

Utility of LLMs as The Foundation for Generalized Reinforcement Learning Agents

I. ABSTRACT

This study explores the feasibility of using out-of-the-box Large Language Models (LLMs) as policy generators for reinforcement learning (RL) agents, without domain-specific training. We aim to test whether the inherent natural language processing ability of LLMs is sufficient to encapsulate key components of established RL algorithms such as exploration, learning, memory, and transfer learning. In contrast to other RL algorithms, our framework bridges the agent-, language- and cognition-centric models of AI by maximizing reward (agent), processing and summarizing text-based information (language), and reasoning (cognition). By integrating LLMs with RL, we aim to develop agents capable of learning and adapting across diverse tasks and leveraging prior experiences to inform decision-making, a key characteristic of AGI.

In this report, we introduce LLM-RL, a novel RL agent framework built upon structured LLM queries composed of observations, action spaces, contextual summaries, and reward signals. We evaluate LLM-RL in four MiniGrid environments—Empty, Go-to-Object, MemoryS7, and Door-Key—covering tasks such as navigation, object interaction, memory usage, and long-horizon planning. The difficulty of tasks vary between environments, enabling a thorough evaluation of the agent’s performance, including domains where it performs reasonably well given carefully crafted prompts, as well as scenarios where it struggles considerably. Prompt engineering (e.g., imperative tone, examples) and lightweight memory augmentation yield only limited gains and sometimes introduce hallucinations or unstable behavior.

Compared to standard RL baselines such as DQN and PPO, LLM-RL, although extremely sample efficient, consistently underperforms fully optimized traditional methods in terms of cumulative reward and task success rate. These findings highlight both the promise and current limitations of LLM-based agents, underscoring the need for future work on grounding techniques, memory mechanisms, and multi-modal integration in language-driven RL to enhance their effectiveness.

II. RELATED WORKS

A. RL Agents

Traditional reinforcement learning (RL) methods such as Q-learning [17], DQN [11], PPO [13], and Actor-Critic [9] have been widely used for policy learning and decision-making. These models rely on reward-based optimization but often struggle with long-horizon planning [3], generalization [10], and efficient learning from offline data [12]. Recent approaches

address these limitations by techniques such as incorporating structured memory and predictive modeling. For instance, Fortunato *et al.* [5] explore RL agents augmented with working and episodic memory, demonstrating improved generalization. Additionally, Friston *et al.* [6] introduce *Active Inference*, an approach that minimizes uncertainty to improve adaptability, making RL agents more robust in dynamic environments.

B. LLM-approaches as RL Agents

Recent research explores the potential of LLMs as RL agents, replacing traditional architectures with *sequence modeling* and *reasoning-based decision-making*.

LLMs have demonstrated *trajectory-based policy learning* as an alternative to standard RL paradigms. *Decision Transformer* [2] reframes RL as a sequence modeling task, where transformers predict future actions conditioned on past trajectories. This method achieves competitive performance with DQN and PPO, highlighting the effectiveness of LLMs in trajectory-based RL.

Memory-augmented approaches further extend LLM utility in RL settings. *Reflexion* [15] enhances RL agents by incorporating *episodic memory and textual feedback*, enabling policy refinement thorough self-correction. *AI-Native Memory* [14] introduces long-term memory optimization to improve context retention and adaptability in RL tasks, aligning with the need for *persistent knowledge accumulation* in generalist agents.

LLMs also exhibit *strong zero-shot and multi-task generalization*. *Voyager* [16] demonstrates *continual learning and reusable skill acquisition* in open-ended environments, suggesting that LLM-based agents can autonomously *expand their knowledge base* over time. *SayCan* [1] integrates *affordance-based planning* with LLMs for robotic control, ensuring that generated actions are *both feasible and goal-oriented*. Similarly, *LLMs as Zero-Shot Planners* [8] show that LLMs can generate task plans without *task-specific fine-tuning*, further reinforcing their flexibility in reinforcement learning applications.

C. Multi-Task RL and Generalization

Scaling RL agents across multiple tasks remains a challenge, but LLMs show promise in this area. *DreamerV3* [7] employs world models for sample-efficient multi-task learning. *Ghost* in the *Minecraft* [18] demonstrates LLM-driven decision-making in open-ended, text-based environments. *Active Inference* in RL [6] introduces uncertainty minimization, improving adaptability in dynamic multi-task settings. These works highlight

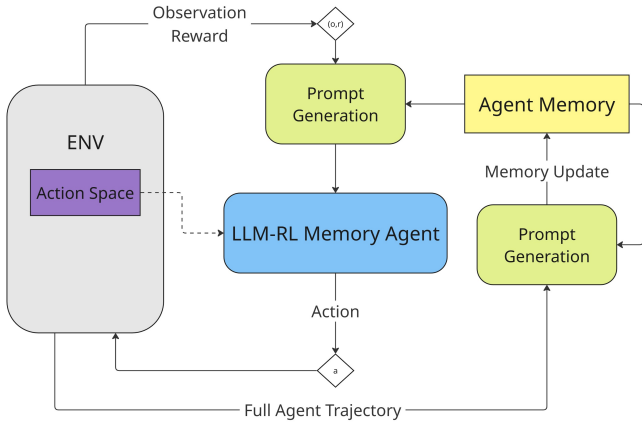


Fig. 1: LLM-RL Memory agent structure

LLMs’ ability to generalize across diverse environments with minimal additional training.

III. METHODS

We iteratively develop a generalized LLM based learning agent, allowing us to compare the performance of a minimal core architecture to one with expanded capabilities.

A. Base LLM-RL Agent

The core agent architecture and how it would interact with an environment is illustrated in Figure 1.

The Base LLM-RL Agent leverages Ollama as its foundation for language model capabilities. Ollama provides a flexible interface for interacting with various language models while maintaining consistent response formats. This allows the agent to utilize different language models (like *Llama*, *DeepSeek*, or *CodeLlama*) while keeping the same core interaction patterns.

The agent class expects an LLM model name, an agent configuration, an action space, an observation space, and a valid action space. The LLM is initiated using Ollama, and the configuration is used to create customizable prompts for the LLM. Custom prompts are equivalent to a lambda function which is passed both the action space and the observation space.

In the policy loop, the LLM-RL Agent uses the agent configuration to generate a prompt for the LLM. Then the agent will call the LLM with the generated prompt and return the action. Unlike a typical LLM call, in the LLM-RL Agent, the LLM is called in a function-calling manner, where the valid action space can be set as a mandatory response format.

This constraint system is implemented at the model inference level within Ollama, ensuring that the language model’s output is always valid according to the schema. Notably, it is not just validating after the fact - it is constraining the model’s generation process to only produce valid responses.

This makes the system more robust and efficient, as invalid responses are prevented rather than just caught and rejected.

The LLM-RL Agent has the following response validation criteria. It is always required to returning a proper action, which is an integer (0-9), and optionally must provide a string containing its reasoning. When a reasoning string is required, the LLM must to explain its reasoning for why it will choose an action prior to selecting it. This is used later to form more advanced prompts.

B. Environment To Text

In order for the LLM to interact with the existing Gym environments, we created wrapper environments that transformed the environments from lists of ints and floats into human-readable descriptions. The original environment consists of tiles represented by three-dimensional tuples containing an object ID, color ID, and state, where the state can be either open, closed or locked. Additionally, the observation space includes the current direction of the agent and an environment-specific, natural language mission.

MiniGrid wrapper environment we’ve built flattens the environment to a 1D list, translating each tile in the $n \times m$ grid. The wrapper also pairs each possible action with an english descriptions, allowing the LLM to interface with the environment through text.

C. Evaluation Environments

We evaluated our agents in four distinct MiniGrid environments: Empty, Go-to-Object, Memory, DoorKey. The available actions vary from one environment to another, but all environments utilize the same three basic actions of turn left, turn right, and move forward.

The Empty environment serves as a baseline to assess the agent’s navigation capabilities. It consists of a simple, open grid with no objects or barriers, and the agent’s objective is simply to reach a designated goal square.

The Go-to-Object environment is designed to test the agent’s ability to navigate toward a specified objects. It emphasizes spatial reasoning, due to the added complexity of navigating around obstacles. The dynamic nature of goals in this environment helps ensure that an agent’s navigation policy is robust enough to handle more nuanced mission statements. In addition to the previously mentioned actions associated with navigation, the agent is given the ability to indicate when it has reached the desired object, a skill useful for more complicated, multi-step tasks.

The MemoryS7 and MemoryS11 environments assess the agent’s memory and recall abilities over the course of a run. In these environments, an agent starts in a small room with a single object. It then must navigate through a splitting hallway in order to find the object that is the same as the object in the starting room.

The DoorKey environment evaluates the agent’s ability to both navigate and interact with objects in a structured way. In this environment, the agent must locate a key, use it to unlock a door, and proceed to a designated goal area. Additional

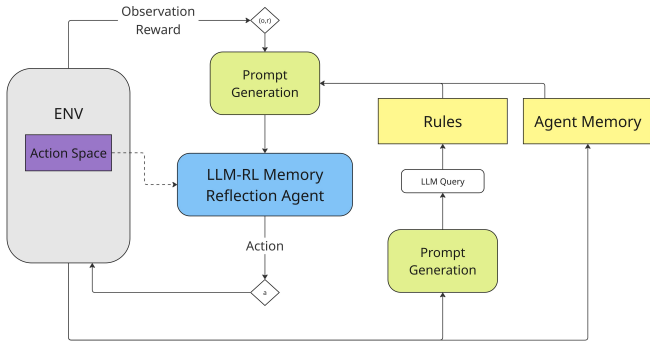


Fig. 2: LLM-RL Memory Reflection agent structure

actions include picking up a key and toggling/activating the said key. This task is particularly useful due to its sparse rewards, requiring the agent to plan a trajectory of actions while avoiding unnecessary movements.

D. LLM-RL Memory Agent

The LLM-RL Memory Agent extends the base agent capabilities by adding a memory of observations, actions, and rewards for each episode. These attributes are appended for each step and fed into the prompt during decision making, allowing the agent to choose the current action based on its history of experiences, as depicted in Figure 1. Leveraging this in-context memory allows the agent to reason over previous episodes implicitly, ideally enabling it to avoid repetitive mistakes.

E. LLM-RL Memory-Reflection Agent

The LLM-RL Memory-Reflection Agent builds on the LLM-RL Memory agent and adds a key reflection component for the agent to explicitly learn from and reflect on past mistakes. The agent architecture is depicted in Figure 2.

Once the agent either reaches its goal or reaches the maximum number of steps, the agent is prompted with its in-context memory and prompted to reflect on improvements it could make. The reflections and rules learned are used alongside the memory in the next decision-making prompt. By continuously recalling previously learned rules and by adding new rules at the end of each episode, the agent is expected to iteratively improve and correct mistakes over time.

Similarly to the enforced format of the action space response, the agent’s reflection is restricted to a list of strings to prevent unwanted behavior, such as summarization or rephrasing of the prompt. This ensures consistency and allows easy integration of these rules into future prompts.

IV. LLM-RL AGENT EVALUATION

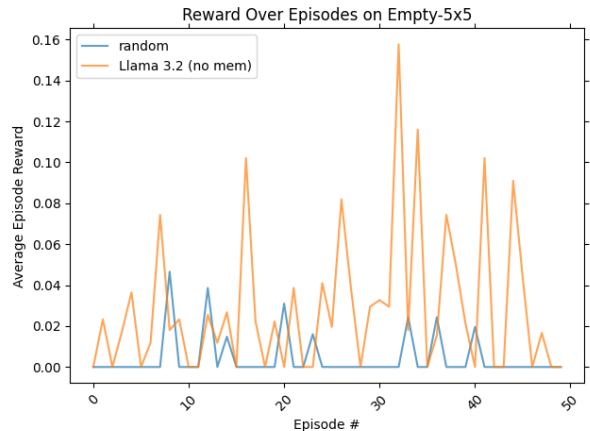
In this section we present the evaluation of the LLM-RL Agent and LLM-RL Memory Agent.

The LLM is expected to have significant reasoning capabilities and out of the box already contains substantial contextual

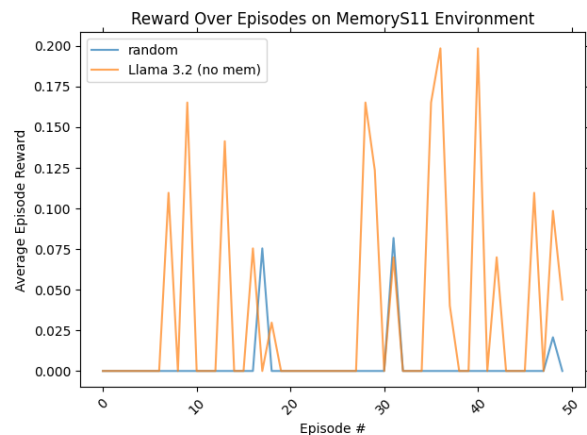
information. For this reason, we run most of our LLM-RL Agent experiments for 100 episodes with each episode terminating after a maximum of 50 steps or upon reaching the goal. We do not fine tune the LLM model and so we expect near maximum performance right away.

A. Context Free LLM-RL Agent

We first evaluate a context free LLM-RL Agent on a couple of simple mini-grid domains; Empty and MemoryS11. We use this to verify our basic implementation and to establish functional prompting for the policy of the LLM-RL Agent.



(a) Empty



(b) Memory

Fig. 3: Mean reward per episode of the LLM-RL Agent for Empty and MemoryS11 environments.

We see in Figure 3 that the LLM-RL Agent is able to outperform the random baseline which indicates some ability to infer the correct action from the ample text context provided by the observations. We point out that the Agent does not *learn* over the course of the experiment because each LLM query is fully independent and no context is saved for this simplified agent.

B. Prompt Testing

Prompt design is critical to shaping the behavior of the base LLM. Therefore, we conduct a series of prompt tests. We use *llama3.2* as the LLM and test on the MiniGrid Empty environment.

We attempt to control the behavior of the agent with prompts in the agent config. We start by including all the necessary information in the prompt, including state, context, legends, action list, navigation strategy, additional notices, and template for responses.

We then attempt several ways to optimize the prompt. We first try to give explicit examples of actions to take in specific states. For instance, we included:

Movement Examples:

- 1: Goal is in a single direction (up, down, left, or right)
-> Turn to face the goal and move forward
- 2: Goal lies between multiple directions (e.g., down-right, up-left)
-> Choose either adjacent direction, turn toward it, and move forward
- 3: You are facing right and there is a wall to your right
-> Do not move forward. First turn toward a direction where the path is clear and then move forward

Although this clarifies most of the common scenarios, the agent’s responses got worse and the agent seems to be confused about the commands. Specifically, the agent always gets stuck in the first square and keeps changing directions without moving forward.

Another attempt we did is hardcoding failure scenarios directly in the prompt. For instance, we find that sometimes the agent will go beyond the grid boundary even though we explicitly say “If blocked, find an alternate route around obstacles” in the prompt. Therefore, we try to be even more detailed and include a specific example in the prompt:

When you face a wall ahead, do not take action 2 and turn to a new direction

This does not make a big difference in terms of the rewards: the agent will still make the same mistakes for similar scenarios.

We also attempt to replace some mild language with imperative tone, which in general works better for LLMs. For instance, for an original prompt like:

Navigation Strategy:

1. First, determine if you need to turn to face the goal
2. Once facing the right direction, move forward if the path is clear
3. If blocked, plan a route around obstacles

We replace it with:

Navigation Strategy:

- Turn to face the goal
- Move forward when the path is clear
- If blocked, find an alternate route around obstacles

This works to some extent. Agent with the original prompt gets stuck around episode 10, while agent with this modified prompt has reasonable rewards up until around episode 45. Undoubtedly, hallucination is still a big problem here. Specifically, the LLM is drawing from unrelated prior training knowledge (e.g., blending in assumptions from general navigation, robotics, or text-based games). Also, the prompt does not strictly constrain the context so that the LLM sometimes “free-associates” beyond what is present. Additionally, there is no grounding mechanism filtering or validating whether the reasoning matches the environment.

C. LLM-RL Mistakes and Hallucinations

In general, we classify the LLM-RL Agents’ mistakes into three categories: partially correct, completely wrong, and hallucinations. For each category, we present an example failure case paired with the LLM-RL Agents’s reasoning.

Partially correct example:

```
[State]
# # # # #
# . . > #
# . . . #
# . . G #
# # # # #
```

[Response] action=2 reasoning=”Since the goal ‘G’ is directly in front of me, I will move forward to advance one square in my current direction (i). This action assumes that there are no obstacles between me and the goal, which appears to be the case based on the provided grid layout.”

Completely wrong example:

```
[State]
# # # # #
# . . ^ #
# . . . #
# . . G #
# # # # #
```

[Response] action=2 reasoning=”I choose to move forward. Since my goal is directly ahead of me, moving forward will take me closer to the green goal square marked as ‘G’ while also avoiding unnecessary turns and potential collisions with walls.”

Hallucination example:

[State]

```
# # # # #  
# . . . #  
# . . . #  
# v . G #  
# # # # #
```

[Response] action=1 reasoning="Since we are currently facing down and have already moved left, we should turn right to face up. This will allow us to move forward towards the goal when it's clear of obstacles. The human inhabitant is present within 3 meters, but since we don't know its exact position yet, turning now allows us to get closer to the goal by minimizing potential encounters with the human."

Based on these prompts, we conduct further experiments and evaluations on the LLM-RL agent.

D. LLM-RL Memory Reflection Agent Mistakes and Hallucinations

It is of note that the mistakes of the LLM-RL Memory-Reflection Agent fall into similar categories when considering its reflection on its actions: partially correct or hallucinations. Partially correct answers mean the agent has learned something about the setup or the task, but draws a wrong conclusion. Hallucinations include answers with components that were not in the game or with irrelevant advice.

Partially correct example:

"Turn left when approaching a T- junction."

Although the agent does not understand that it must path plan differently for each setup, it has gathered a few key concepts, such as obstacles and turning.

Hallucination example:

"Keep an eye on your fuel level and plan accordingly."
"Use available resources (e.g. health, ammo) efficiently."

E. LLM Model Testing

We evaluate the LLM-RL Memory Agent on Empty, DoorKey, Go-to-Object, and Memory MiniGrid environments for two versions of our prompting: regular prompting and imperative tone prompting. We additionally test the performance of different base LLM models. We show the performance of the *llama3.2*, *codellama*, and *gemma2* LLMs and compare against the context free LLM-RL Agent along with a random baseline.

As we can see in Figure 4 that prompting without imperative tone performs poorly. For Go-to-Object, DoorKey, and Memory environments we see all the models achieve no reward over the course of training. These additional results are included in Appendix Figures 18 and 19.

We similarly test the LLM-RL Memory Reflection Agent on Empty, DoorKey, Go-to-Object, and Memory MiniGrid

environments using regular prompting and the same three base LLM models. The LLM-RL Memory Reflection Agent also performs poorly across all base LLM models in the Empty MiniGrid environment, as shown in Figure 5. It shows decreased performance when compared to the LLM-RL Memory Agent, with no reward across all three base LLMs for the Memory S11 environment, as shown in Appendix Figure 24. Similarly to the LLM-RL Memory agent, the Memory Reflection Agent achieves no reward over the course of training on more complex environments like Go-to-Object and DoorKey. These results are included in Appendix Figures 22 and 23.

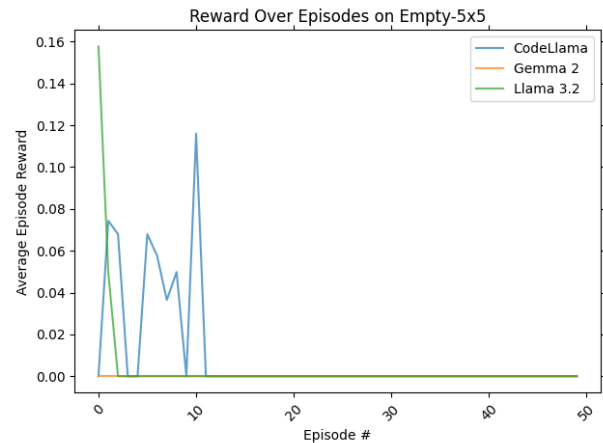


Fig. 4: Average Reward for LLM-RL Memory Agent with regular prompting for Empty MiniGrid environment

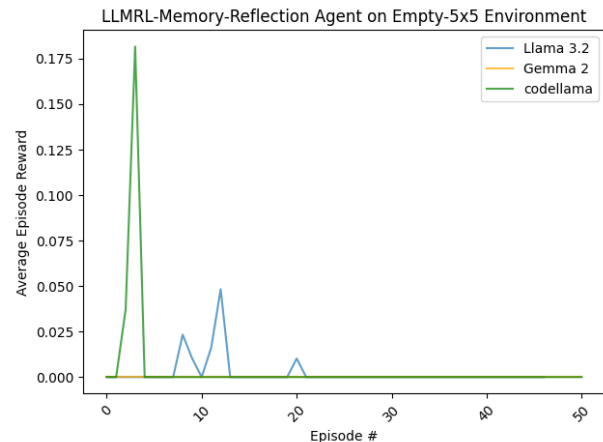


Fig. 5: Average Reward for LLM-RL Memory Reflection Agent with regular prompting for Empty MiniGrid environment.

For both the LLM-RL Memory Agent and the Memory Reflection Agent, the empty environment we see the best performance from the *codellama* model, but it is unable to achieve consistent performance and falls to zero reward later in training. We see somewhat better performance on the LLM-RL Memory Agent when using imperative tone prompting.

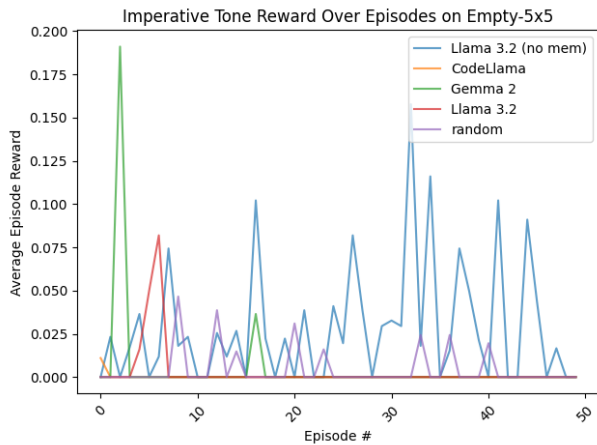


Fig. 6: Average Reward for LLM-RL Memory Agent with imperative prompting for Empty MiniGrid environment

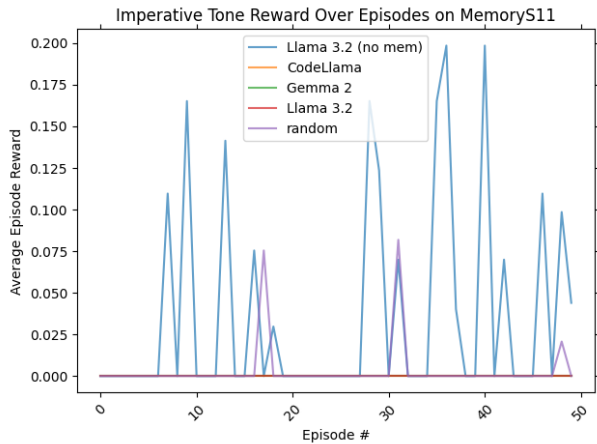


Fig. 7: Average Reward for LLM-RL Memory Agent with imperative prompting for MemoryS11 MiniGrid environment

For imperative tone prompting tests on the LLM-RL Memory Agent we additionally show the context free LLM-RL Agent and the random baseline. The results are plotted in Figures 6, and 7. For Go-to-Object and DoorKey environments we see all agent variants achieve no reward over the course of training. These additional results are included in appendix Figures 20 and 21.

As we would expect, we see generally better performance on the Empty and MemoryS11 environments when using imperative tone but surprisingly, the best performance is achieved by the context free LLM-RL agent. Based on these results, the additional information stored in the agent context was in fact detrimental to overall performance. This implies that the information that the LLM is able to extract from past experiences mainly serves to mislead future actions. This is a reflection of the LLM’s inability to generalize individual experiences into a cohesive policy or plan for future interactions.

For more complex environments like DoorKey and Go-to-Object we see zero reward from all agents including random.

The Go-To-Object environment requires the agent to take the *done* action in a particular position and it is very easy to fail this by taking the done action at the wrong time or by walking past the desired location. Similarly, the DoorKey environment requires a narrow and specific sequence of actions for the agent to succeed. In both cases, the agent needs the ability to plan a long sequence of steps and avoid failure. From our results it is clear that the LLM-RL Memory Agent and the LLM-RL context free agent cannot achieve this level of reasoning.

V. BASE LINE COMPARISONS

To evaluate the LLM-RL agent, we designed an experiment comparing its performance against Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) models. For each environment, DoorKey, Empty, Go-to-Object, and MemoryS7, we ran both baseline and optimized versions of the PPO and DQN agents, providing structured comparison points for assessing the LLM-RL agent’s effectiveness.

A. Baseline

To obtain a baseline comparison value for DQN and PPO models, we utilized the documentation of Stable-Baselines-3 (SB3).

SB3 offers tested, efficient implementations of different RL algorithms. We chose to use SB3 because its agents have been evaluated across multiple environments to ensure reproducible results and have been consistently used in precedent work as a starting point for comparison. Additionally, each agent comes with well-tuned hyperparameters that deliver reasonable performance across different tasks, making SB3 a reliable baseline for comparison.

We used the same hyperparameters as SB3 outlines for the DQN and PPO agents in the four environments. We looked at the agent’s performance across 1 million timesteps to use as a basis of comparison with our experimental hyperparameter tuning, referred to as Skyline. These results are shown in Figures 8, 25, 9, and 26 and in the Appendix section VIII-E.

The results indicate that in both the Go-to-Object and DoorKey environments, the baseline DQN and PPO agents perform significantly better than the LLM-RL Memory Agent. In the Empty, environment, the DQN baseline model performs better after 200000 timesteps compared to the LLM-RL Memory agent’s best model.

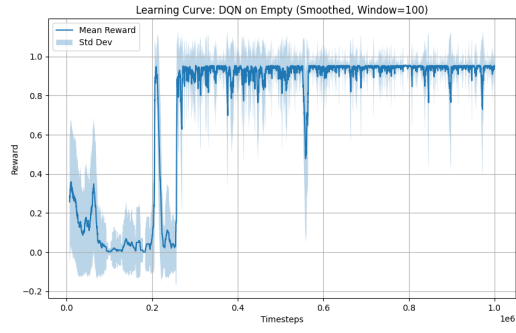
B. Skyline Hyperparameter Tuning and Results

To identify the optimal performance of the DQN and PPO models across multiple environments, we conducted an extensive hyperparameter search, referred to as the *Skyline* analysis. For DQN we varied `learning_starts` and `buffer_size`; for PPO we varied `learning_rate`, `n_epochs`, and `gae_lambda`. All searches were run on the same four environments.

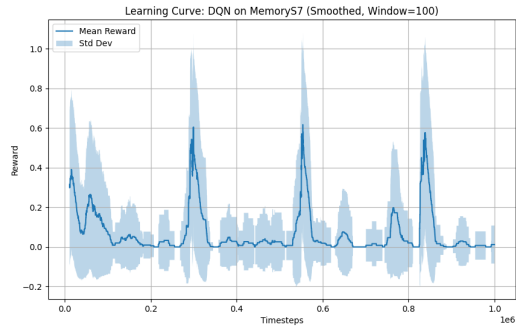
DQN Grid Search Results:

PPO Grid Search Results:

For each environment, Figures 10, 11, and 27 show the DQN learning curves; Figures 12 and 28 show the PPO curves.



(a) Empty



(b) MemoryS7

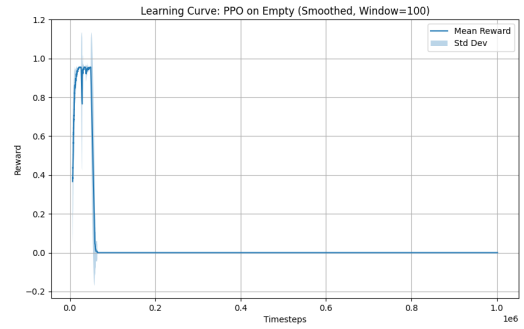
Fig. 8: Mean reward progression of the baseline DQN model from SB3 across training iterations for Empty and MemoryS7 environments.

Environment	Learning Starts	Buffer Size	Mean Reward
DoorKey	10000	500000	0.97
Empty	10000	50000	0.96
Go-to-Object	10000	500000	0.10
MemoryS7	50000	500000	0.05

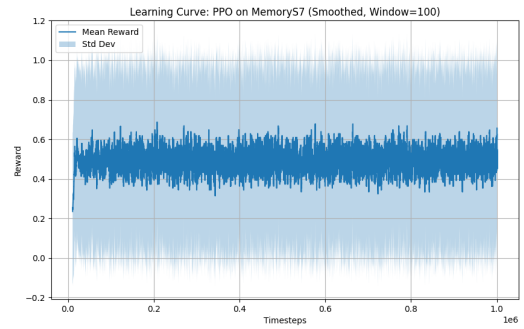
TABLE I: Optimal DQN hyperparameters and mean rewards per environment.

Environment	Learning Rate	n_epochs	Mean Reward
DoorKey	3e-05	20	0.9627
Empty	3e-05	20	0.9550
Go-to-Object	3e-05	20	0.7360
MemoryS7	3e-05	20	0.4960

TABLE II: Optimal PPO hyperparameters and mean rewards per environment.



(a) Empty



(b) MemoryS7

Fig. 9: Mean reward progression of the baseline PPO model from SB3 across training iterations on MemoryS7 and Empty environments.

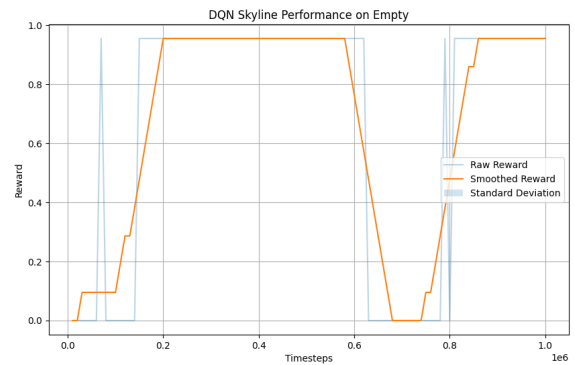


Fig. 10: DQN mean reward over training for Empty environment.

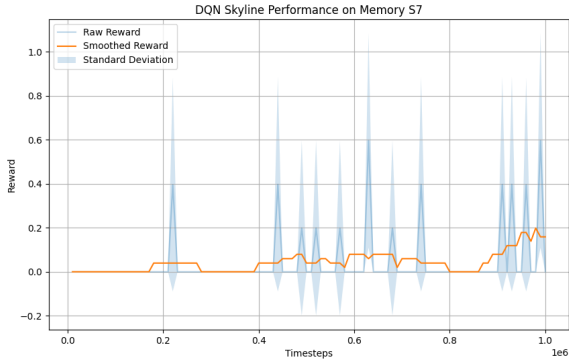
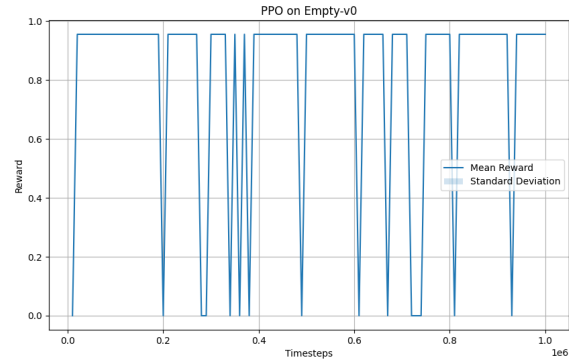
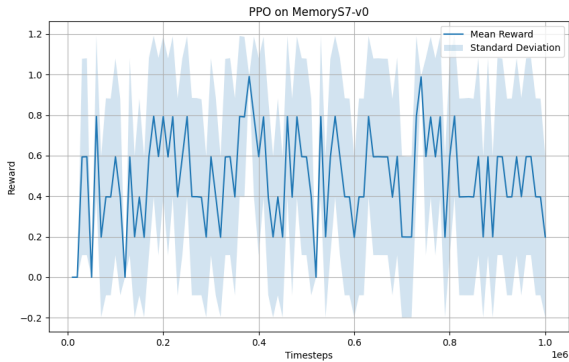


Fig. 11: DQN mean reward over training for MemoryS7 environment.



(a) Empty



(b) MemoryS7

Fig. 12: PPO mean reward over training for Empty and MemoryS7 environments.

With the results for each environment, we can then compare the LLM and random agents’ performance across the same timesteps for DQN in Fig 13a and Fig 13b. What we see in these Figures is that the LLM achieves higher performance than the DQN agent in the early timesteps. In fact, the DQN achieves a mean reward of 0 across the LLM’s timesteps.

We also compare the LLM and random agents’ performance across the same timesteps for PPO in Figures 14 and 15. We see that the LLM achieves higher performance than the PPO agent in the early timesteps.

C. K-Fold Training and Results

To evaluate the generalization capabilities of traditional RL algorithms across multiple environments using a singular policy, similar to the LLM-RL agents, we implemented a k-fold training approach. Unlike the single-environment training in our skyline experiments, this method trains a singular policy across multiple environments with the same state and action spaces sequentially, allowing us to assess the agent’s ability to transfer knowledge between tasks.

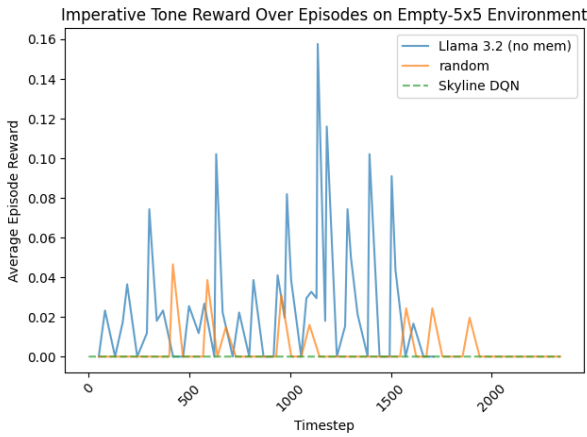
To train these agents, we used the same environments as used for our previous experiments: DoorKey, Empty, Go-to-Object, and MemoryS7. The agent was trained by sampling episodes in each environment sequentially, with an environment switch occurring every 10 episodes. This ensured that the agent received balanced exposure to all tasks. To evaluate the agent, we periodically tested its performance in the environments on new instances, giving a benchmark to track learning progress during training.

We found that the optimal hyperparameters for the k-fold agent training differ from those identified in our skyline single-environment experiment, though there was little success on some environments. Mainly, the size and exploration of the agent was increased to allow for the larger required context. Table III summarizes the best hyperparameters for our k-fold experiments.

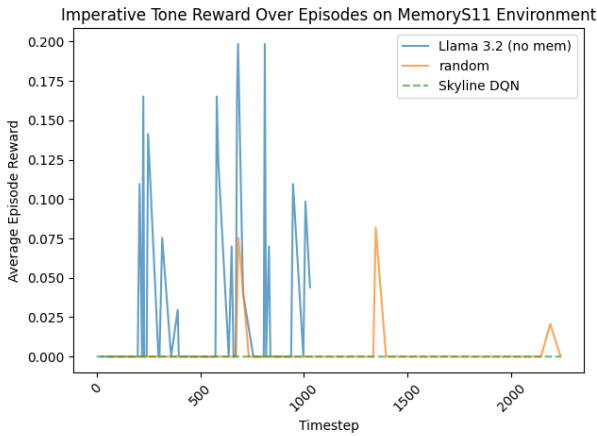
The results for the DQN model are seen in Figure 16 and the results for the PPO are seen in Figure 17. DQN demonstrates poor stability, performing the best on the Empty environment. It occasionally receives non-zero reward from other environments, but it generally performs sporadically and is unable to learn across multiple environments. PPO is able to learn Empty quickly, and it learns DoorKey shortly after. While it performs poorly on DoorKey at times, it is generally able to maintain its performance on both Empty and DoorKey. On Go-to-Object

TABLE III: Hyperparameters for K-Fold Cross-Environment Training

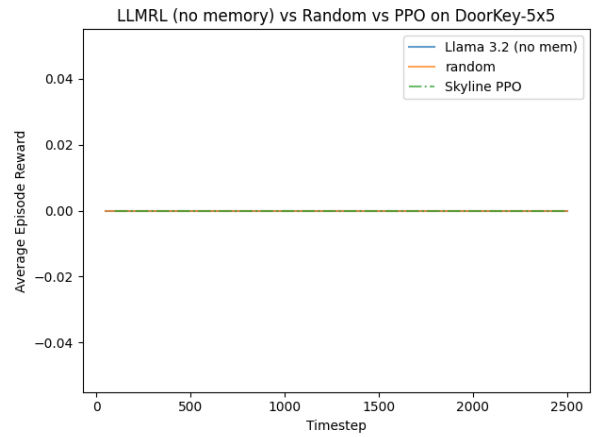
Parameter	PPO	DQN
n_steps	2048	-
batch_size	64	-
n_epochs	8	-
ent_coefficient	0.01	-
buffer_size	-	500000
learning_starts	-	50000
exploration_fraction	-	0.3



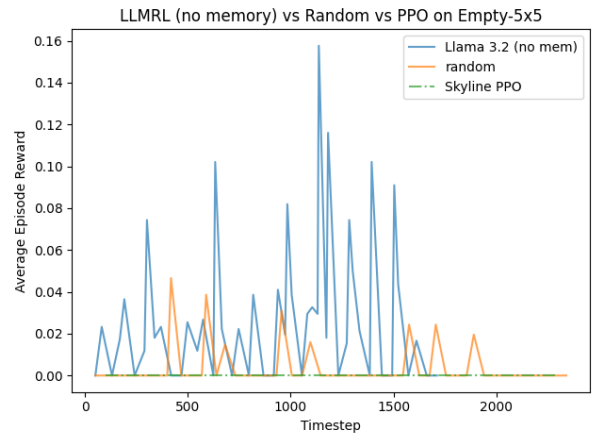
(a) Empty



(b) MemoryS11



(a) DoorKey



(b) Empty-5x5

Fig. 14: Comparison of LLM-RL (no memory), random agent, and Skyline PPO baseline on DoorKey and Empty MiniGrid environments.

and Memory, it gets non-zero reward unlike DQN, however, it is unable to ever stabilize or receive a high reward.

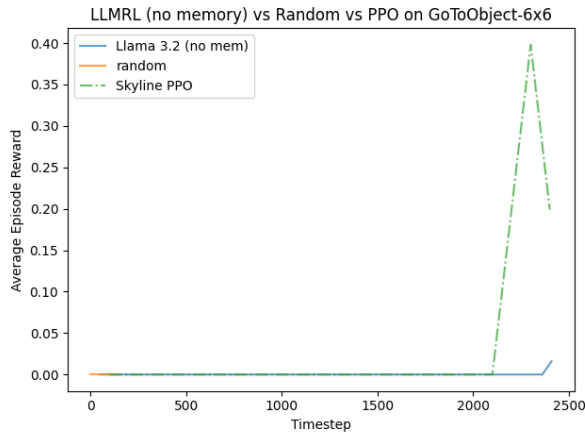
VI. CONCLUSION

In this report, we introduced LLM-RL, a novel framework exploring the feasibility of utilizing out-of-the-box LLMs as policy generators for reinforcement learning agents in text-based environments. We make three core contributions. The first is a structured prompting mechanism that translates environment observation and action spaces into a natural language format. The second is a memory component used to enhance LLM context retention. The third is a feedback loop wherein the agent explicitly reflects on its performance and provides itself with a list of "rules" to follow.

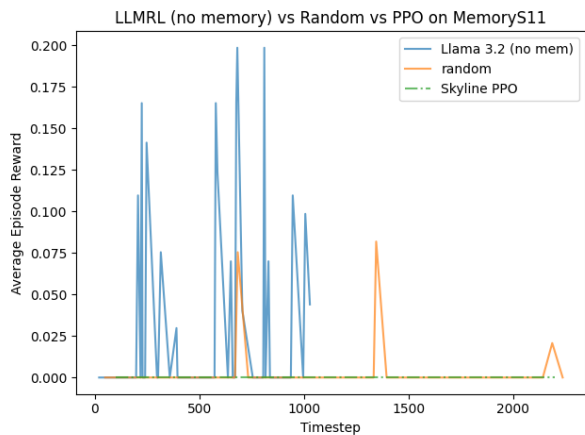
Our evaluations across various MiniGrid environments revealed several challenges with our approach. While the framework successfully enabled interaction with these environments, the LLM-RL Agent struggled considerably. In simple domains the LLM-RL Agent could outperform random but failed to complete more complex tasks requiring sequential planning and interaction with objects. Prompt engineering, including the use of imperative tones and explicit examples, yielded only marginal improvements, failing to overcome issues like

hallucination and nonadherence to the environments rules. Furthermore, the tested memory augmentation mechanism did not contribute positively to the performance, nor did the reflection mechanism. This suggests that the LLM-RL Agent's ability to leverage past experiences for improved decision making (either implicitly or explicitly) is limited with 10B parameter models. While traditional RL methods like DQN and PPO perform similarly or worse for a small set of examples on individual tasks, their performance eventually surpasses LLM approaches in the long run. This highlights a current gap in task-specific performance between natural language and learned policy-based approaches.

We find that there are still significant limitations to using pretrained LLMs as direct, generalist policy generators in RL tasks. While LLMs possess remarkable language understanding and reasoning capabilities, translating these into effective action sequences in interactive environments, without the need for task-specific training, remains a significant challenge. Leveraging the inherent knowledge within pretrained LLMs



(a) Go-to-Object



(b) MemoryS11

Fig. 15: Comparison of LLM-RL (no memory), random agent, and Skyline PPO baseline on Go-to-Object and Memory environments.

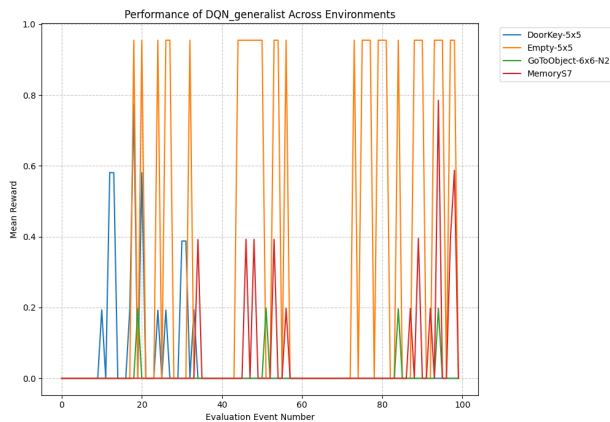


Fig. 16: Periodic reward progression of generalist DQN model across environments and training episodes.

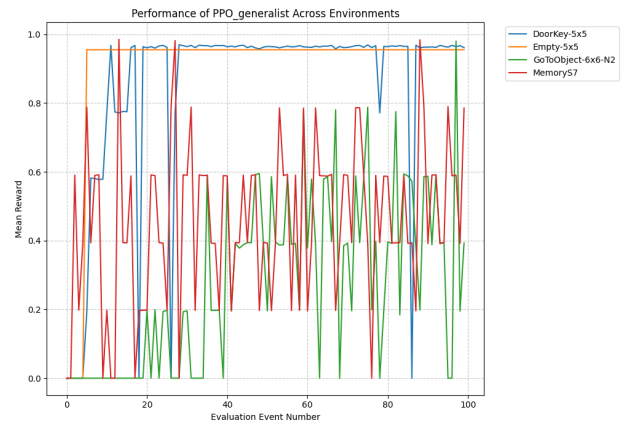


Fig. 17: Periodic reward progression of generalist PPO model across environments and training episodes.

for complex sequential decision-making may require more robust grounding mechanisms to be successful.

Despite the underwhelming performance of our agent, LLM-RL conceptually bridges the agent-, language-, and cognition-centric approaches to AI. Our findings suggest that the generation of LLMs used is ill-suited to meet long-term AGI goals, particularly without enhanced grounding and reasoning. The inability of these agents to retain context, reason over time, or plan must be addressed before LLM-based systems can serve as the backbone of AGI.

VII. FUTURE WORK

In our experiments, we found the LLM-RL Memory agent to be ineffective at summarizing experiences to inform future decisions. While this result does suggest that LLMs are not well suited for complex reasoning, it doesn't mean it is not possible to achieve with carefully tuned prompts, as we see in existing literature [14]. It is possible that a more powerful LLM model along with careful prompt and system engineering could result in an agent with sensible memory updates. Testing a more structured approach than our implementation could provide better insight into the real limit of LLMs in the RL setting.

Future directions for this exploration of the LLM-RL agent include extending the evaluation to a broader range of environments to identify tasks where an LLM-based agent may be more effective. Specifically, future works may include testing the agent on continuous domains requiring reasoning about spatial relations and dynamics, as well as image-based domains using out-of-the-box image labeling to improve the agent's perceptual capabilities. Additionally, testing a scaled up version of the LLM-RL Agent would aid in assessing whether there are emergent abilities as model size increases. This scaling would involve utilizing larger and more recent LLM models with increased parameter counts and using models with enhanced reasoning capabilities. This would allow us to perform tests such as feeding the entire agent history to the LLM at each time step, typically exceeding prompt

limits. Additionally, larger models would allow us to test both larger memory sizes and more advanced memory compression techniques for the LLM-RL Memory Agent.

However, perhaps the most important direction for future work involves fine-tuning the base LLM using reinforcement learning (RL), as recently demonstrated by approaches such as DeepSeek-R1 [4]. This could directly adjust the model’s internal representations, aligning them more closely with sequential decision-making dynamics inherent to RL tasks. Specifically, RL fine-tuning could reduce hallucinations and improve context retention by explicitly reinforcing accurate, task-specific responses.

Yet, a critical consideration is whether such RL fine-tuning would cause the LLM to overfit to specific domains, potentially sacrificing its broad pretrained world knowledge. An essential open question is the extent to which RL-based fine-tuning can preserve generalizable knowledge acquired during pre-training. If successful, this approach could result in a versatile, general-purpose system capable of adapting to diverse real-world tasks beyond the specific training domains. Evaluating the generalization capability of such RL-fine-tuned LLMs remains an important area for future exploration.

REFERENCES

- [1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Daniel Choi, Helen Chu, Brian Coates, Alex Dai, Chelsea Finn, Henry Fu, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.
- [2] Lili Chen, Kevin Lu, Aravind Rajeswaran, Harrison Lee, Aditya Grover, Anuj Kumar, and Pieter Abbeel. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.
- [3] Joseph Clinton and Robert Lieck. Planning transformer: Long-horizon offline reinforcement learning with planning tokens. *arXiv preprint arXiv:2409.09513*, 2024.
- [4] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.
- [5] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, and Timothy Lillicrap. Generalization of reinforcement learners with working and episodic memory. *arXiv preprint arXiv:1910.05793*, 2019.
- [6] Karl J Friston, Thomas Parr, and Bert de Vries. Active inference: A process theory. *Neural computation*, 33(2):313–364, 2021.
- [7] Danijar Hafner, Jost Tobias Springenberg, Kurtland Chua, Michael Tschannen, Gellért Munk, Cecilia Pinneri, Yuval Tassa, Lars Buesing, and David Silver. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.
- [8] Wenlong Huang, Aviral Singh, Jiajun Fu, Sean Sohn, Yilun Wu, Dorsa Sadigh, Fei-Fei Li, Aniruddh Thomas, and Jiajun Wu. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2206.10498*, 2022.
- [9] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.
- [10] Ezgi Korkmaz. A survey analyzing generalization in deep reinforcement learning. *arXiv preprint arXiv:2401.02349*, 2024.
- [11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fiedland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dhharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.
- [12] Rafael Figueiredo Prudencio, Marcos R. O. A. Maximo, and Esther Luna Colombini. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *arXiv preprint arXiv:2203.01387*, 2022.
- [13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.
- [14] Jingbo Shang, Zai Zheng, Jiale Wei, Xiang Ying, Felix Tao, and Mindverse Team. Ai-native memory: A pathway from llms towards agi, 2024.
- [15] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.
- [16] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023.
- [17] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King’s College, Oxford, 1989.
- [18] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory, 2023.

VIII. APPENDIX

A. Code and Reproducibility

Most of the code to run experiments, gather data, and generate plots can be found at: <https://github.com/alexanderivanov2424/LLMRLAgent>. Additionally, some experimental data is saved in the `experiment_data` directory which allows reproducing some of the plots without rerunning the experiments themselves.

B. LLM-RL Memory Agent Regular Prompt Tests

Here we have additional results for the DoorKey and Go-to-Object environments. We see that using regular prompting, without imperative tone, results in complete collapse and no learning.

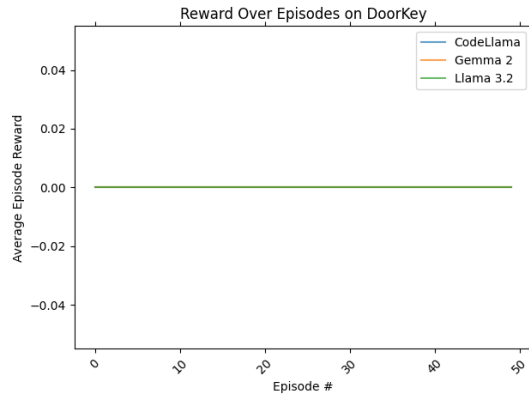


Fig. 18: Average Reward for LLM-RL Memory Agent with regular prompting for DoorKey MiniGrid environment

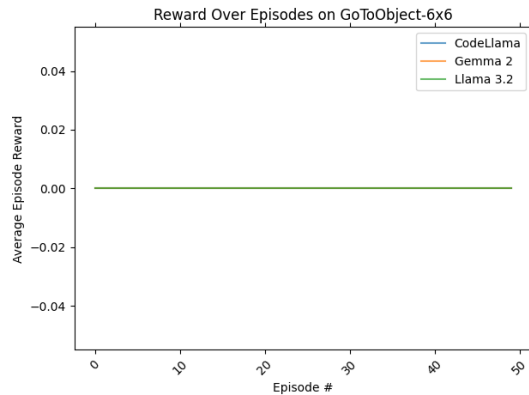


Fig. 19: Average Reward for LLM-RL Memory Reflection Agent for Go-to-Object MiniGrid environment.

C. LLM-RL Memory Agent Imperative Prompt Tests

Even with imperative tone prompting we still see 0 return across all LLM models for the DoorKey and Go-to-Object environments. The no context LLM-RL Agent does have a single episode in the Go-to-Object environment where it does successfully achieve the goal but, we believe this is essentially a random occurrence based on our data.

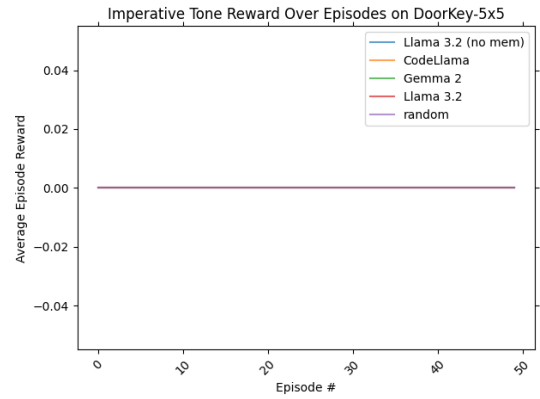


Fig. 20: Average Reward for LLM-RL Memory Agent with imperative prompting for DoorKey MiniGrid environment

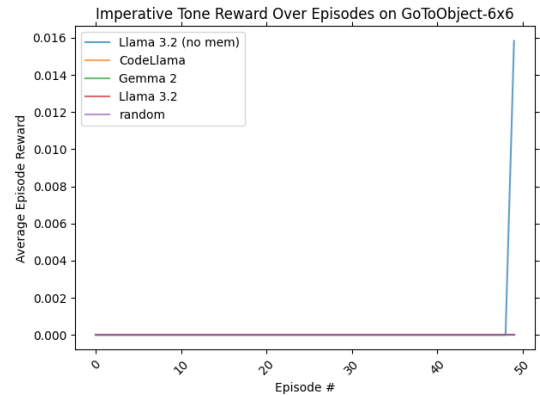


Fig. 21: Average Reward for LLM-RL Memory Agent with imperative prompting for Go-to-Object MiniGrid environment

D. LLM-RL Memory Reflection Agent Results

In figures 22, 23, and 24 we have additional results for the DoorKey, Go-to-Object, and Memory environments. We see complete collapse and no learning in environments with more complexity than the Empty 5x5 MiniGrid.

E. Baseline Agent Performance

In Figure 25, we have demonstrated the baseline DQN agent performance in the DoorKey and Go-to-Object environments. Figure 26 exhibits the baseline PPO agent performance in these same environments.

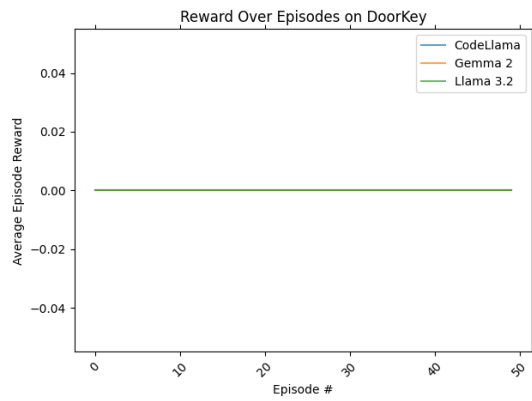


Fig. 22: Average Episode Reward for LLM-RL Memory Reflection Agent with on the DoorKey MiniGrid environment. Across all three LLMs, the agent never reaches the reward.

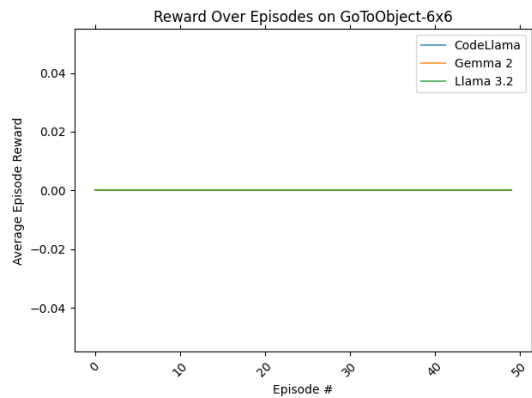
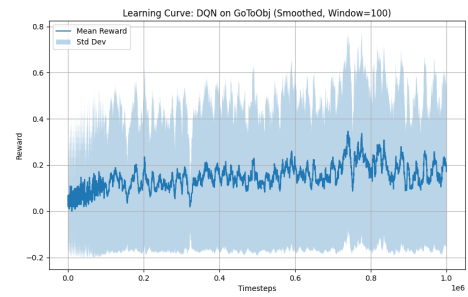


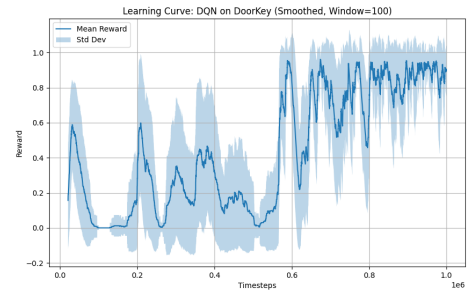
Fig. 23: Average Reward for LLM-RL Memory Agent with regular prompting for Go-to-Object MiniGrid environment. Across all three LLMs, the agent never reaches the reward.



Fig. 24: Average Reward for LLM-RL Memory Agent with regular prompting for the Memory S11 MiniGrid environment. Across all three LLMs, the agent never reaches the reward.

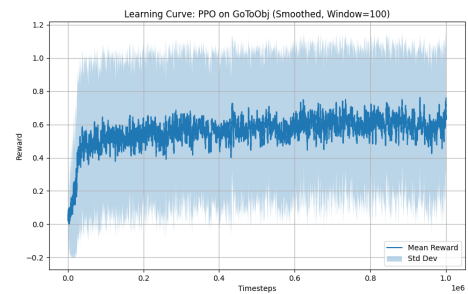


(a) Go-to-Object

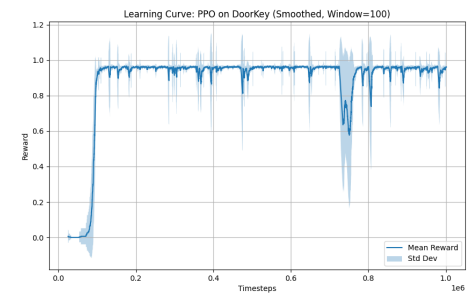


(b) DoorKey

Fig. 25: Mean reward progression of the baseline DQN model from SB3 across training iterations for Go-to-Object and DoorKey environments.



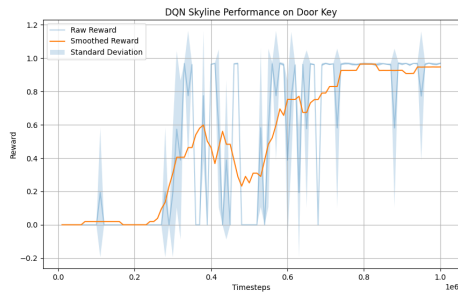
(a) Go-to-Object



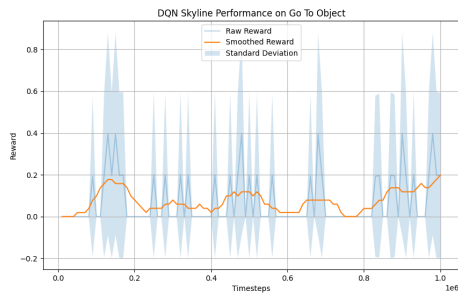
(b) DoorKey

Fig. 26: Mean reward progression of the baseline PPO model from SB3 across training iterations on Go-to-Object and DoorKey environments.

F. Skyline Agent Performance

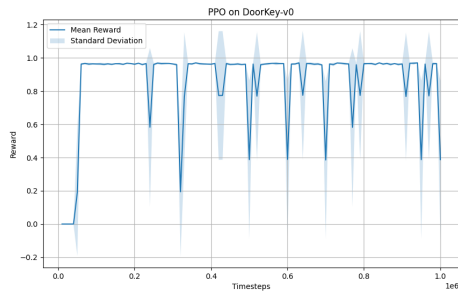


(a) DoorKey

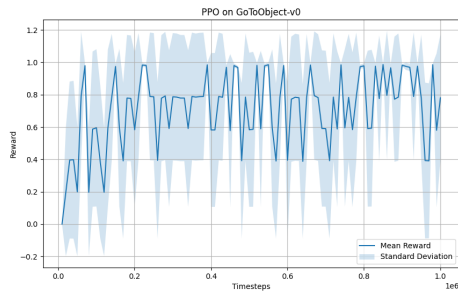


(b) Go-to-Object

Fig. 27: DQN mean reward over training for DoorKey and Go-to-Object environments.



(a) DoorKey



(b) Go-to-Object

Fig. 28: PPO mean reward over training for DoorKey and Go-to-Object environments.

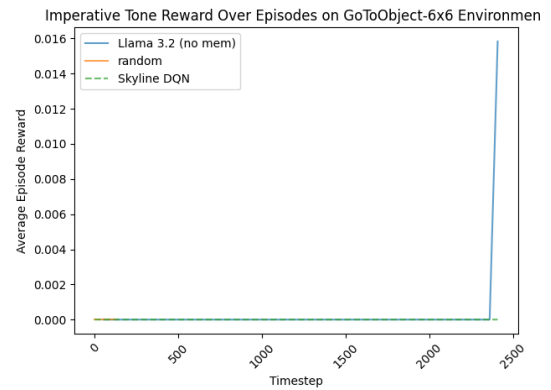
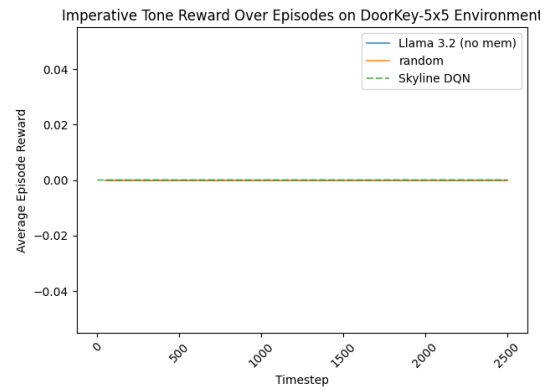


Fig. 29: Mean reward comparison between LLM, random, and DQN agent for DoorKey and Go-to-Object environments.