thereby stagnating the learning process. Even when a reward is eventually obtained, standard DQN struggles with effective credit assignment for crucial earlier actions,

as it typically updates only the Q-value of the most recent state-action transition in its basic implementation.

4.3 Experiments Without LLM - Analysis II (PPO & SAC)

PPO and SAC generally exhibited better performance than DQN in these baseline scenarios, particularly in MiniGrid DoorKey. This can be ascribed to their policy-based nature and, in some cases, more sophisticated intrinsic exploration mechanisms, such as entropy regularization in SAC. In the MiniGrid DoorKey environment, SAC slightly outperformed PPO. Although SAC was originally developed for continuous action spaces, it can be adapted for discrete spaces and often demonstrates strong performance due to its exploration-exploitation balance. These algorithms showcase superior learning capabilities in sparse-reward environments compared to a basic DQN, yet they still faced insurmountable challenges in ALFWorld.

4.4 Experiments Without LLM - Analysis III (ALFWorld)

In the highly complex ALFWorld environment, none of the baseline algorithms (DQN, PPO, SAC) achieved convergence. This outcome is attributed to the environment's inherent difficulties, including its high-dimensional state and action spaces, long task horizons, and the stringent requirement for precise sequences of actions to achieve success. Rewards in ALFWorld are typically granted only upon the successful completion of an entire task, making the problem of credit assignment exceptionally challenging without advanced techniques such as reward shaping or curriculum learning. PPO and SAC, while more robust than DQN, still rely fundamentally on trial-and-error exploration. Lacking inherent high-level planning capabilities, they struggle to discover the long, coherent action sequences necessary for success in such intricate, language-grounded tasks.

4.5 Experiments With LLM - Results

The integration of LLM-guided exploration yielded varied results across the different RL algorithms, as detailed in Table 2. A significant improvement in performance was observed for the DQN algorithm when augmented with LLM guidance. Conversely, the performance of PPO and SAC degraded when LLM-guided exploration was incorporated using our direct suggestion method.

4.6 Experiments With LLM - Analysis I (DQN)

The DQN algorithm demonstrated substantial improvements with the integration of LLM guidance, as illustratively compared in Figure 5. The semantic knowledge provided by the LLM offered a more informed exploration strategy compared to the naive random action selection of standard ε-greedy. This guided exploration proved particularly beneficial in mitigating the challenges posed by sparse rewards. With LLM assistance, DQN's convergence in the MiniGrid DoorKey environment (around 1,650 iterations) approached the performance levels of standalone PPO and SAC without LLM guidance. More strikingly, in the ALFWorld environment, where the baseline DQN failed to converge, LLM guidance enabled it to

Algorithm	Convergence Result (Iterations)	Comments
DQN	Converges 1,650 for MiniGrid DoorKey. Converges 5,700 for ALFWorld.	LLM significantly aids DQN in sparse reward settings.
PPO	No convergence / Performance degradation.	LLM's discrete action/probability suggestions may conflict with PPO's policy learning.
SAC	No convergence / Performance degradation.	Similar to PPO, LLM guidance as implemented may disrupt SAC's learning dynamics.

Table 2: Convergence results with LLM integration.

achieve convergence (around 5,700 iterations). These results strongly suggest that LLMs can significantly enhance DQN's performance, especially in environments where common-sense reasoning is advantageous for exploration or where reward signals are infrequent.

4.7 Experiments With LLM - Analysis II (PPO & SAC)

The observed degradation in performance for PPO and SAC when using our LLM-guided exploration method suggests an incompatibility between the direct LLM action/probability suggestions and the learning dynamics of these algorithms. PPO and SAC learn a stochastic policy and rely on specific mathematical properties of the action probability distributions they generate and sample from. Forcing exploration actions based on LLM suggestions—whether as discrete actions or as LLM-generated probability distributions—might introduce behavior that is effectively off-policy or involves distributions not well-aligned with the agent's evolving policy structure. This mismatch could lead to training instability, hinder the policy update mechanism, or guide the agent towards suboptimal policies. This indicates that naively replacing exploration steps with LLM outputs may not be a suitable strategy for these advanced policy gradient methods, and more sophisticated integration techniques are required to harness LLM knowledge effectively.

4.8 Experiment with LLM Alone (MiniGrid DoorKey)

To further understand the capabilities of the LLM in isolation, we conducted an experiment where the LLM attempted to solve the MiniGrid DoorKey environment directly for 100 games, without the involvement of any RL algorithm. The LLM was prompted with the textual state representation and asked to output an action at each step. As conceptually illustrated in Figure 6, while the LLM demonstrated an ability to achieve high rewards in some instances, indicative of its semantic understanding, it failed to secure any reward in the majority of the games. This outcome strongly suggests that while LLMs possess valuable semantic knowledge, they may lack the inherent trial-and-error based learning, adaptation, and robust sequential decision-making capabilities characteristic of RL methods, which are often necessary to consistently solve such interactive and partially observable tasks.

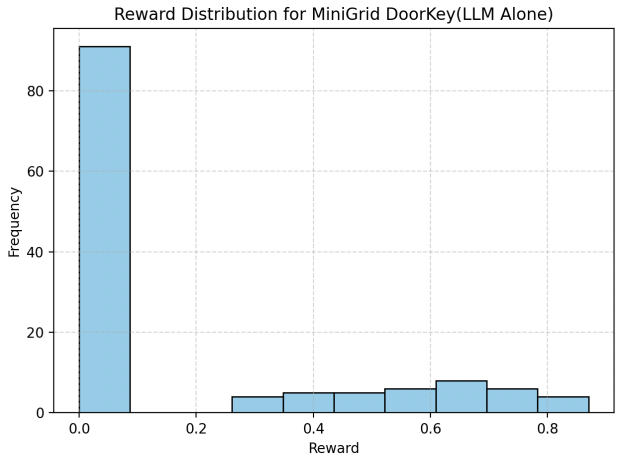

Figure 6: Performance of LLM Alone for MiniGrid DoorKey (Illustrative)

5 Conclusion

Our investigation reveals that leveraging the semantic knowledge of Large Language Models can offer substantial benefits to Reinforcement Learning, particularly for algorithms relying on action-value estimation, such as DQN. In these instances, LLMs provide more informed exploration choices compared to standard random selection, proving especially advantageous in environments with sparse rewards. This guidance led to significant improvements in performance and accelerated convergence for DQN.

However, the direct integration of LLM suggestions (either as deterministic actions or as LLM-generated probability distributions during exploration phases) with probabilistic policy gradient algorithms like PPO and SAC resulted in performance degradation. This suggests that the current methods of soliciting guidance from LLMs may not produce action probabilities or choices that are compatible with the intricate learning dynamics of these advanced algorithms. The nuanced way PPO and SAC update their policies based on sampled actions and their resulting probabilities appears to be disrupted by the external imposition of LLM-derived exploratory actions.

These findings indicate that while LLMs hold considerable promise for enhancing RL, the method of integration is crucial. For value-based methods like DQN, a relatively straightforward substitution of random exploration with LLM suggestions can be effective. For policy gradient methods, more sophisticated integration strategies are likely necessary. Future research could explore avenues such as using LLMs for intelligent reward shaping, hierarchical goal generation, or as a component within a more complex cognitive architecture that reasons about plans or sub-policies, rather than directly dictating low-level actions during exploration.

References

[1] Lili Chen, Kevin Lu, Aravind Rajeswaran, Michael Lee, Aditya Grover, and Pieter Abbeel. *Decision Transformer: Reinforcement Learning via Sequence Modeling.* arXiv

preprint arXiv:2106.01345, 2021.

[2] Shunyu Yao, Jie Yang, Daisy Zhang, Percy Liang, and Linxi "Jim" Nie. *ReAct: Synergizing Reasoning and Acting in Language Models*. arXiv preprint arXiv:2210.03629, 2022.