Large Language Models (LLMs) Augmented Planners for Long-Horizon Tasks

Circle Chen, Colin Cai, Lingqi Zeng
{circlecly,cai9,lzengaf}@berkeley.edu

Extended Abstract

In the dynamic field of Reinforcement Learning (RL) research, mastering long-horizon sparse-reward tasks remains a formidable challenge. Traditional RL agents, lacking prior knowledge, rely solely on sparse environmental rewards to discern effective actions. In contrast, humans leverage past knowledge to efficiently adapt and accomplish new tasks, a capability that current RL methodologies often lack.

Recent advancements in Large Language Models (LLMs) like GPT and BERT have showcased their remarkable ability to encode vast world knowledge and perform contextual reasoning. However, LLMs are not inherently grounded in specific tasks or environments, and directly using them as primary agents can lead to uncertainty and instability. Moreover, the computational demands of these pre-trained models, with billions or trillions of parameters, make them impractical for local deployment. Concurrently, Hierarchical Reinforcement Learning (HRL) methods have shown promise in managing complex tasks by exploiting their hierarchical nature, central to which is an effective high-level planning policy that can reason about composing skills.

Our work leverages the planning capabilities of LLMs to augment a Deep Q-Network (DQN) planner within an HRL framework. We first train a set of basic skill agents using curriculum learning in various BabyAI environments. These skills are then composed using a DQN planner, forming the basis of our hierarchical approach. The DQN planner is further augmented with an LLM-matching reward bonus, based on the similarity between its decisions and those suggested by the LLM during training. This integration offers three key advantages:

1. It eliminates the need for an LLM during evaluation, which is beneficial in environments with limited internet access, reduces LLM API costs and avoids rate-limiting issues.
2. The DQN's learning process is accelerated by providing additional signals that would typically require manual specification.
3. With sufficient experience, the system retains the potential to outperform pure LLM-based planners.

We benchmark our method against various baseline models in multiple complex test environments. These include models trained using Proximal Policy Optimization (PPO), Advantage Actor Critic (A2C), a standalone DQN planner, and a pure LLM planner. Our approach demonstrates improved performance over these baselines in certain scenarios and can even surpass a pure LLM-based planner due to the DQN's additional optimization. Both the LLM and DQN are shown to contribute to the model's performance through ablation studies. However, our method's performance is not consistently optimal, highlighting future research directions such as further training and benchmarking against state-of-the-art models in the BabyAI environment.
[1]

[1]Code can be found at: https://github.com/CS285-Project/minigrid

1 Introduction

Hierarchical reinforcement learning (HRL) is a promising approach to extend traditional reinforcement learning (RL) methods to solve more complex tasks by decomposing complex tasks into manageable subtasks Nachum et al. (2018) Botvinick (2012). However, HRL agents typically face difficulties in environments with sparse rewards and a large number of potential actions or skills, often requiring manual intervention for optimal performance.

Recent advancements in Large Language Models (LLMs) have demonstrated their impressive capabilities in reasoning and generating commonsense responses. These attributes position LLMs as promising tools for enhancing the understanding of environmental observations in RL settings. This concept has been explored in various contexts Yuan et al. (2023a) Brohan et al. (2023), with one notable approach being the use of LLMs as planners for RL agents. However, pre-trained language models come with inherent biases and lack grounding in specific tasks or environments, which can lead to uncertainty and instability in their guidance. Additionally, the computational demands of LLMs limit their applicability in less advanced setups.

In light of these considerations, we hypothesize that the insights extracted from LLMs can provide valuable supplementary signals to a traditional Deep Q-Network (DQN) planner within an HRL framework. By augmenting the DQN planner with LLM-derived signals, we anticipate accelerated learning and enhanced performance, leveraging both the strategic insights from LLMs and the environment-specific adaptations of the DQN planner.

2 Related Work

BabyAI environment using RL and Language We looked into studies that use the same environment library - BabyAI - as us. For instance, Exploration through Learned Language Abstraction (ELLA) Mirchandani et al. (2021) proposed a reward-shaping approach using relevance and termination classifiers and learned those classifiers on the fly. Carta et al. (2022) improved upon the ELLA approach, by querying a Question Answering model with the current observations to generate an intrinsic award. These studies, however, do not use the language to help with planning and require samples on the scale of 10^7 or 10^8 environment interactions, which is relatively sample inefficient.

LLM as a Provider of Intrinsic Reward Planning is not the only way to solve a long-horizon, sparse-reward task. If one can reshape the reward function from sparse to dense by introducing some form of intrinsic reward, then the agent might be able to solve a long-horizon task. This rationale is the idea behind various studies Parisi et al. (2021) Andres et al. (2022), and has inspired researchers to use an LLM to provide extra rewards, such as the one in Kwon et al. (2023).

LLM as Planners Numerous studies investigate using LLM as a "planner" to assist agents in performing long-horizon and sparse-reward tasks. These studies typically formulate an HRL agent. One extensively studied environment in this field is the MineDojo environment Fan et al. (2022) which simulates a game of Minecraft. For instance, Voyager Wang et al. (2023) powered by GPT-4 models encourages the agent to make novel discoveries via a three-component approach: an automatic curriculum, a skill library, and an iterative prompting mechanism using code as action space. Goal-Sensitive Backbone (GSB) Cai et al. (2023) encouraged the agent to encode game information using goal-relevant visual state representations, and Plan4MC Yuan et al. (2023b) manually prompted the LLM to construct a skill graph to help agents achieve a long-term goal (crafting iron armor) through short-term actions (chopping wood, finding stone, etc.).

Other environments are also used to study the potential of LLMs as language planners. For instance, SayCan Ahn et al. (2022) proposes combining the LLM response with the agent's knowledge of the world state via an "affordance score", and enables the LLM and embodied agent to complement each other's knowledge. Their method was tested in the real world on robots. LLM-Planner Song et al. (2023) proposes a method that allows LLMs to effectively plan for the agents in a few shots, and tested their work on the ALFRED dataset. SPRING Wu et al. (2023) employs a directed acyclic graph (DAG) with game-related questions in the Crafter environment, and traversed each node in topological order, asking the LLM at each step. HELM Paischer et al. (2022) uses a pre-trained

Transformer model to store history observations as a memory module, demonstrating success in the RandomMaze, Minigrid, and Procgen environments.

3 PROBLEM STATEMENT

We consider a system wherein an agent receives natural language descriptions of tasks, constituting the goals within a predefined environment. These descriptions conform to a fixed set of phrase formats specific to the environment. The system comprises a high-level planner agent with access to a set of primitive skill agents, denoted as $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$. Each skill agent S_i is capable of processing environmental observations and a fixed-format natural language goal to produce an action within the environment. The primary objective is to develop a planner agent that effectively selects the optimal skill policy S_i and corresponding input to fulfill the task.

3.1 EXPERIMENTAL ENVIRONMENT

Our experiments are conducted within the BabyAI platform Chevalier-Boisvert et al. (2019), which aligns with the system requirements. This environment is a grid-based world featuring diverse rooms and objects. The task given to the agent also varies between episodes, and is described in natural language with fixed grammer, such as "pick up the (red/blue/yellow) key" or "go to the red box". The complexity of these environments varies, ranging from simple single-room layouts to intricate multi-room structures with locked doors and non-essential objects. Figure 1 illustrates the environment, which is configured in both fully and partially observed settings for our experiments.

3.2 ACTION SPACE

Planner Agent The planner agent operates in a discrete action space $\mathcal{A}_p$, which involves selecting a skill number and corresponding goal text. In our study, $\mathcal{A}_p$ comprises 48 distinct choices, as detailed in Section 5.

Skill Agent The skill agent's action space $\mathcal{A}_s$ includes 7 discrete actions: turn left, turn right, move forward, pick up an object, drop an object, toggle the object in front, or declare task completion.

3.3 REWARD STRUCTURE

A reward of $1 - 0.9 \times \frac{s}{m}$ is granted upon task completion, where s represents the number of steps taken and m is the maximum step limit, which varies from environment to environment. During intermediate steps, or when the agent fails to achieve the goal, the reward given is 0. This formulation of sparse reward makes the environment extremely hard for traditional RL methods, which rely on active and constant feedback from the environment to differentiate between good and bad actions to apply successfully.

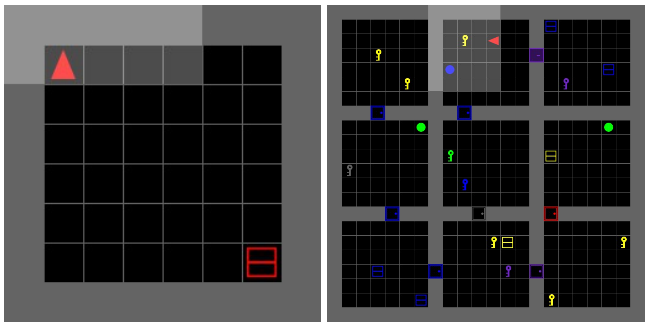

Figure 1: **Left**: GoToObj environment, with mission text "go to the red box". The agent only needs to reach the red box to solve the environment. **Right**: GoToImpUnlock environment, with mission text "go to the green ball". The agent may need to open locked doors to reach the target object.

4 METHODOLOGY

Our proposed model adopts a hierarchical structure in the realm of reinforcement learning. The architecture is designed with two levels of operation:

- **High-Level Planner:** At the higher echelon, we employ a Large Language Model (LLM) and/or a Deep Q-Network (DQN) as the strategic planner. This component is responsible for overarching decision-making and guiding the overall direction of the learning process. It works by breaking the long-term goal into short-term subgoals hence making the task easier to solve.
- **Low-Level Agents:** The lower tier consists of skill networks trained with Proximal Policy Optimization (PPO). These networks function as tactical agents, executing specific skills and actions as directed by the high-level planner. Skills are manually selected and the corresponding agents are pre-trained for the tasks and are frozen when used in the HRL architecture.

The interaction between these two levels is governed by a defined query frequency. At regular intervals, the high-level planner (LLM/DQN) is queried for its decision. Upon each query, the LLM/DQN selects an appropriate skill to activate and simultaneously proposes a goal for that skill.

4.1 LLM AUGMENTED DQN PLANNER

Our methodology introduces an innovative approach by augmenting a traditional Deep Q-Network (DQN) planner with a Large Language Model (LLM). Our method is described in Algorithm 1. This integration aims to enhance the decision-making process in a hierarchical reinforcement learning framework. The key components of our approach are as follows:

1. **DQN Planner:** The DQN planner operates as a conventional reinforcement learning agent. It outputs discrete actions, each corresponding to a specific skill and goal combination. The DQN is trained to select the right skill that results in obtaining rewards from the environment.
2. **LLM Integration:** The LLM is utilized to provide strategic guidance to the DQN planner. Unlike the DQN, which is queried at fixed environment step intervals, the LLM is queried during the training phase of the DQN planner. The result of LLM is validated such that the words are from a pre-defined dictionary, making it easier to parse and harder to deviate from the desired results.
3. **Query Mechanism:** The planner queries the DQN at fixed environment step intervals for immediate decision-making. In contrast, the LLM is queried after a predetermined number of DQN queries.
4. **Reward Shaping:** A novel aspect of our approach is the reshaping of the planner reward. A reward bonus, in addition to the environment reward mentioned in item 1, is given when the DQN's output matches the LLM's recommendation. The reward calculation is as follows: a match in chosen skills yields a reward of 1; similarly, matches in object color and type each contribute a reward of 1. Non-matches receive no reward. The final reward is the average of these individual rewards, promoting alignment with LLM guidance.
5. **Training and Updating:** The DQN is continuously trained and updated based on the reshaped reward signal. This training process incorporates feedback from both the environment and the LLM, ensuring that the DQN's policy evolves to effectively integrate the strategic guidance provided by the LLM.

This methodology aims to complement the strengths of both the DQN and LLM, aiming to cultivate a decision-making capability in the DQN that eventually surpasses the LLM in complex environments.

4.2 LOW-LEVEL AGENTS TRAINED WITH CURRICULUM LEARNING

In our approach, we train low-level skill agents in a range of environments for generalizability in complex settings. Drawing from curriculum learning, we progressively expose agents to environments of increasing difficulty based on their performance. This involves a sequence of scenes around

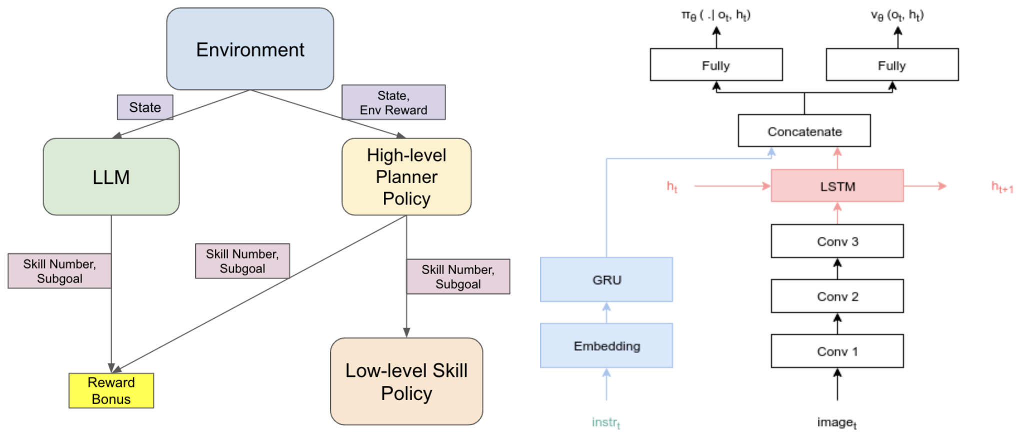

Figure 2: **Left**: LLM augmented DQN planner. **Right**: The "basic" model architecture Chevalier-Boisvert et al. (2019). A Gated Recurrent Unit (GRU) processes the mission text, and a ConvNet in addition to an LSTM is used to process the current and historical observations. The features are concatenated and sent to fully connected layers to output action distributions and values.

Algorithm 1 Integration of LLM with DQN in HRL Framework

```
Require: Initialize DQN planner, LLM, and the environment
Require: Set DQN query interval, LLM query frequency, and reward shaping mechanism
 1: while environment is not terminated do
 2:     Observe current state $s$
 3:     Replay Buffer $B$
 4:     $a \leftarrow$ invoke $dqn\_decision(s)$
 5:     Execute $a$ in environment
 6:     Observe reward $r$ and new state $s'$
 7:     Add $(s, a, s', r)$ to $B$
 8:     if DQN query interval is reached then
 9:         $dqn\_decision \leftarrow$ DQN planner$(s)$
10:     end if
11:     Sample mini-batch from $B$
12:     if LLM query frequency is reached then
13:         for $(s_i, a_i, s'_i, r_i)$ in $B$ do
14:             $llm\_recommendation \leftarrow$ LLM$(s_i)$
15:             $dqn\_recommendation \leftarrow$ DQN planner$(s_i)$
16:             if $dqn\_recommendation$ matches $llm\_recommendation$ then
17:                 Reshape reward: $r_i \leftarrow r_i +$ bonus
18:             end if
19:         end for
20:     end if
21:     Update DQN with mini-batch
22:     $s \leftarrow s'$
23: end while
```

a single set of related tasks, each with varying levels of challenges, such as increasing the number of distractors.

As stated in algorithm 2, agents progress or regress based on performance thresholds. If an agent struggles (failing to progress after three attempts), we shift to a different scene. The training process has two stopping criteria: 1) the agent successfully navigates all levels across all scenes, or 2) the training reaches a predetermined total frame count.

Algorithm 2 Curriculum Learning for Skill Agents

```
Require: Set of scenes $S$, thresholds $T_{up}$, $T_{down}$, max attempts $M$, total frames $F_{max}$
 1: Initialize $F \leftarrow 0$, current scene $s \leftarrow$ first in $S$, level $l \leftarrow 0$
 2: while $F < F_{max}$ do
 3:     Train and evaluate agent in scene $s$ at level $l$, update $F$
 4:     if $P \geq T_{up}$ then
 5:         $l \leftarrow l + 1$ {Advance level}
 6:     else if $P < T_{down}$ and $l > 1$ then
 7:         $l \leftarrow l - 1$ {Reduce level}
 8:     end if
 9:     if Max attempts at $l$ reached then
10:         Next scene in $S$, reset attempts, $l \leftarrow 1$
11:     end if
12:     if All levels in all scenes completed then
13:         break
14:     end if
15: end while
```

5 Experiments and Analysis

5.1 Hyperparameter Tuning for Basic Environments

The efficacy of our complex planner hinges on the performance of underlying basic skill policies, necessitating optimal hyperparameter tuning in simpler environments. Our experiments focused on variables like observation size and memory size. Consequently, we adopted an observation size of 11×11 and a memory size that allows back-propagation over 20 frames, ensuring the skill policies are adequately trained to support the complex planner's functionality.

5.2 Baselines

5.2.1 PPO and A2C

Preliminary investigations suggested that full observation is crucial for the LLM's effective assistance in skill agent tasks. This led us to question the impact of providing full observation directly to the basic skill model.

To explore this, we trained two baseline models using the "basic" network structure (Figure 2) with Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C) in the `BabyAI-GoToImpUnlock-v0` environment under full observation. The training configurations for these algorithms were as follows: a batch size of 1280 and 64 processes. For the number of frames per process before an update, we used a default of 5 for A2C and 128 for PPO. As shown in Figure 5, these models underperformed despite full observation, likely due to sparse rewards and large input dimensions affecting the policy's sequential learning for long-term goals.

Considering these results and our insights on reward shaping (see Appendix for details), we infer that combining LLM-shaped rewards with the basic network might not yield optimal performance even with full observability, so we decided to only keep the above 2 baselines without reward shaping.

5.2.2 Pure LLM Planner with Skill Agent

We also assessed the performance of a pure Large Language Model (LLM) planner. In this setup, the skills, trained via curriculum learning, were frozen, and an LLM was employed as the planner. We conducted many experiments with different LLMs including GPT-3.5, GPT-4, Llama2, and Vicuna. Considering cost and performance, we eventually used GPT-3.5 for all of the experiments. Then we conducted experiments across different query frequencies —every 80, 160, and 800 frames—in the `GoToImpUnlock` environment, and found that query at once per 160 frames yielded the best performance. Similar tests were extended to other environments. The results are depicted in Figure 4.

5.3 Using Curriculum Learning to Refine Basic Networks

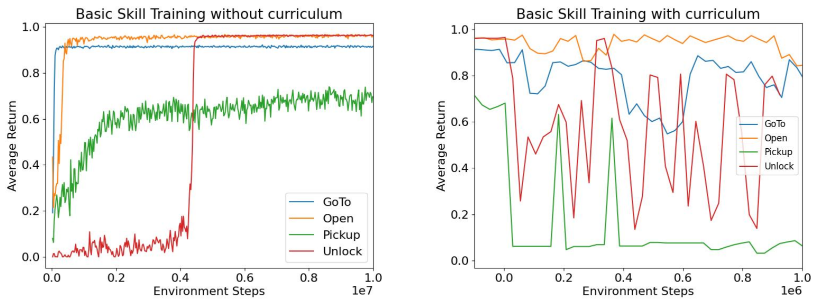

Figure 3: **Left**: Training curves for basic skills without curriculum learning. **Right**: Fine-tuning of basic skills using curriculum learning, with environment steps indicated post-pretraining. Fluctuations in return are due to periodic environment switches.

During the preliminary investigation phase, we found that training basic skills in a single basic environment causes poor generalizability to the complex test environment, due to distribution shift. To address this, we exposed the basic networks to a diverse range of training environments.

The sequence of environment exposure was not arbitrary. Drawing inspiration from curriculum learning concepts Soviany et al. (2022), we assessed and ranked the difficulty of each environment. This ranking informed the training sequence, as outlined in Algorithm 2, with parameters set as $T_{up} = 0.6$, $T_{down} = 0.3$, $M = 3$, and $F_{max} = 10^7$. The training curves under this curriculum learning framework are depicted on the right in Figure 3.

5.4 DQN Planner Augmented with LLM

A notable limitation of LLM Planners is their fixed parameters, restricting their adaptability to new environmental information. Additionally, it has a reliance on API requests, which are often subject to rate limits. To address these issues, we explored the use of a DQN planner augmented with LLM responses. The DQN planner, tasked with generating skills and goals, learns from both the LLM feedback and environmental interactions during training. Notably, in the evaluation phase, the planner operates independently, relying solely on its learned strategies without requiring LLM input.

This architecture offers two primary advantages. First, it eliminates the need for LLM access during evaluation, contrasting with several state-of-the-art methods (e.g., Hu et al. (2023)) that depend on continuous LLM availability. Second, by learning from environmental rewards, the DQN planner potentially exhibits greater flexibility and adaptability than a pure LLM planner, especially with increased experience. The training configuration included a batch size of 1280, 64 processes, and a frames-per-process setting of 40, with the LLM queried every 160 frames.

The performance of the LLM-augmented DQN planner is presented in Figure 4. Our findings indicate that this augmented planner outperforms the PPO and A2C baselines in certain scenarios, although it does not consistently surpass the pure LLM planner. This suggests a balanced integration of LLM guidance and DQN's environmental learning can enhance overall task performance.

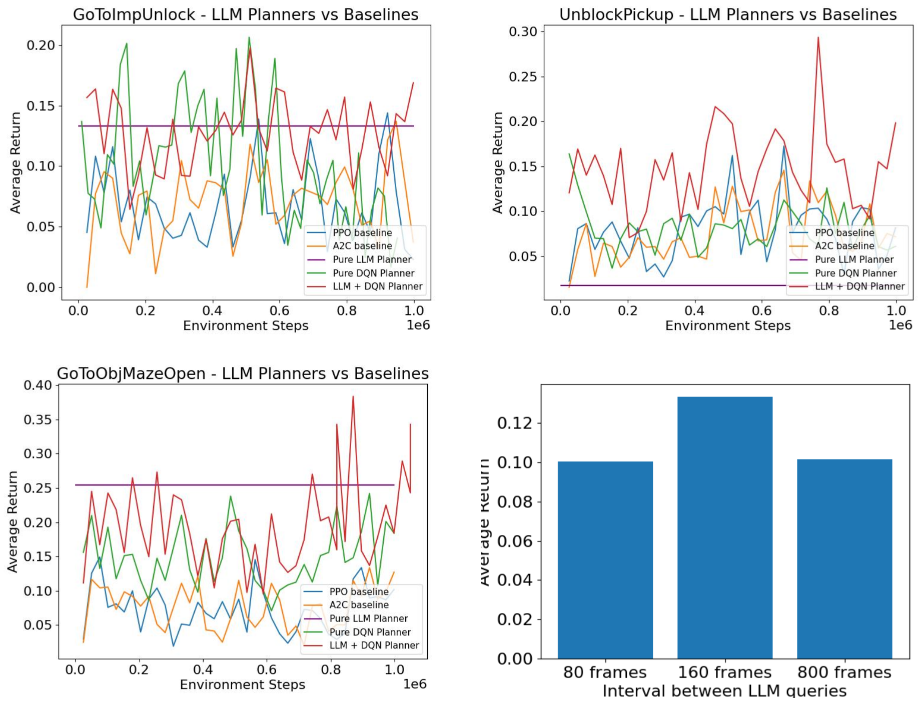

Figure 4: Test environment results with different planners on different environments. **Top Left, Top Right, Bottom Left:** performance curves in different environments. **Bottom Right:** Average return with pure LLM model using different query frequencies. See Appendix for more performance plots.

5.5 Ablation Study

In an ablation study on the LLM-augmented DQN planner mentioned above, we remove the LLM augmentation and train the DQN with only environment reward. As shown in Figure 4, the pure DQN planner generally performs worse than the LLM+DQN planner. However, in a very complicated environment like in the `GoToImpUnlock` environment, the gain can be marginal. It's likely dye to the fact that the performance of pure LLM planners is about the same as pure DQN, and thus not providing valuable information.

5.6 Performance Upper Bound - Human Planner

To test the upper bound of our model, we also substituted the high-level planner for a human. When it is time to plan, the program gives the user the image and waits for the user's input on the skill number and goal. We notice that our model is generally able to follow instructions, achieving the planned goals 80% to 90% of the time. It indicates that our basic skills are functional; however, it's too labor-intensive to be scaled to all test cases.

6 Conclusion and Discussion

This paper proposes an LLM-augmented DQN planner HRL method to solve sparse-reward long-horizon tasks. This proposed method, unlike most other state-of-the-art LLM integrations to RL, doesn't require having access to the full LLM model during evaluation. Thus, it can be applied when the deployed environment doesn't have internet access or the capacity to run full LLM models locally, making it more versatile. It has shown a noticeable performance increase compared to baseline models in some cases, and can even outperform a pure LLM-based planner thanks to the additional optimization done by the DQN algorithm. Despite having huge potential, we also recognize that it isn't giving the best performances consistently.

We think there is a lot more fine-tuning that can be done with the prompt mechanism and the reward bonus calculation formula. Due to time and resource constraints, we had to limit the training steps, training batch size, and reward bonus frequency when running experiments with the LLM-augmented DQN planner. We believe having the ability to tune those parameters freely would have unlocked much better performance. For future works, we also want to benchmark more modern models like Carta et al. (2022) and Mirchandani et al. (2021) to see how our method compares.

TEAM MEMBER CONTRIBUTIONS

Circle Chen mainly contributed to the LLM prompting mechanism and the LLM planner. Colin Cai mainly wrote the DQN planner and the integration with LLM. Lingqi Zeng is mainly responsible for the curriculum training skill agents and validation of LLM responses. All contributed equally to the project.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022.

Alain Andres, Esther Villar-Rodriguez, and Javier Del Ser. Towards improving exploration in self-imitation learning using intrinsic motivation, 2022.

Matthew Michael Botvinick. Hierarchical reinforcement learning and decision making. *Current opinion in neurobiology*, 22(6):956–962, 2012.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pp. 287–318. PMLR, 2023.

Shaofei Cai, Zihao Wang, Xiaojian Ma, Anji Liu, and Yitao Liang. Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction, 2023.

Thomas Carta, Pierre-Yves Oudeyer, Olivier Sigaud, and Sylvain Lamprier. Eager: Asking and answering questions for automatic reward shaping in language-guided rl, 2022.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning, 2019.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge, 2022.

Bin Hu, Chenyang Zhao, Pu Zhang, Zihao Zhou, Yuanhang Yang, Zenglin Xu, and Bin Liu. Enabling intelligent interactions between an agent and an llm: A reinforcement learning approach, 2023.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models, 2023.

Suvir Mirchandani, Siddharth Karamcheti, and Dorsa Sadigh. Ella: Exploration through learned language abstraction, 2021.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Fabian Paischer, Thomas Adler, Vihang Patil, Angela Bitto-Nemling, Markus Holzleitner, Sebastian Lehner, Hamid Eghbal-Zadeh, and Sepp Hochreiter. History compression via language models in reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17156–17185. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/paischer22a.html`.

Simone Parisi, Victoria Dean, Deepak Pathak, and Abhinav Gupta. Interesting object, curious agent: Learning task-agnostic exploration, 2021.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models, 2023.

Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum learning: A survey, 2022.

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.

Yue Wu, Shrimai Prabhumoye, So Yeon Min, Yonatan Bisk, Ruslan Salakhutdinov, Amos Azaria, Tom Mitchell, and Yuanzhi Li. Spring: Studying the paper and reasoning to play games, 2023.

Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, and Zongqing Lu. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*, 2023a.

Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, and Zongqing Lu. Skill reinforcement learning and planning for open-world long-horizon tasks, 2023b.

A Appendix

A.1 Design of the Subgoal

The `color` can be one of `red, green, blue, purple, yellow, or grey`.
`type` is chosen among `ball, box, key`.

For each skill, the goal is structured as follows:

Skill 0 (GoToObj): The agent is prompted to go to an object with a specified color and type.

```
go to the {color} {type}
```

Skill 1 (Open): The agent's goal is to open a door of a specified color.

```
open the {color} door
```

Skill 2 (Pickup): The task involves picking up an object with a specific color and type.

```
pick up the {color} {type}
```

Skill 3 (Unlock): The agent is required to unlock a door of a certain color.

```
unlock the {color} door
```

A.2 Prompt Design

The `color` can be one of `red, green, blue, purple, yellow or grey`, and `type` is chosen among `ball, box, key`.

For the skill `GoToObj`, the goal is structured as follows:

```
go to the {color} {type}
```

A.3 LLM Prompt Structure

We prompt the LLM with the following structure:

```
json_example = "{ skill : 0, goal : 'go to the red box' }"
You are an agent in a Minigrid environment. Your agent is in Room
{(room_r, room_c)}. Rooms are 7 by 7, and there are 9 of them, connected by
doors. Your mission is {mission}. Here is the state of the entire environment:

{image_as_str}

You have the following skills and their allowed goal grammar below:
Skill 0: Go to Object
    "go to the [color] [type]"
    [color]: the color of the object. Allowed values are red, green, blue,
    purple, yellow or grey.
    [type]: the type of the object. Allowed values are ball, box, key.
Skill 1: Open door
    "open the [color] door"
    [color]: the color of door. Allowed values are red, green, blue, purple,
    yellow or grey.
Skill 2: Pickup an item
    "pick up the [color] [type]"
    [color]: the color of the object. Allowed values are red, green, blue,
    purple, yellow or grey.
    [type]: the type of the object. Allowed values are ball, box, or key.
Skill 3: Unlock a door
```

```
    "unlock the [color] door"
    [color]: the color of the object. Allowed values are red, green, blue,
    purple, yellow or grey.

Based on the current state of the agent, what is the first skill it should use,
and what would be the short-term goal for that skill (most likely different
from the long-term)? Format your answer in json format. As an example, you can
return {json_example}.
```

B Preliminary Investigations

B.1 Investigations with one Basic Network

To familiarize ourselves with the BabyAI environments and understand why an LLM planner might be better for our tasks, we conducted various preliminary experiments in the environment. We then explain the main takeaways we gained from these experiments and apply them in further experiments. These investigations use only one "basic" model architecture, as illustrated in Figure 2.

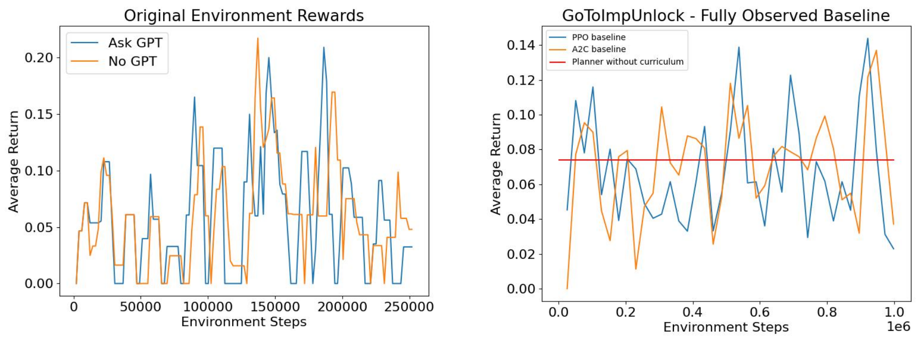

Figure 5: **Left**: Using GPT-3.5 as the provider of reshape rewards for a basic network with partial observation. **Right**: Using PPO and A2C to train an agent with full observation of the environment, as well as using a GPT-3.5 planner with basic skills learned without curriculum learning.

Using LLM as a means of reward shaping Inspired by state-of-the-art reward shaping methods in the BabyAI environment, such as Mirchandani et al. (2021) and Carta et al. (2022), as well as LLM-based reward shaping methods Kwon et al. (2023), we attempt to train a basic model in conjunction with an LLM prompted to reshape the reward. We prompted the LLM to either directly return the intrinsic reward based on the goal, observation, and intended action, or return a trajectory for the next 10 time steps, whose similarity with the actual actions of the policy is taken as an intrinsic reward.

This attempt is not successful, as shown in the left image in Figure 5, and the reasons we believe lie in (1) the LLM's failure to understand the significance of the numerical values of the rewards, thus giving numbers that don't make sense, and (2) the LLM's failure to connect the current state with the global state, given that only the partial observation is contained in the prompt. The main takeaway from this section is that (1) we should avoid asking LLMs to provide environment-specific output (such as a numerical reward) and (2) we should provide the LLM with the whole output.

B.2 Fully Observed policy with the Basic Model Architecture

Preliminary investigations indicated that full observation is essential for the LLM to effectively assist the skill agent. While we could just start experimenting with a hierarchical RL approach, one natural question is: if we directly provide the basic skill model with the full observation, what would it do?

To answer this, we trained two baseline models with the "basic" network structure in Figure 2 using Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C) directly on the test environment `BabyAI-GoToImpUnlock-v0`. The results, illustrated in the right image of Figure 5, indicate that the basic structure struggles to achieve high performance, even with full observation. This limitation is likely attributed to the combination of sparse rewards and increased input dimensions, which hinders the policy's ability to learn and execute a sequence of steps towards a long-term goal.

Combining this and our findings about reward shaping, we extrapolate that LLM-shaped reward combined with full observability would not perform well, so we shift our focus to using hierarchical RL approaches.

B.3 Integrating Multiple Basic Networks

This section marks our shift towards a hierarchical RL approach, where we integrate multiple networks with the "basic" architecture using a planner.

Initially, we selected four fundamental skills: `GoToObj`, `OpenDoor`, `PickUpObj`, and `UnlockDoor`. Corresponding basic networks were initialized and trained in their respective environments. For instance, `GoToObj` was trained in `BabyAI-GoToObj-v0` (see left in Figure 1). Each skill underwent approximately 10^7 training steps using PPO, achieving a minimum reward of about 0.7. The training progress is depicted in Figure 3. Post-training, these networks were frozen, preventing further updates during integration with the high-level planner.

Subsequently, the LLM was prompted with full observation details, informed about available skills, and asked to select the next skill network and its short-term goal for a fixed number of frames. The chosen skill network, receiving only partial observation, generated the action. Despite potential rewards, neither the skill networks (already frozen) nor the planner (operating as a blackbox LLM) underwent updates.

However, this method did not yield significant improvements over our baselines. As shown in the right image of Figure 5, the planner with basic skills from these environments did not surpass the baseline performance, indicated by the red horizontal line.

In isolating the issue, we replaced the LLM planner with a human planner. The agent predominantly exhibited random movement, confounded by distractors, leading to an average reward comparable to that in Figure 5. This suggests that the limitations likely lie within the basic skills themselves, prompting us to further refine our approach by training the basic skills across multiple environments.

B.4 Additional Performance Plots

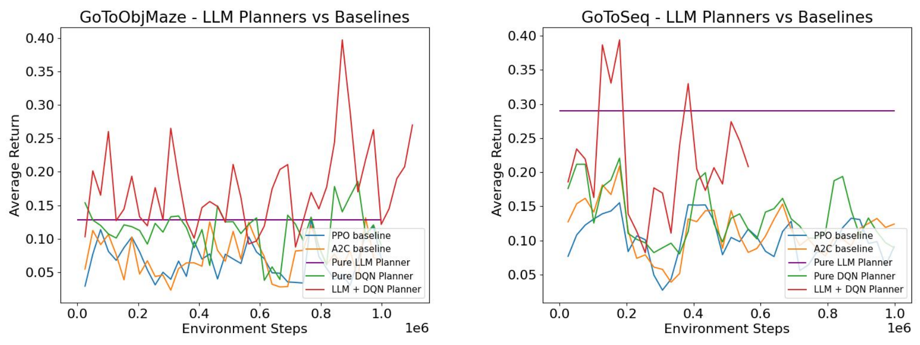

Figure 6: Test environment results with different planners on different environments. Performance curves in more different environments.