

Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)

<https://creativecommons.org/licenses/by-nc/4.0/>

Access to this work was provided by the University of Maryland, Baltimore County (UMBC) ScholarWorks@UMBC digital repository on the Maryland Shared Open Access (MD-SOAR) platform.

Please provide feedback

Please support the ScholarWorks@UMBC repository by emailing scholarworks-group@umbc.edu and telling us what having access to this work means to you and why it's important to you. Thank you.

Using LLMs for Augmenting Hierarchical Agents with Common Sense Priors

Bharat Prakash^{1, 2}, Tim Oates¹, Tinoosh Mohsenin^{1, 2}

¹University of Maryland, Baltimore County, ² Johns Hopkins University
bhp1@umbc.edu, oates@.umbc.edu, tinoosh@umbc.edu *

Abstract

Solving long-horizon, temporally-extended tasks using Reinforcement Learning (RL) is challenging, compounded by the common practice of learning without prior knowledge (or *tabula rasa* learning). Humans can generate and execute plans with temporally-extended actions and quickly learn to perform new tasks because we almost never solve problems from scratch. We want autonomous agents to have this same ability. Recently, LLMs have been shown to encode a tremendous amount of knowledge about the world and to perform impressive in-context learning and reasoning. However, using LLMs to solve real world problems is hard because they are not grounded in the current task. In this paper we exploit the planning capabilities of LLMs while using RL to provide learning from the environment, resulting in a hierarchical agent that uses LLMs to solve long-horizon tasks. Instead of completely relying on LLMs, they guide a high-level policy, making learning significantly more sample efficient. This approach is evaluated in simulation environments such as MiniGrid, SkillHack, and Crafter, and on a real robot arm in block manipulation tasks. We show that agents trained using our approach outperform other baselines methods and, once trained, don't need access to LLMs during deployment.

Introduction

Humans can generate and execute plans with temporally extended actions to perform complex tasks in a dynamic and uncertain world. We would like autonomous agents to have the same capabilities. Massive engineering efforts can lead to agents that are remarkably robust, such as rovers in space, and surgical and industrial robots. In the absence of such resources, techniques such as Reinforcement Learning (RL) can be used to extract robust control policies from experience. However, RL has many challenges, such as exploration under sparse rewards, generalization, safety, etc. This makes it difficult to learn good policies in a sample efficient way. Popular ways to tackle these problems include using expert feedback (Christiano et al. 2017; Warnell et al. 2018) and leveraging the hierarchical structure of complex tasks.

There is significant prior work on learning hierarchical policies to break down tasks into smaller sub-tasks (1999; 2017; 2017).

Hierarchical Reinforcement Learning (HRL) does indeed mitigate some of the problems mentioned above. However, as the number of options or skills increases, we face some of the same problems again. Using some form of supervision, such as providing details about the sub-tasks or intermediate rewards or high-level human guidance, is one approach (Prakash et al. 2021; Jiang et al. 2019; Le et al. 2018).

One of the reasons that humans are so good at dealing with unfamiliar situations is that we almost never solve problems from scratch. Presented with a new task and a library of skills, we are able to choose a subset of skills that seem most relevant and explore from there. We might perform some trial and error exploration (as in RL), but we quickly learn the right subset of skills as well as the correct sequence in which they need to be executed. For example, the door handles on newer cars lie flat against the door, unlike most other car door handles in existence. That presents a problem the first time you try to open one. Humans immediately narrow down to a few exploratory actions, like trying to get a finger under the handle or pushing on it in different places. We don't, unlike most RL algorithms might, tap the window or pull the side mirrors, as we believe that such options are causally irrelevant based on deep world knowledge.

Large language models (LLMs) have been shown to encode a tremendous amount of knowledge about the world by virtue of being trained on massive amounts of text. We hypothesize that this knowledge can be leveraged to focus the training of hierarchical policies, making them significantly more sample efficient. In particular, we explore how large pretrained language models can be used to inject common sense priors into hierarchical agents.

In this approach we assume access to several low level skills. These can be, for example, engineered planners or policies learned using RL and sub-task rewards. Based on a high-level task description and current state, the LLMs guide the agent by suggesting the most likely courses of action. Instead of random exploration, we use these suggestions to intelligently explore the various options. Because LLMs are not grounded in the domain, they are only used to bias action selection and their influence is reduced as training progresses. This results in a policy that can be de-

*This project was sponsored by the U.S. Army Research Laboratory under Cooperative Agreement Number W911NF2120076. Copyright © 2024 by the authors. This open access article is published under the Creative Commons Attribution-NonCommercial 4.0 International License.

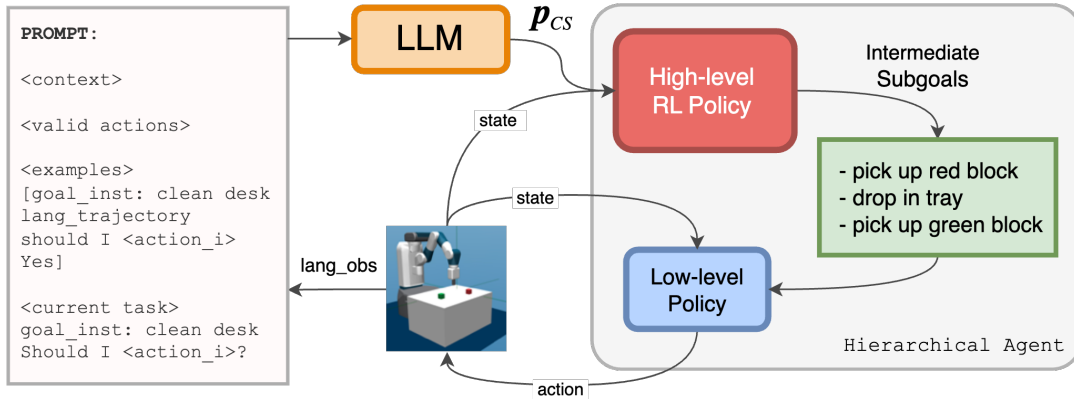


Figure 1: The LLM guides the high-level policy and accelerates learning. It is prompted with the context, some examples, and the current task and observation. The LLM’s output biases high-level action selection

ployed without depending on the LLM at run time. We evaluate this approach on several simulated environments (MiniGrid (Chevalier-Boisvert, Willems, and Pal 2018), SkillHack (Matthews et al. 2022), and Crafter (Hafner 2021)), showing that it can learn to solve complex, long-horizon tasks much faster than baseline methods. Experiments with a real robot arm in block manipulation tasks using a tabular Q-learning version of the same algorithm show that it can learn policies much faster with less experience in that domain. Our contributions are summarized as follows:

- an approach for using LLMs to guide exploration by extracting common sense priors;
- a hierarchical agent that uses these priors to solve long-horizon tasks;
- an evaluation of the framework in simulation as well as a simple real-world environment, showing that it performs significantly better than baselines;
- a discussion of (1) the advantages of our method compared prior work and (2) potential future work.

Related Work

Language and HRL There is significant prior work on hierarchical RL where the standard MDP is converted into a semi-Markov decision process (SMDP). The most common approach is to incorporate temporally extended actions, also known as options or skills (Bacon, Harb, and Precup 2017). Typically, a low-level policy achieves sub-tasks by executing primitive actions and a high-level policy plans over temporally extended options or skills. Natural language is a popular way to specify sub-tasks and achieve generalization due to its inherent compositionality and hierarchical structure (Jiang et al. 2019; Prakash et al. 2021; Waytowich et al. 2019; Hu et al. 2019). Most of these methods specify or generate a high-level plan in natural language, which is then executed sequentially by a separate low-level policy. These approaches face challenges when operating in high-dimensional observation spaces. They also rely on manual data collection to train the high-level policies and are therefore difficult to generalize to new tasks (Andreas, Klein, and

Levine 2017; Mirchandani, Karamcheti, and Sadigh 2021; Goyal, Niekum, and Mooney 2019).

RL and Foundation Models Recently, large language models such as GPT-3 have been used to build agents capable of acting in the real world based on language instructions (Brown et al. 2020; Bommasani et al. 2021). The in-context learning and intelligent prompting strategies supported by these models have been used to design language-guided hierarchical agents. (Huang et al. 2022) use LLMs as zero-shot planners to enable embodied agents to act in real world scenarios. Similarly, (Ahn et al. 2022) use LLMs along with affordance functions to generate feasible plans that guide a robot to achieve goals specified in natural language instructions. Our work is closely related to (Du et al. 2023), where they improve exploration by using LLMs to provide intermediate rewards and encourage the agent to seek novel states.

Methods

Problem Statement

We consider a system that receives instructions in the form of natural language describing a task, similar to (Ahn et al. 2022). The instructions can be long, may contain warnings and constraints, and may not include all of the necessary individual steps. We also assume that the agent has access to a finite set of skills or sub-policies that can be executed in sequence to solve long-horizon tasks. These skills can be hand-coded, or trained using reinforcement learning or imitation learning with manual reward design. They must be accompanied by a simple description in natural language, such as “pick up red block” or “open blue door”. They must also be able to detect sub-task completion to switch control back to the high-level policy.

Our problem setting similar to the options framework (Sutton, Precup, and Singh 1999). Options or skills are policies π_L which are accompanied by a termination condition β and an initiation set I . I refer the reader to (Sutton, Precup, and Singh 1999) for more details. We ignore I and we assume access to a termination condition or horizon τ . Using this we can formulate a fully observable semi-

Markov decision process (MDP) SM described by a tuple (S, G, R, P, γ) . Here S is the state space, G is the finite set of skills or options, R is the sparse reward function $R(s_0|s, g_i)$, P is the transition function $P(s'|s, g_i)$ and γ is the discount factor. Our objective is to learn a high-level policy that selects among this set of options or skills.

Using LLMs to Guide High-level Policies

This section introduces our method for using LLMs to improve exploration in the high-level policy of an HRL system. The semantic knowledge and planning capabilities of LLMs improve high-level action selection given a task description and current state in the form of language. The core idea is to use LLMs to obtain a value that approximates the probability that a given skill or sub-task is relevant to achieve the larger goal. As mentioned earlier, each skill is accompanied by a language description l_{skill} and the current trajectory is translated into language, l_{traj} . There is also a high level instruction, l_{goal_inst} , describing the larger goal along with optional constraints.

Algorithm 1

```

high_inst  $\leftarrow$  high level goal in language
 $\pi_\theta \leftarrow$  high level policy
 $f_{LLM} \leftarrow$  common sense priors from LLM
procedure LLMxHRL (high_inst)
  init  $\pi_\theta$ 
  while  $\pi_\theta$  not converged do
    init  $\tau \leftarrow \{\}$ 
    for  $t \leftarrow 0$  to  $T$  do
       $p_{CS} \leftarrow f_{LLM}(\text{high\_inst}, \tau)$ 
       $a_t \leftarrow \text{cat\_dist}[\pi_\theta(\tau) + \lambda.p_{CS}(\tau)].\text{sample}()$ 
       $s_t, r_t \leftarrow \text{act}(a_t)$ 
       $\tau \leftarrow \text{append}(s_t, r_t, a_t)$ 
    end for
    update  $\pi_\theta$ 
  end while
  return  $\pi_\theta$ 
end procedure

```

The LLM is used to evaluate the function for each skill at every high-level decision step - $f_{LLM}(l_{skill_i}, l_{goal_inst}, l_{traj})$. Essentially, the LLM answers the following question: given the task, l_{goal_inst} , and trajectory so far, l_{traj} , should we choose skill l_{skill_i} ? The output of the LLM, ‘yes’ or ‘no’, can easily be converted to an int (“0” or “1”). This kind of closed form question-answering prompt has been shown to work better than open ended prompts (Du et al. 2023). After evaluating this for each of the k skills, we get $F_{LLM} = [f_{LLM_1}, f_{LLM_2}, f_{LLM_3}, \dots, f_{LLM_k}]$. For example, $F_{LLM} = [0, 1, 0, \dots, 0, 1, 0, 0]$. A LOG-SOFTMAX function is applied to these logits to get the common sense priors from the LLM denoted by $p_{CS} = \text{log_softmax}(F_{LLM})$. Relying entirely on p_{CS} is not enough to solve complex tasks. At the same time, using RL and exploring without any common

sense intuition is inefficient. Therefore we still use RL and sparse rewards to obtain high-level policies but also use the common sense priors, p_{CS} , from the LLM to guide exploration. More details about the RL algorithms used are in the Experiments section. The action selection strategy in the high-level exploration policy π_H usually samples actions from a categorical distribution where the logits are obtained by the policy head processing the state. These logits are biased with the common sense priors p_{CS} and a weight factor λ . So the action selection looks like this: $a = \text{Categorical}[\pi_H(s_t) + \lambda.p_{CS}(s_t)]$. Here, the action a is the temporally extended macro action or the skill. The weight factor starts from $\lambda = 1$ and is annealed gradually until it reaches zero by the end of training. This means that the trained agent does not continue to rely on the LLM during deployment. This process is summarized in Algorithm 1 and Figure 1.

LLM Queries and Prompt Design. We use the *gpt-3.5-turbo* GPT provided by OpenAI APIs. To reduce the number of API calls, the LLM responses for all possible combinations of l_{goal_inst} and l_{traj} are cached. We use the cached responses instead of calling the API throughout the training process. A simplified version of l_{traj} is used to denote the current trajectory history using the past two actions. The main prompt used in our experiments has the following structure

Goal: l_{goal_inst}
So far: l_{traj}
Question: Should I l_{skill_i} ?
Answer :

l_{traj} represents the trajectory history which captures what the agent has done so far. We find that the 2 actions are sufficient to capture this since the tasks we test on are not extremely complicated. As mentioned in previous sections, l_{goal_inst} describes the goal along with details, warnings and constraints. Each skill is associated with a language description l_{skill_i} . The LLM is shown a few examples of responses to such queries and the prompt specifies that a one word Yes/No answer is required.

Experiments

This section describes the experimental setup and results of testing the framework in three simulation environments and one real world robotic arm block manipulation task. The framework relies on communicating with the LLM using text. As mentioned earlier, each skill corresponds to a text description l_{skill_i} and the larger goal is described using l_{goal_inst} . We assume access to a captioner that maps the current observation history to l_{traj} . This could be automated by using modern vision to language models such as (Radford et al. 2021), but that is left for future work. Instead we use a CLIP-based model along with an LLM in the experiments with a real robot to convert visual input to a discrete low dimensional state. In each environment, our method is compared with baseline hierarchical agents without any guidance from LLMs, and an oracle and a SayCan-like agent without affordances.

Table 1: A description of the methods used in the experiments

| Method | Description |
|------------------|---|
| LLM x HRL (ours) | Use an LLM to bias high-level action selection as explained in Section 3. Only receives reward on task completion. |
| Vanilla HRL | A baseline hierarchical agent that has no guidance from the LLM. |
| Shaped HRL | Same as the Vanilla HRL with no LLM guidance. But here agents receive shaped rewards for successful sub-task completion. Requires hand-engineered reward functions. |
| Oracle | This is the upper bound. The high-level policy is an oracle state-machine that provides the right sub-tasks in the correct sequence. (Goecks et al. 2021) |
| SayCan w/o Aff | A SayCan (Ahn et al. 2022) like architecture but without an affordance function that blindly trusts the LLM. This method requires LLM access during deployment |

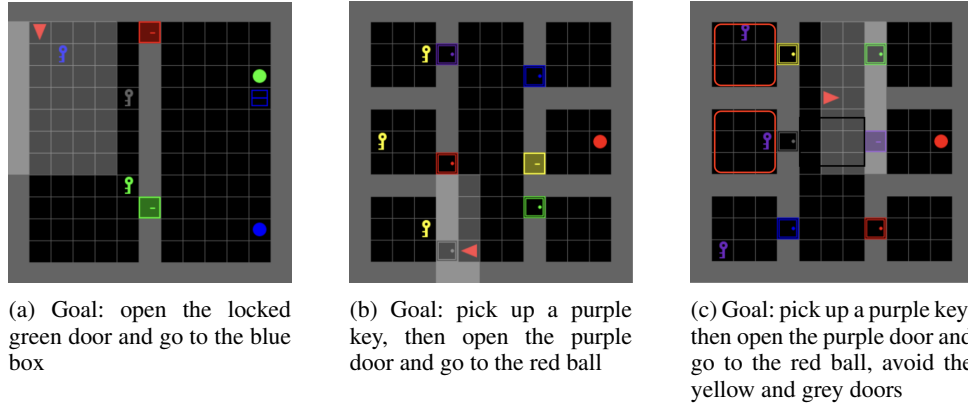


Figure 2: The three tasks in the MiniGrid environment.

MiniGrid Experiments

Setup The experiments described in this section were performed on the MiniGrid environment by (Chevalier-Boisvert, Willems, and Pal 2018), which is a simple grid world. The environment can be designed with multiple rooms with doors, walls, and goal objects. These objects can have different colors and the agent and goal objects are spawned at random locations. The action space is discrete which allows movement in the 4 compass directions, opening and closing doors, and picking up and dropping objects. We designed multiple tasks in this setup which can be broken down into smaller sub-tasks as shown in Figure 2.

- The *UnlockReach* task consists of a random object in a room which are behind a locked door. The agent has to first find the right key based on the door color, unlock the door, and then navigate to the goal object.
- The *KeyCorridor v0* task consists of a corridor with multiple rooms on either side. A goal object is inside a locked room whose key is in another room. The agent has to first find the key and then unlock the door to ultimately reach the goal
- *KeyCorridor v1* is similar to v0, but some of the rooms have defective keys. The goal instruction comes with the rooms to avoid. This task is much more difficult for standard HRL methods.

Each task has a single reward that is only provided on successful task completion. The agents have access to several temporally extended skills: *GoToObject*, *PickupObject*, *UnlockDoor*, and *OpenBox*. These are conditioned on the type and color of the objects. For example the *Object* may refer to a *key*, *ball*, or *box*, and the color can be *red*, *green*, *blue*, *yellow*, etc. These low-level sub-tasks were pretrained and frozen using PPO (Schulman et al. 2017) and manual reward specification. The high-level policies are also trained using PPO where the sub-tasks are treated as actions. We compared against Vanilla HRL, Shaped HRL, an Oracle, and a SayCan-like method as described in Table 1. The results are summarized in Figure 3. It’s clear that our method outperforms both the baseline HRL methods with and without shaped rewards. It is also able to converge to the optimal policy much sooner than the other methods. The Oracle and SayCan are not trained using RL and so we show their performance using horizontal lines. Although they are comparable to our method, one benefit of our method is that it does not rely on the LLM during deployment.

SkillHack

The NetHack Learning Environment (Küttler et al. 2020) is an RL environment based on the classic game of NetHack. It is notoriously difficult because of the large number on entities and actions, procedural generation, and the stochastic nature of the game. MiniHack (Samvelyan et al. 2021)

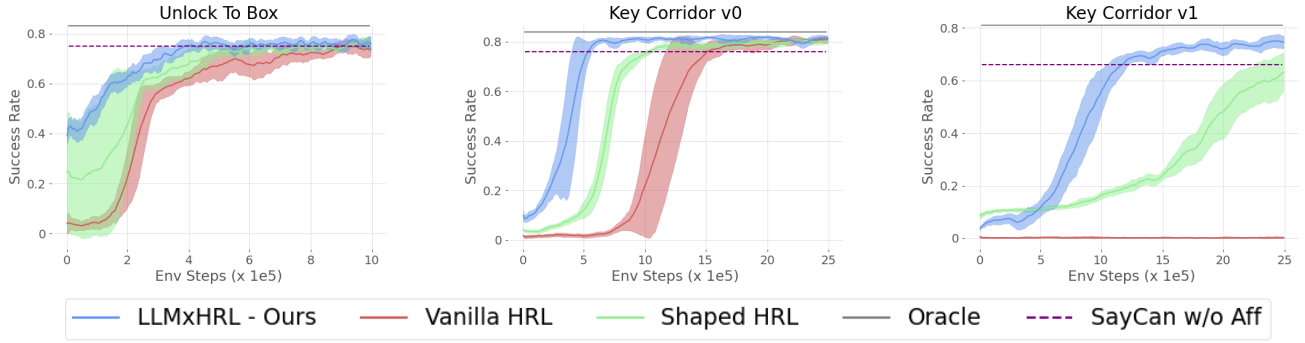
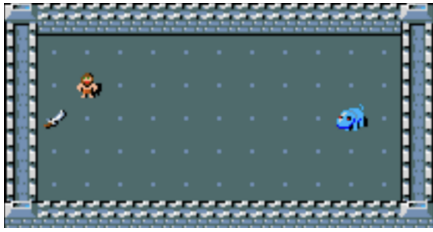
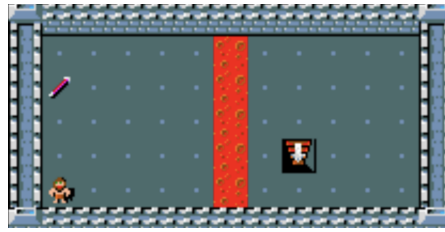


Figure 3: The plots show the success rate of different methods on the three tasks in the MiniGrid Environment. Our method consistently outperforms the baseline hierarchical agents both with and without intermediate rewards. The success rate is marginally better than SayCan without affordance functions and is comparable to the oracle.



(a) Battle: The goal is to pick up the sword, navigate to the monster and fight



(b) FrozenLavaCross: Here the goal is to pick up the wand, navigate to the lava to zap, creating a bridge and then navigate to the exit



(c) Crafter Env: 2D Minecraft like environment

Figure 4: Left and Middle: The SkillHack environment suite based on the text based NetHack game. Right: Crafter: Here, the agent explores the open world environment to collect resources like wood, stone and create new objects and tools.

and SkillHack (Matthews et al. 2022) are extensions of NetHack that enable creation of custom levels and tasks. They are simpler than the full game while retaining most of the interesting complexities. The SkillHack suite contains 16 skills such as *PickUp*, *Navigate*, *Fight*, *Wear*, *Weild*, *Zap*, *Apply*, etc. These skills can be executed sequentially to achieve larger tasks. We consider two such tasks - *Battle*, *FrozenLavaCross*, Figure 4.

- In the *Battle* task, the agent needs to *PickUp* a randomly placed Sword, *Wield* the Sword and finally *Fight* and kill a Monster.
- In the *FrozenLavaCross* task, the agent needs to *PickUp* either a WandOfCold or a FrostHorn based on what is available, then create a bridge across the lava with either *ZapWandOfCold* or *ApplyFrostHorn*. Finally, the agent must *NavigateLava* across the newly made bridge to reach the staircase on the other side.

In this environment we compare against Vanilla HRL and an Oracle high-level policy. The low level skills are trained using IMPALA (Espeholt et al. 2018) with the code provided by (Matthews et al. 2022). The high-level policy is also trained using IMPALA where the policy skills are macro actions. As seen in the first two plots in Figure 5, in both the tasks, *Battle* and *FrozenLavaCross*, our method clearly outperforms the HRL agent without LLM guidance.

Crafter

Crafter (Hafner 2021) is a 2D version of Minecraft which has the same complex dynamics but with a simpler observation space and faster simulation speeds. Similar to Minecraft, it involves collecting and building artifacts along an achievement tree. We modified the game slightly to make it easier by slowing down health degradation and having fewer dangers to fight. We evaluated on two tasks that have a natural hierarchical structure - *MakeWoodPickaxe* and *MakeStonePickaxe*, Figure 4. The state consists of a simple representation encoding the 2D map, the items on the map, an inventory and the health of the agent. The high-level skills l_{skill_i} we consider such as *chop tree*, *create table*, *make wood pickaxe* etc can also be naturally described using language phrases. Similar to our other experiments we pre-train policies for multiple skills using PPO. The high level policy is then trained to select among these skills. The last two plots in Figure 5 show how our method performs better than the baseline HRL method.

uArm Real Robot Experiments

We also tested on a real robot arm on a simpler tabular Q-learning version of our method. uArm Swift Pro (uArm Developer 2018) is an open-source desktop robotic arm. We designed two block manipulation tasks - *DeskCleanUp*

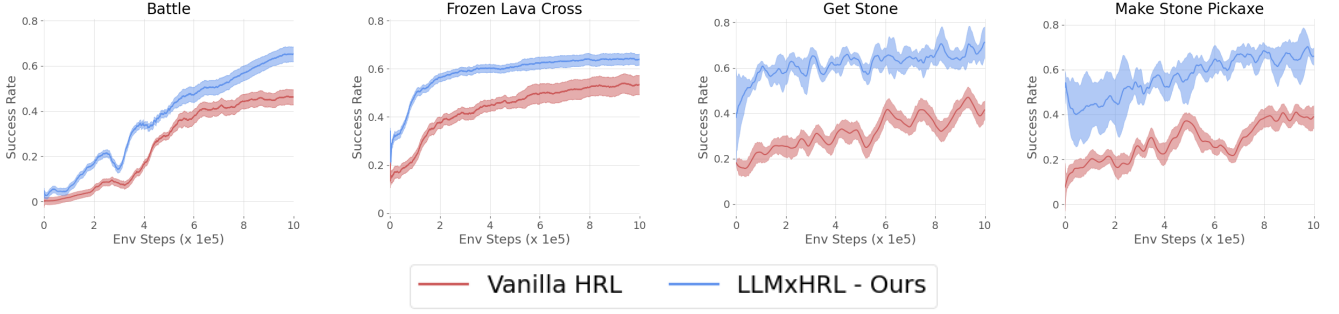


Figure 5: The 2 plots on the left show the success rate of different methods on the SkillHack - Battle and Frozen Lava Cross. The 2 plots on the right show the success rate of different methods on the Crafter - Get Stone and Make Stone Pickaxe

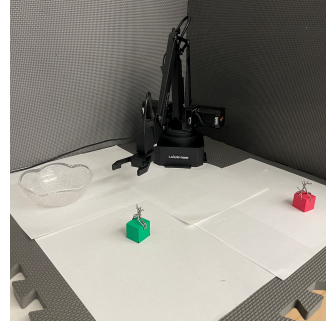
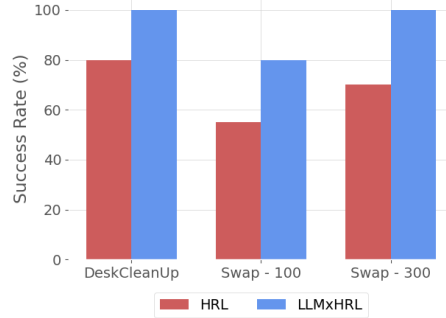


Figure 6: Left: The results showing the success rate for the uArm experiments. *Swap - 100* denotes performance after 100 episodes and *Swap - 300* is after 300 episodes. We can see that using LLMs to guide agent exploration give us better performance in fewer trials. Right: The robot setup for the desk cleanup task

and *SwapBlocks*. Similar to the previous simulation experiments, we assume access to various skills that can be used to solve larger, more complex tasks. In our setup, video from a camera is used to convert the robot arm and block positions into an array of discrete values representing the state. From this simplified state, we are able to learn a high-level policy with tabular Q-learning. Like before, we calculate $p_{CS} = \log\text{-softmax}(F_{LLM})$ using an LLM and access to $f_{LLM}(l_{skill_i}, l_{goal_inst}, l_{traj})$.

Figure 6 show the results of our experiments. In the *DeskCleanUp* task, there are three locations with a tray and two blocks (red and green). The episode is initialized with blocks in random locations. The goal is to pick up the blocks and place them in the tray, essentially cleaning the desk. This task was trained for 100 episodes. In the *SwapBlocks* task there are also three locations (or zones) with two blocks in two random locations. The goal is to swap the positions of the blocks. In Figure 6, *Swap - 100* denotes performance after 100 episodes and *Swap - 300* is after 300 episodes. We can see that using LLMs to guide agent exploration give us better performance in fewer trials.

Discussion

In this work we present a framework for using LLMs to guide exploration in hierarchical agents. Instead of learning from random exploration without any prior knowledge, we

use the LLMs to suggest high-level actions based on the task and current state. We evaluate our method on long horizon tasks in simulation environments as well as with a real robot. We show that our method performs better than baselines and does not require manual reward shaping. Moreover, once the agent is trained, it no longer depends on the LLM during deployment unlike some prior methods.

One limitation of our work is that we require LLM access during the training process. We reuse the LLM responses by caching in our experiments which is not always be feasible. Exploring more ways to reduce this dependency is potential future work. Another drawback is that we assume a fixed set of skills or options to train the high level policy. In reality, this restrict the the agent’s behaviour. Extending the same method with continuous space of skills like (Pertsch, Lee, and Lim 2020) along with vision to language models is an exciting future direction. This work can also be extended in several ways to make it more end-to-end. We currently assume access to a function that provides us language descriptions of the current trajectory and state. This can be automated using recent advancements in vision language models (VLM). It will also be interesting to extend this framework for more than one level or hierarchy to tackle longer tasks.

References

- Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.
- Andreas, J.; Klein, D.; and Levine, S. 2017. Modular multi-task reinforcement learning with policy sketches. In *International Conference on Machine Learning*, 166–175. PMLR.
- Bacon, P.-L.; Harb, J.; and Precup, D. 2017. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.
- Bommasani, R.; Hudson, D. A.; Adeli, E.; Altman, R.; Arora, S.; von Arx, S.; Bernstein, M. S.; Bohg, J.; Bosselut, A.; Brunskill, E.; et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.
- Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33:1877–1901.
- Chevalier-Boisvert, M.; Willems, L.; and Pal, S. 2018. Minimalistic gridworld environment for openai gym. <https://github.com/maximecb/gym-minigrid>.
- Christiano, P. F.; Leike, J.; Brown, T.; Martic, M.; Legg, S.; and Amodei, D. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, 4299–4307.
- Du, Y.; Watkins, O.; Wang, Z.; Colas, C.; Darrell, T.; Abbeel, P.; Gupta, A.; and Andreas, J. 2023. Guiding pre-training in reinforcement learning with large language models. *arXiv preprint arXiv:2302.06692*.
- Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, 1407–1416. PMLR.
- Fruit, R., and Lazaric, A. 2017. Exploration-exploitation in mdps with options. In *Artificial Intelligence and Statistics*, 576–584. PMLR.
- Goecks, V. G.; Waytowich, N.; Watkins-Valls, D.; and Prakash, B. 2021. Combining learning from human feedback and knowledge engineering to solve hierarchical tasks in minecraft. *arXiv preprint arXiv:2112.03482*.
- Goyal, P.; Niekum, S.; and Mooney, R. J. 2019. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*.
- Hafner, D. 2021. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*.
- Hu, H.; Yarats, D.; Gong, Q.; Tian, Y.; and Lewis, M. 2019. Hierarchical decision making by generating and following natural language instructions. *Advances in neural information processing systems* 32.
- Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, 9118–9147. PMLR.
- Jiang, Y.; Gu, S. S.; Murphy, K. P.; and Finn, C. 2019. Language as an abstraction for hierarchical deep reinforcement learning. *Advances in Neural Information Processing Systems* 32.
- Küttler, H.; Nardelli, N.; Miller, A.; Raileanu, R.; Selvatici, M.; Grefenstette, E.; and Rocktäschel, T. 2020. The nethack learning environment. *Advances in Neural Information Processing Systems* 33:7671–7684.
- Le, H.; Jiang, N.; Agarwal, A.; Dudik, M.; Yue, Y.; and Daumé III, H. 2018. Hierarchical imitation and reinforcement learning. In *International conference on machine learning*, 2917–2926. PMLR.
- Matthews, M.; Samvelyan, M.; Parker-Holder, J.; Grefenstette, E.; and Rocktäschel, T. 2022. Hierarchical kickstarting for skill transfer in reinforcement learning.
- Mirchandani, S.; Karamcheti, S.; and Sadigh, D. 2021. Ella: Exploration through learned language abstraction. *Advances in Neural Information Processing Systems* 34:29529–29540.
- Pertsch, K.; Lee, Y.; and Lim, J. J. 2020. Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning (CoRL)*.
- Prakash, B.; Waytowich, N.; Oates, T.; and Mohsenin, T. 2021. Interactive hierarchical guidance using language. *arXiv preprint arXiv:2110.04649*.
- Radford, A.; Kim, J. W.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, 8748–8763. PMLR.
- Samvelyan, M.; Kirk, R.; Kurin, V.; Parker-Holder, J.; Jiang, M.; Hambro, E.; Petroni, F.; Kuttler, H.; Grefenstette, E.; and Rocktäschel, T. 2021. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.
- Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1-2):181–211.
- uArm Developer. 2018. uArm-Python-SDK.
- Warnell, G.; Waytowich, N.; Lawhern, V.; and Stone, P. 2018. Deep tamer: Interactive agent shaping in high-dimensional state spaces. *AAAI Conference on Artificial Intelligence* 1545–1553.
- Waytowich, N.; Barton, S. L.; Lawhern, V.; and Warnell, G. 2019. A narration-based reward shaping approach using grounded natural language commands.