Large Language Model for Text-based Decision Making

Juang Ian Chong
A0304474H
e1373562@u.nus.edu

Shao Fu Low
A03005009R
e1374097@u.nus.edu

Liao Hung Chieh
A0304830N
e1373918@u.nus.edu

Nontaphat Charuvajana
A0309391B
e1415345@u.nus.edu

Abstract

Deep Reinforcement Learning (DRL) has achieved impressive successes, yet struggles with sample inefficiency, poor generalization, and limited interpretability. Conversely, large language models (LLMs) excel in reasoning, making them promising for general decision-making and planning. This work presents LLM-Zero, a training-free framework to enhance LLMs' performance in planning tasks. We evaluate LLM-Zero against leading LLM-based and traditional DRL methods, demonstrating its competitive potential without additional training. These findings highlight LLM-Zero as a promising approach to integrate LLMs into planning and decision-making tasks.

1 Introduction

Deep learning has driven significant advancements in reinforcement learning, yet current deep reinforcement learning (DRL) methods face persistent challenges:

1. **Sample Efficiency**. Despite efforts to improve sample efficiency, DRL methods require substantial trajectory data to train competent agents, imposing high computational costs.
2. **Generalizability**. DRL agents often struggle to adapt to new or modified environments, necessitating costly retraining for each new task.
3. **Training Complexity**. Unstable learning dynamics and extensive hyperparameter tuning hinder scalability to new environments.

These challenges pose a barrier to deploying DRL in real-world applications such as robotic manipulation, autonomous driving, and drug discovery, where generating training data is expensive, and adaptability is crucial. Addressing these issues requires methods that are sample-efficient, generalizable, and capable of handling diverse environments with minimal retraining.

The emergence of large language models (LLMs) provides an exciting avenue for overcoming these challenges. Pre-trained on extensive text corpora, LLMs exhibit unique capabilities that make them well-suited for decision-making tasks:

1. **Commonsense Knowledge**. LLMs can make informed initial decisions, circumventing the inefficient random exploration typical of traditional RL methods.
2. **Generalizable Reasoning**. LLMs can adapt to novel tasks using its reasoning capabilities, showcasing a level of generalizability that is challenging for DRL methods to achieve.

3. **Instruction via Natural Language**. LLMs are inherently responsive to natural language inputs, enabling flexible and intuitive control through human-readable instructions.

These strengths position LLMs as a promising alternative to traditional DRL methods. This work explores their potential in solving *Natural Language Planning Problems* (NLPP), defined as finding an optimal policy in a Markov Decision Process (MDP) where states are expressed in *natural language* and transition dynamics and reward structures are *latent*. The main challenges in NLPP are extracting relevant information from text-based states and inferring latent dynamics and rewards.

The most straightforward approach is to use LLMs directly as decision-making agents, a paradigm referred to as the *LLM-Policy*. However, it suffers from key limitations such as hallucination issues, which make them unreliable for long-term planning tasks. A single incorrect decision can significantly impact overall outcomes, especially in complex scenarios.

To address this, recent work such as LLM-MCTS [1] integrates LLMs with Monte Carlo tree search (MCTS), leveraging tree-based planning to refine decisions and mitigate critical errors. However, LLM-MCTS still has notable limitations:

1. **Environment-Specific Dependency**. LLM-MCTS relies on predefined, environment-specific state transition functions, limiting its applicability in novel environments.
2. **Inaccuracy in Rollouts**. Rollouts with small number of simulations can yield imprecise value estimates, especially in environments with high stochasticity, sparse rewards, or large action spaces.

We propose **LLM-Zero**, a novel framework inspired by MuZero [2], to overcome these limitations. LLM-Zero eliminates the reliance on predefined transition functions by enabling LLMs to predict state dynamics directly. It also reduces inaccuracies in value estimation by leveraging the reasoning capabilities of LLMs, requiring fewer simulations to achieve reliable estimates. To evaluate LLM-Zero, we compare its performance against both LLM-based and DRL-based methods in text-based environments. In summary, our key contributions are as follows:

1. We introduce LLM-Zero, a novel framework that integrates LLMs with tree-based planning, inspired by MuZero.
2. We design and implement a transformer-based DRL network as a baseline for comparison in text-based environments.
3. We conduct a comprehensive evaluation of DRL-based and LLM-based methods for text-based decision-making and planning.

2 Related Work

2.1 Classical Planning and Decision Making

Classical planning focuses on generating action sequences to transition from an initial state to a goal state within a defined environment. Typically, tasks are represented using the Planning Domain Definition Language (PDDL), a standardized framework for modeling planning problems [3]. A notable example in classical planning is the elevator problem, which involves optimizing elevator movements to efficiently handle a sequence of requests. While PDDL is effective for structured planning tasks, its complexity can hinder accessibility for reinforcement learning (RL) applications. Tools like pyRDDLGym [4] enable the translation of PDDL-defined problems, such as the elevator problem, into simplified RL benchmarking environments.

Beyond the elevator environment, modern frameworks like VirtualHome, TextWorld, and ALFWorld offer text-based challenges for RL agents. These environments focus on tasks such as object manipulation and navigation, emphasizing planning, reasoning, and learning high-level policies [5, 6, 7].

2.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a powerful paradigm for solving sequential decision-making problems in complex, high-dimensional environments by integrating reinforcement learning with

deep neural networks. Early breakthroughs began with the introduction of Deep Q-Networks (DQN) [8], which demonstrated human-level performance in Atari games by approximating Q-values using convolutional neural networks. DQN has also been adapted to other environments such as text-based games and has shown similar promising results [9].

Subsequent advancements improved the effectiveness of DQN. Trust Region Policy Optimization (TRPO) [10] and its computationally efficient successor, Proximal Policy Optimization (PPO) [11], introduced stable updates to policy-based methods, making them widely applicable to continuous and discrete action spaces.

More recently, MuZero [2] bridged model-free and model-based approaches by learning latent models of environment dynamics. With these models, it can perform Monte Carlo Tree Search (MCTS) in a value-equivalent environment to find the optimal policy. Building upon these concepts, we propose LLM-Zero which retains a similar architecture but eliminates the need for training.

2.3 Decision Making and Planning with LLMs

Large Language Models (LLMs) have gained attention in planning tasks due to their extensive knowledge base and natural language capabilities. A straightforward approach involves directly querying LLMs for plans, as demonstrated in zero-shot planning methods [12]. While LLMs can generate step-by-step plans without prior examples, their responses often lack accuracy and consistency due to inherent uncertainties. To address this, techniques like ReAct [13] incorporate intermediate reasoning steps, and Reflexion [14] refines planning by learning from previous failures.

LLM-MCTS [1] innovatively combines LLMs with Monte Carlo Tree Search (MCTS). It leverages LLMs to build commonsense world models for initializing simulated states and uses LLMs as policies to guide node expansions and simulations. This approach narrows the MCTS search space by prioritizing high-utility states and actions. However, LLM-MCTS relies on predefined, environment-specific models for state transitions and rollouts, limiting its flexibility in dynamic or unseen environments.

3 Methods

3.1 Deep Reinforcement Learning Network

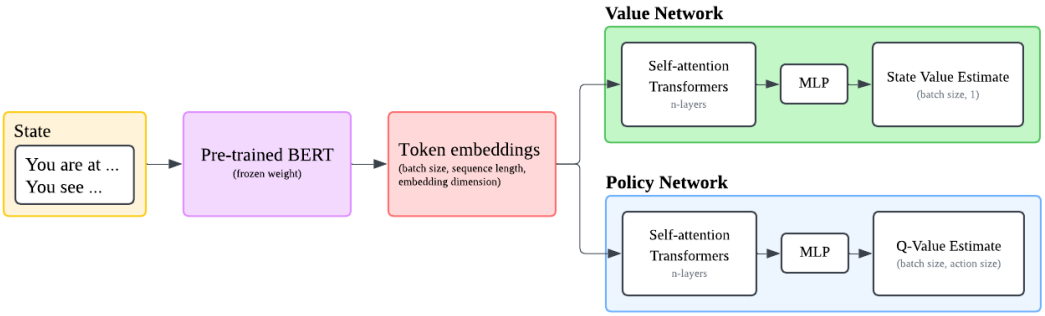


Figure 1: Network architecture of our actor-critic network to be trained with PPO. We only train the policy network for DAgger.

As a baseline for evaluating traditional DRL methods in our environments, we implement our own Actor-Critic network and train it on one of the environments. Most DRL works focus on visual observations and employ convolutional neural networks (CNNs) to extract dense feature representations. However, since our observations are in natural language, we adopt a Transformer architecture with a pre-trained language model as the backbone.

The overall architecture of our network is straightforward. First, we encode the text-based observations using a pre-trained MPNet model [15], which generates high-dimensional token embeddings. These embeddings are then passed through n layers of self-attention transformers to capture task-specific patterns and dependencies. Finally, a fully connected feed-forward network maps the processed

representations to the output, which consists of action Q-values for the policy network (actor) and state utility values for the value network (critic). The weights of the pre-trained model are frozen. This simple design allows us to fine-tune the network to produce environment-specific policies and state values with minimal parameter updates, reducing the computational cost of our experiments.

We train the network using Proximal Policy Optimization (PPO) [11], a reinforcement learning algorithm known for its stable and efficient training performance. PPO iteratively updates the policy by optimizing a clipped surrogate objective, which constrains the magnitude of policy updates to avoid destabilization during training.

In addition to PPO, we employ Dataset Aggregation (DAgger) [16], a straightforward imitation learning method that trains the agent on expert trajectories. This approach simplifies the problem into a supervised learning task, which serves as a controlled baseline to assess whether our network can effectively represent the state and map it to Q-values, independent of the inherent challenges posed by reinforcement learning.

3.2 LLM-Zero

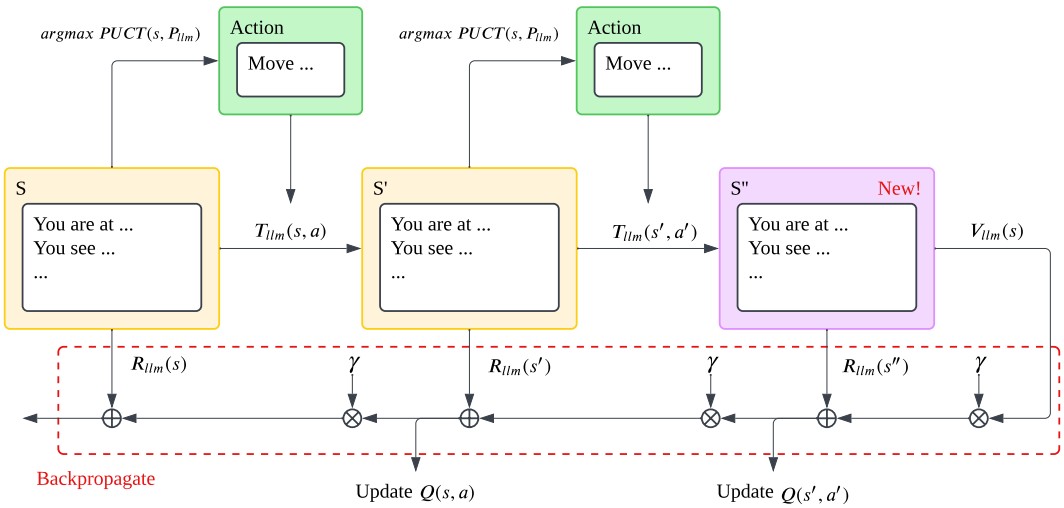


Figure 2: Flowchart of LLM-Zero for a single MCTS simulation. Starting from the root state node, LLM-Zero selects the best action using the PUCT formula, guided by the probability prior from P_{LLM}. The next state and reward are then predicted by T_{LLM} and R_{LLM} respectively. At the leaf node, the state utility is estimated using V_{LLM} and backpropagated to the root node along with the rewards.

As mentioned previously, LLM-MCTS requires environment-specific transition functions to perform state expansion and rollouts in Monte Carlo Tree Search (MCTS). Drawing inspiration from the MuZero framework [2], which integrates deep neural networks to replace multiple MCTS components, we propose LLM-Zero. LLM-Zero harnesses the power of large language models (LLMs) as function approximators to estimate key components of MCTS. Specifically, LLM-Zero employs the following four LLM-based function approximators:

1. **LLM-Policy**. $P_{\text{LLM}}(S) = \pi(a|s) \in \mathbb{R}^{|\mathcal{A}|}$ maps the state S to action probability priors over the action space $\mathcal{A}$. The LLM is prompted to return the optimal action for the given state. Cosine similarities between the text embeddings of the LLM's output and all valid actions are then computed to form an action distribution $\hat{\pi}(a|s)$. Temperature scaling is subsequently applied to $\hat{\pi}(a|s)$ to obtain $\pi(a|s)$, which is used in PUCT. A higher temperature encourages more exploration by reducing the influence of LLM-Policy.
2. **LLM-Transition**. $T_{\text{LLM}}(S, a) = S' \in \mathcal{T}$ predicts the most probable next state S' given a state-action pair (S, a). This is achieved by prompting the LLM with a query such as: "What is the most probable next state given S and a?"

3. **LLM-Reward**. $R_{\text{LLM}}(S) = r \in \mathbb{R}, d \in \{0, 1\}$ estimates the reward r and terminal flag d for the current state S using environment-specific prompts to guide the LLM.
4. **LLM-Value**. $V_{\text{LLM}}(S) = v \in \mathbb{R}$ evaluates the utility of a state S by estimating the expected sum of discounted future rewards from S onward. This is done by querying the LLM with prompts to estimate the sum of discounted future rewards.

In addition to the LLM-Policy already introduced in LLM-MCTS, LLM-Zero incorporates three additional modules to enable MCTS without relying on ground-truth transition functions:

1. **State Node Expansion**. Approximated using T_{LLM} and R_{LLM}, which predict the next state and reward, respectively.
2. **Random Rollout Replacement**. Substituted with V_{LLM}, which directly estimates the utility of a state without requiring simulated rollouts.

In summary, LLM-Zero can be viewed as a hybrid approach, combining elements of *model-free* methods (as represented by LLM-Policy) and *model-based* methods (as represented by LLM-Transition, LLM-Reward, and LLM-Value). A flow diagram illustrating a single simulation loop is provided in Figure 2 for better visualization, and more details of the LLM-Zero algorithm can be found in the appendix. Beyond eliminating the need for ground-truth transition functions, LLM-Zero offers the additional advantage of requiring fewer simulations compared to LLM-MCTS. This reduction is achieved by replacing random rollouts, which typically demand numerous simulations to converge to an accurate value estimate, with V_{LLM}. Since V_{LLM} directly estimates the value of a state with a single query, it enables a significantly more accurate estimate with lower number of simulations. This advantage is demonstrated in the experimental results presented in the next section.

4 Benchmarking Environments and Experiment Setup

4.1 Environment introduction

For all the algorithms discussed above, we conduct experiments in two NLPP environments: the **Elevator** environment and the **ALFWorld** environment.

Elevator environment represents a building with multiple floors, where the agent functions as the elevator. The objective is to maximize the number of passengers delivered to the first floor. At each time step, new passengers may arrive on different floors, adding dynamic complexity and stochasticity to the task. The environment also incorporates constraints, including the maximum number of passengers that can wait on each floor and the elevator's carrying capacity, requiring the agent to make strategic decisions to optimize its performance.

- **State description**:
  1. Number of people waiting in each floor.
  2. The elevator's current location
  3. The elevator's moving direction
  4. Whether the elevator is opened/closed
  5. Number of people in the elevator
- **Action space**:
  1. **Nothing**: The elevator do nothing.
  2. **Move**: The elevator move upward/downward for one floor based on moving direction
  3. **Close door**: The elevator close the door, allowing the elevator to Move.
  4. **Open door**: The elevator open the door, allowing people to get in or leave the elevator.
- **Goal**: Deliver passengers to the first floor efficiently.

ALFWorld environment simulates a household setting with items and objects. The agent can perform a wide range of actions to progress towards completing the task. This environment requires the agent to leverage common-sense knowledge to infer potential object locations and effectively prune unreasonable actions.

- **State description**: A list of items categorized as movable objects, containers, surfaces, or rooms. Each item type has specific attributes that influence the actions that can be performed with or on the item.
- **Action space**:
  1. **Go to <**$item_1$**>**: Go to the location of the item.
  2. **Open <**$item_1$**>**: Open an item, the item can only be a container.
  3. **Close <**$item$**>**: Close an item, the item can only be a container.
  4. **Take <**$item_1$**> from <**$item_2$**>**: Take an item with the agent from a location. <$item_1$> must be a movable object.
  5. **Put <**$item_1$**> in/on <**$item_2$**>**: Put the item with the agent in a container or in a surface. $item_1$ must be a movable object, $item_2$ can be either a container or a surface.
- **Goal**: Move a target movable object to the desired location.

4.2 Differences on the two environments

The primary differences between the two environments lie in the size of their state and action spaces, as well as the length of their time horizons.

In the **Elevator** environment, the state and action spaces are relatively smaller due to the limited number of floors and restricted set of possible actions. However, as the problem involves a continuous process, the time horizon is considerably longer. Conversely, in the **ALFWorld** environment, the state and action spaces are significantly larger, owing to the numerous items being observed and the diverse actions the agent can take on each item. Despite this, the task's ultimate goal is relatively simpler, resulting in a shorter time horizon; in most cases, the agent can reach the goal state within ten steps.

This contrast in the nature of the two environments is a central focus of this project. We aim to compare the results to determine how variations in the size of state and action spaces, as well as the number of time steps, influence the LLM's ability to identify the optimal policy across different model structures.

5 Experiments

5.1 Experimental setup

Environment Settings. Due to budget and time constraints, we evaluate the performance of each method on a single episode for Elevator. The episode is initialized with a fixed seed to ensure reproducibility and fairness. For ALFWorld, all methods are tested on 10 simple pick-and-place tasks, each with a limit of 10 steps.

LLM Models. Experiments utilize a mix of language models, including the open-source **Mistral-7B** [17] and OpenAI's closed-source, pay-per-token **GPT-4** [18]. To ensure fairness, the same LLM model is used for the same task (episode) across all methods.

Hyperparameters. For both LLM-Zero and LLM-MCTS, we set the number of Monte Carlo Tree Search (MCTS) simulations to 50 for the Elevator environment and 20 for ALFWorld. These choices are partly due to budget constraints, but also highlights LLM-Zero's ability to perform effectively with a limited number of simulations. For the PPO baseline, we train the network exclusively on the Elevator environment for 8,000,000 steps, using hyperparameters tuned to the best of our efforts.

5.2 Main Results

Referred to Table 1, LLM-Zero outperforms all baseline methods in both environments, showcasing its effectiveness in natural language planning problems. Based on our observations, LLM-Zero was able to correct mistakes from LLM-Policy by leveraging the its model-based counterparts (i.e. LLM-Transition, LLM-Reward and LLM-Value) to obtain a better estimate of action values via simulated tree search.

Surprisingly, LLM-MCTS did not outperform LLM-Policy and completely failed in the Elevator environment. This outcome is attributed to LLM-MCTS's inability to provide accurate value estimates

during the rollout phase, primarily due to the limited number of simulations. In the Elevator environment, this issue is further exacerbated by its long horizon and inherent stochasticity, which cause rollout values to deviate significantly, leading to incorrect adjustments to LLM-Policy's decisions.

Despite training for 8 million steps with best-effort hyperparameter tuning, PPO underperformed compared to both LLM-Policy and LLM-Zero, underscoring the advantages of LLM-based methods. As we will explore in the next section, this performance gap stems from challenges inherent to PPO learning rather than limitations in the network's capacity.

Table 1: Normalized Scores across each method tested on Elevator and ALFworld environment. [1]

Normalized Score/Methods	PPO	LLM-Policy	LLM-MCTS	LLM-Zero
Elevator	73.55	82.03	13.54	**96.76**
ALFworld	-	60.0	60.0	**70.0**

5.3 Comparing between PPO and DAgger

Table 2 compares the results of training the same network using PPO and DAgger. DAgger achieves near-perfect testing performance, demonstrating the network's ability to effectively learn and generalize from the expert policy. Although DAgger outperforms LLM-Zero, it relies on high-quality expert trajectories, which are often difficult to obtain for novel tasks.

This underscores the inherent challenges of PPO learning compared to supervised learning. While the presented PPO performance could likely be improved with further training and hyperparameter tuning, our key point is that achieving such improvements demands significantly greater effort compared to deploying LLM-based methods.

Table 2: Normalized Scores of PPO and DAgger in the Elevator environment.

Methods	DAgger	PPO
Normalized Score	**99.94**	73.55

5.4 Quality of Prompt

A key factor in all LLM-based methods is prompt engineering. To illustrate this, we compare the performance of LLM-Policy using a well-crafted system prompt against a generic prompt, as shown in Table 3. The high-quality prompt includes detailed descriptions of the environment, the task objective, generic strategies, and a few examples for the LLM to follow. In contrast, the low-quality prompt provides only generic instructions to return the optimal action given a state and valid actions. Full details of each prompt are available in our code base. The results clearly show that the simple prompt failed disastrously, highlighting the critical importance of proper prompt engineering for LLM-based methods.

Table 3: Comparison of prompt engineering quality in the Elevator environment.

Prompt	High Quality	Poor Quality
Normalized Score	**82.03**	2.75

[1] The scores are different from the presentations as we have updated our code to improve performances on ALFWorld. The scores are normalized based on estimated min-max values for better visualization.

6 Discussion and Conclusion

The experimental results demonstrate that LLM-Zero is effective in solving NLPPs, outperforming its predecessor, LLM-MCTS, as well as the vanilla LLM-Policy. Additionally, both LLM-Zero and LLM-Policy surpassed the performance of our PPO actor-critic network, highlighting the potential of LLMs to eventually replace traditional DRL methods. However, LLM-based methods still have notable limitations that need to be addressed:

1. **Computational Cost**: LLM-based methods, particularly LLM-MCTS and LLM-Zero, are computationally expensive as they require multiple inferences per action. This makes them impractical for real-time applications unless LLM inference costs decrease significantly in the future.
2. **Manual Prompt Engineering**: Currently, LLM-based methods rely heavily on well-crafted system prompts or few-shot examples to generate satisfactory outputs. This dependency limits their utility as fully plug-and-play solutions in novel environments. Nevertheless, designing effective prompts is generally less technical and more accessible compared to tasks such as coding transition dynamics or tuning hyperparameters.

These limitations suggest promising directions for future research, such as developing LLM-based frameworks that minimize inference overhead while maximizing performance or creating systems that automatically construct prompts through interaction with novel environments, akin to approaches like ReAct and Reflexion. We hope this project serves as both a foundation and an inspiration for advancing the integration of Large Language Models into general AI planning.

Broader Impact

LLMs are an emerging technology, with their transformative potential reshaping industries ranging from natural language processing to decision-making systems. In this project, we aim to contribute to this promising area by exploring the integration of LLMs with Reinforcement Learning. By combining the reasoning and generative capabilities of LLMs with well-studied Reinforcement Learning techniques, we hope to advance the understanding and application of these technologies to solve complex, real-world problems more effectively.

Acknowledgments and Disclosure of Funding

We would like to acknowledge the researchers and contributors of LLM-MCTS, PyRDDLGym, and ALFWorld for providing the strong foundation that facilitated our research. We also wish to express our sincere appreciation to our lecturer, Leong Tze Yun, and Muhammad Rizki Aulia Rahman Maulana, and teaching assistants, Ma Haozhe who guided us throughout this project.

References

[1] Chen L, Zhou Y, Wu L, Zhao K, Li J, Zhao Z, et al. Large Language Model-Based Monte Carlo Tree Search: A Sample-Efficient Approach to Decision Making and Planning. arXiv preprint arXiv:230514078. 2023. Available from: https://arxiv.org/abs/2305.14078.

[2] Schrittwieser J, Antonoglou I, Hubert T, Simonyan K, Sifre L, Schmitt S, et al. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. Nature. 2020;588(7839):604-9. Available from: https://www.nature.com/articles/s41586-020-03051-4.

[3] Russell SJ, Norvig P. Artificial intelligence: a modern approach. Fourth edition ed. Pearson Series in Artificial Intelligence. Hoboken, NJ: Pearson; 2021.

[4] Taitler A, Gimelfarb M, Jeong J, Gopalakrishnan S, Mladenov M, Liu X, et al.. None, editor. pyRDDLGym: From RDDL to Gym Environments. arXIv; 2024. Available from: https://arxiv.org/abs/2211.05939.

[5] Shridhar M, Yuan X, Côté MA, Bisk Y, Trischler A, Hausknecht M. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In: Proceedings of the International Conference on Learning Representations (ICLR); 2021. p. None. Available from: https://arxiv.org/abs/2010.03768.

[6] Côté MA, Kádár A, Yuan X, Kybartas B, Barnes T, Fine E, et al. TextWorld: A Learning Environment for Text-based Games. CoRR. 2018;abs/1806.11532.

[7] Puig X, Shu T, Li S, Wang Z, Tenenbaum JB, Fidler S, et al.. None, editor. Watch-And-Help: A Challenge for Social Perception and Human-AI Collaboration. arXiv; 2020.

[8] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing Atari with Deep Reinforcement Learning. CoRR. 2013;abs/1312.5602. Available from: http://arxiv.org/abs/1312.5602.

[9] He J, Chen J, He X, Gao J, Li L, Deng L, et al. Deep Reinforcement Learning with a Natural Language Action Space. In: Erk K, Smith NA, editors. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Berlin, Germany: Association for Computational Linguistics; 2016. p. 1621-30. Available from: https://aclanthology.org/P16-1153.

[10] Schulman J, Levine S, Moritz P, Jordan MI, Abbeel P. Trust Region Policy Optimization. CoRR. 2015;abs/1502.05477. Available from: http://arxiv.org/abs/1502.05477.

[11] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:170706347. 2017.

[12] Huang W, Abbeel P, Pathak D, Mordatch I. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. CoRR. 2022;abs/2201.07207. Available from: https://arxiv.org/abs/2201.07207.

[13] Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, et al.. None, editor. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv; 2023. Available from: https://arxiv.org/abs/2210.03629.

[14] Shinn N, Cassano F, Berman E, Gopinath A, Narasimhan K, Yao S. None, editor. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv; 2023. Available from: https://arxiv.org/abs/2303.11366.

[15] Song K, Tan X, Qin T, Lu J, Liu T. MPNet: Masked and Permuted Pre-training for Language Understanding. CoRR. 2020;abs/2004.09297. Available from: https://arxiv.org/abs/2004.09297.

[16] Ross S, Gordon G, Bagnell D. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS); 2011. p. 627-35.

[17] Jiang AQ, Sablayrolles A, Mensch A, Bamford C, Chaplot DS, Casas Ddl, et al. Mistral 7B. arXiv preprint arXiv:231006825. 2023.

[18] Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, Aleman FL, et al. Gpt-4 technical report. arXiv preprint arXiv:230308774. 2023.

Appendix

LLMZero Algorithm

Detailed algorithm for LLMZero implementation.

Algorithm 1: LLM-Zero

Hyperparameters: *number of iterations N, exploration constant c, discount factor γ*
Input: Root state s_0
Output: Optimal action a^* from s_0

Initialize $\mathcal{N}(s) \leftarrow 0, \mathcal{N}(s,a) \leftarrow 0, Q(s,a) \leftarrow 0, P(s,a) \leftarrow |\mathcal{A}|^{-1}, \forall (s,a) \in \mathcal{S} \times \mathcal{A}$
for $i = 1$ **to** N **do**
  $s_{\text{leaf}}, trajectory \leftarrow$ **TraverseTree**(s_0) `// select and expand`
  $v \leftarrow V_{LLM}(s_{\text{leaf}})$ `// query value estimate`
  Backpropagate$(trajectory, v)$ `// backpropagate Q values`
end
return $\arg\max_a Q(s_0, a)$ `// Choose the action with the highest value`
Function `TraverseTree(`s`):`
  $trajectory \leftarrow [s], d \leftarrow false$ **while** $d = false$ **do**
    if $N(s) = 0$ **then**
      $P(s,a) \leftarrow P_{LLM}(v.\text{state})$ `// query prior distribution`
      return $s, trajectory$
    else
      $a \leftarrow \arg\max_a \left(Q(s,a) + c \cdot P(s,a) \sqrt{\frac{\ln \mathcal{N}(s)}{\mathcal{N}(s,a)+1}} \right)$ `// PUCT formula`
      $s \leftarrow T_{LLM}(s,a)$ `// predict next state`
      $r, d \leftarrow R_{LLM}(s)$ `// predict reward and terminal`
      $trajectory$.`append(`(s,a,r)`)`
    end
  end
  return $s, trajectory$
Function `Backpropagate(`$trajectory, v$`):`
  for $i = |trajectory| - 1$ **to** 0 **do**
    $s, a, r \leftarrow trajectory[i]$
    $\mathcal{N}(s) \leftarrow \mathcal{N}(s) + 1, \mathcal{N}(s,a) \leftarrow \mathcal{N}(s,a) + 1$ `// increment count`
    $v \leftarrow r + \gamma \cdot v$ `// update value estimate`
    $Q(s,a) \leftarrow Q(s,a) + \frac{v - Q(s,a)}{\mathcal{N}(s,a)}$ `// update Q value estimate`
  end

LLM Models Used

For this project, multiple LLM models have been utilized to ensure robust and diverse natural language processing capabilities. The following models were employed:

1. **GPT-4o**: A high-performing large language model known for its advanced reasoning, creativity, and context understanding, used for generating and refining complex outputs. This is the best performing model in general but also the most expensive.
2. **GPT-4o-mini**: A lighter version of GPT-4o, optimized for efficiency in scenarios requiring reduced computational cost while maintaining high accuracy.
3. **open-mixtral-8x7b**: An open-weight LLM from MistralAI. The model is hosted on MistralAI's platform and is free for non-commercial use.

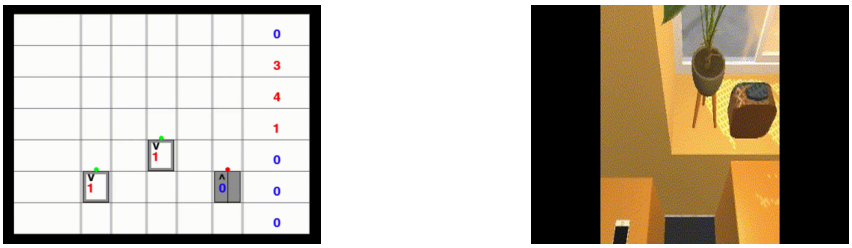
These models were selected for different tasks and sub-tasks based on their difficulties in order to minimize cost while ensuring reasonable performance.

Prompt Engineering

Further information of the prompts that have been specifically construct for both environments can be referred from the following GitHub repository:

https://github.com/spicytomatoes/LLMZero/tree/main/prompts.

Environments representation



(a) Elevator environment (b) ALFworld environmment

Figure 3: Visual representation of (a) Elevator and (b) ALFworld environment.

```
Elevator door is closed.

---------- Step 0 ----------
Enter action: move
Next state:
People waiting at floor 2: 0
People waiting at floor 3: 1
People waiting at floor 4: 1
People waiting at floor 5: 0
Elevator at floor 2.
There are 0 people in the elevator.
Elevator is moving up.
Elevator door is closed.

Reward: 0.0
---------- Step 1 ----------
Enter action: move
Next state:
People waiting at floor 2: 0
People waiting at floor 3: 1
People waiting at floor 4: 1
People waiting at floor 5: 0
Elevator at floor 3.
There are 0 people in the elevator.
Elevator is moving up.
Elevator door is closed.

Reward: -6.0
---------- Step 2 ----------
Enter action: open door
Next state:
People waiting at floor 2: 0
People waiting at floor 3: 1
People waiting at floor 4: 1
People waiting at floor 5: 0
Elevator at floor 3.
There are 0 people in the elevator.
Elevator is moving down.
Elevator door is open.

Reward: -6.0
---------- Step 3 ----------
Enter action:
```

Figure 4: Text-based representation of Elevator Environment

```
 -= Welcome to TextWorld, ALFRED! =-

You are in the middle of a room. Looking quickly around you, you see a bed 1, a desk 1, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a laundryha

Your task is to: put a alarmclock in desk.

valid actions: ['go to bed 1', 'go to desk 1', 'go to drawer 1', 'go to drawer 2', 'go to drawer 3', 'go to garbagecan 1', 'go to laundryhamper 1', 'go
---------- Step 0 ----------
Enter action: go to sidetable 1

Next state:
You arrive at loc 20. On the sidetable 1, you see a alarmclock 3, a alarmclock 2, a alarmclock 1, a creditcard 1, a desklamp 1, a keychain 1, a pen 3,

valid actions:
['examine sidetable 1', 'go to bed 1', 'go to desk 1', 'go to drawer 1', 'go to drawer 2', 'go to drawer 3', 'go to garbagecan 1', 'go to laundryhamper
 1', 'take alarmclock 2 from sidetable 1', 'take alarmclock 3 from sidetable 1', 'take creditcard 1 from sidetable 1', 'take keychain 1 from sidetable
le 1', 'use desklamp 1']

You arrive at loc 20. On the sidetable 1, you see a alarmclock 3, a alarmclock 2, a alarmclock 1, a creditcard 1, a desklamp 1, a keychain 1, a pen 3,
---------- Step 1 ----------
Enter action: take alarmclock 1 from sidetable 1

Next state:
You pick up the alarmclock 1 from the sidetable 1.

valid actions:
['examine alarmclock 1', 'examine sidetable 1', 'go to bed 1', 'go to desk 1', 'go to drawer 1', 'go to drawer 2', 'go to drawer 3', 'go to garbagecan
mclock 1 in/on sidetable 1', 'use desklamp 1']

You pick up the alarmclock 1 from the sidetable 1.
---------- Step 2 ----------
Enter action: go to desk 1
```

Figure 5: Text-based representation of ALFworld Environment

Score Normalization Details

The scores for both environments are normalized using MinMax scaling for better visualization, where

$$Score = \frac{r_{agent} - r_{min}}{r_{optimal} - r_{min}} * 100$$

.

For Elevator, we assume the optimal expected score $r_{optimal}$ to be around 0 and infer minimum score from the random agent which is -4191. For ALFWorld, the optimal score is 10 for completing all tasks and the minimum score is 0 for completing no tasks.