LLM Augmented Hierarchical Agents

Bharat Prakash[1]**, Tim Oates**[1]**, Tinoosh Mohsenin**[1, 2]
University of Maryland, Baltimore County[1]
Johns Hopkins University [2]

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 tasks 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.

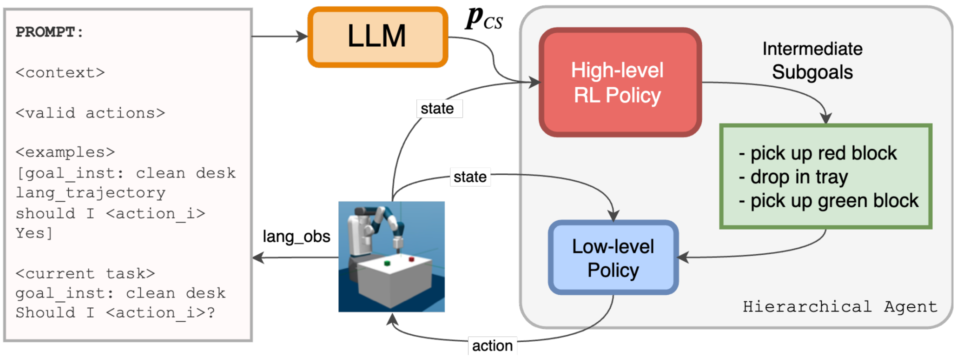
1 Introduction

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 the rovers in space, 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 Sutton et al. [1999], Fruit and Lazaric [2017], Bacon et al. [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 handle on newer cars lie

FMDM Workshop @ 37th Conference on Neural Information Processing Systems (NeurIPS 2023).



```
PROMPT:

<context>

<valid actions>

<examples>
[goal_inst: clean desk
lang_trajectory
should I <action_i>
Yes]

<current task>
goal_inst: clean desk
Should I <action_i>?
```

Figure 1: We use the LLM to guide the high-level policy and accelerate learning. The LLM is prompted with the context, some examples, and the current task and observation. We then use the LLM output to bias the high-level action selection

flat against the door, unlike every other car door handle 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. We also want to avoid relying fully on the LLMs and hence only use them to bias action selection. Moreover, we gradually anneal the LLM's influence as training progresses. This also results in a policy which can be deployed without depending on the LLM. We evalaute this method on several simulation environments, MiniGrid Chevalier-Boisvert et al. [2018], SkillHack Matthews et al. [2022], and Crafter Hafner [2021]. We show that this method can learn to solve complex long horizon tasks much faster then baseline methods. We also evaluate this method on a real robot arm in block manipulation tasks using the tabular Q-learning version of the same algorithm. These results also show how we can learn policies much faster with fewer episodes. Our contributions are summarized as follows:

- We propose an approach for using LLMs to guide exploration by extracting common sense priors.
- We present a hierarchical agent which uses these priors to solve long horizon tasks
- We evaluate the framework in simulation as well as a simple real world environment and show that is performs better than baselines.
- We discuss the advantages of our method compared similar prior methods and potential future work

2 Related Work

Langauge and HRL There has been a lot of work on hierarchical RL where the standard MDP is converted into a SMDP, a semi Markov decision process. The most common approach is to incorporate temporally extended actions also known as options or skills Bacon et al. [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. Language has been a popular way to specify sub-tasks and achieve generalization due to its inherent properties such as compositionality and hierarchical structure Jiang et al. [2019], Prakash et al. [2021], Waytowich et al. [2019], Hu et al. [2019]. Most of these methods generate a high-level plan in natural language, which is then executed sequentially by a separate low-level policy model. These approaches still have a lot of challenges when operating in high-dimensional observation spaces. They also rely on manual data collection and to train the high-level policies and therefore difficult to generalize to new tasks.

RL and Foundation Models Recently, large language models such as GPT-3 have been used to design agents capable of acting in the real world based on language instructions Brown et al. [2020], Bommasani et al. [2021]. The in-context learning along with intelligent prompting strategies shown by these models have been used to design language guided hierarchical agents. Huang et al. [2022] use LLMs as zero-shot planners to enable embodies agents to act in the real world scenarios. Similarly, Ahn et al. [2022] use LLMs along with affordance functions to generate feasible plans which guides a robot to acheive goals specified in 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.

3 Methods

3.1 Problem Statement

We consider a system where it receives an instruction in the form of natural language describing the task, similar to Ahn et al. [2022]. The instruction can be long, may contain warnings and constraints and may not include all the individual steps. We also assume that the agent has access to a finite set of skills or sub-policies, which can be executed in sequence to solve long-horizon tasks. These skills can be either hard-coded if possible or trained using reinforcement learning or imitation learning with manual reward design. These skills are accompanied by a language description, for example, "pick up red block", "open blue door" etc. These policies also detect sub-task completion in order to switch control back to the high level policy. Given a finite set of options or skills, our objective is to obtain a high-level decision-policy that selects among them.

3.2 Using LLMs to Guide High-level policy

In this section we introduce our method, which uses LLMs to improve exploration in the high level policy. Given a task description and current state in the form of language, we want to use the semantic knowledge and planning capabilities of LLMs to improve high-level action selection. The core idea of our method is to use LLMs to obtain a quantity that captures the probability that a given skill or sub-task is relevant in order to achieve the larger task. As mentioned earlier, each skill is accompanied by a language description l_{skill} and the current trajectory is translated into language, l_{traj}. We also have a high level instruction, l_{goal_inst} describing the larger goal along with optional constraints.

We want to use the LLM to evaluate the function $f_{LLM}(l_{skill_i}, l_{goal\_inst}, l_{traj})$ for each skill at every high-level decision step. Essentially, the LLM is used to answer the following question. Given the task l_{goal_inst}, and trajectory so far l_{traj}, should we choose the skill l_{skill}? The output of the LLM, ‘yes’ or ‘no’, can easily be converted to an int (“0" or “1"). We chose to use this kind of closed form question-answering prompts as they have shown to work better than open ended prompts Du et al. [2023]. After evaluating this for each of the k skills, we get $F_{LLM} = [f_{LLM_1}, f_{LLM_2}, f_{LLM_3}, ..., f_{LLM_k}]$. For example, $F_{LLM} = [0, 1, 0, ..., 0, 1, 0, 0]$. These are considered as logits and we apply a LOG SOFTMAX function to it. Ultimately we get the common sense priors from the LLM denoted by $p_{CS} = log\_softmax(F_{LLM})$. 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 LLMs to guide the exploration. More details about the RL algorithms used are in the Experiments section. The action selection in the exploration policy samples actions from a categorical distribution where the logits are obtained by the policy head processing the state. We bias these logits with the common sense priors p_{CS} and a weight factor λ. So the action selection looks like this - $a = \texttt{Categorical}[\pi(s_t) + \lambda.p_{CS}(s_t)]$. 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 our trained agent does not continue to reply on the LLM during deployment. The process is summarized in Algorithm 1 and Figure 1.

Algorithm 1

```
high_inst ← high level goal in language
$\pi_\theta$ ← high level policy
$f_{LLM}$ ← 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
            $p_{CS} \leftarrow f_{LLM}(high\_inst, \tau)$
            $a_t \leftarrow cat\_dist[\pi_\theta(\tau) + \lambda.p_{CS}(\tau)].sample()$
            $s_t, r_t \leftarrow act(a_t)$
            $\tau \leftarrow append(s_t, r_t, a_t)$
        end for
        update $\pi_\theta$
    end while
    return $\pi_\theta$
end procedure
```

LLM Queries and Prompt Design. We use the *gpt-3.5-turbo* GPT provided by OpenAI APIs. In order to reduce the calls to the APIs, we cache the LLM responses for all possible combinations of l_{goal_inst} and l_{traj}. We use a very simplified version of l_{traj} to denote the current trajectory history using the past 2 action, refer to Appendix. The main prompt we use in our experiments has the following structure `Goal:`l_{goal_inst}`, So far I have:` l_{traj}`, Should I` l_{skill_i}`?` . We provide few examples of responses to such queries and specify that we expect one word Yes/No answers. Example prompts are in the Appendix.

4 Experiments

In this section, we describe the experimental setup and results where we test out framework in three simulation environments and one real world robotic arm block manipulation environment. Our framework relies on communicating with the LLM using text. As mentioned earlier each skill corresponds to a text description l_{skill_i} and the high level goal, l_{goal_inst}. We assume access to a captioner which maps the current observation historiy to l_{traj}. Potentially this can be automated by using modern vision to language models such as Radford et al. [2021] but leave that for future work. However we do use a a CLIP based model along with an LLM in our real robot experiments to convert visual input to a discrete low dimensional state. More details about obtaining l_{traj} is in the Appendix. In each environment, we compare our method with baseline hierarchical agents without any guidance from LLMs. An oracle and a SayCan like agent without affordance; see Table 1.

4.1 MiniGrid Experiments

Setup These experiments are performed on the MiniGrid Environment by Chevalier-Boisvert et al. [2018] which is a simple grid world environment. The environment can be designed with multiple rooms with doors, walls and goal objects. These entities can have different colors and the agent and goal objects are spawned at random locations. The action space is discrete which allows movement in all 4 directions, opening and closing doors and picking up and dropping objects. We designed multiple long tasks in this setup which can be broken down into smaller sub tasks.

- *UnlockReach* task consists of a random object in a room which is behind a locked door. The agent has to first find the right key based on the door unlock it and then navigate to the goal object.
- *KeyCorridor v0* task consists of a corridor with multiple rooms on either ride. A goal object is inside a locked room which has keys is in another rooms. The agent has to first find the key and then unlock the door to ultimately reach the goal

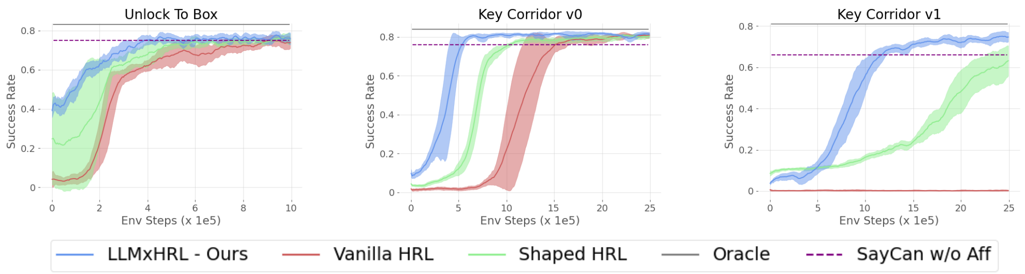


Figure 2: The plots show the success rate of different methods on the three tasks in the MiniGrid Environment.

- *KeyCorridor v1* is similar to v0, but some of the rooms have defective keys. The goal instruction comes with the rooms to avoid. This makes is much more difficult than the simpler version.

All these tasks have a sparse reward that is only provided in the end after successful task completion. The agents have access to several temporally extended skills - *GoToObject, PickupObject, UnlockDoor, OpenBox*. These are conditioned on the type and color of the objects. For example the *Object* may refer to key, ball, box and the color can be *red, green, blue, yellow* etc. These low-level subtasks 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 . 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 2 We see that our method outperform both the baseline HRL methods with and without shaped rewards. It is also able to converge to the optimal policy much sooner that other methods. The Oracle and SayCan are not trained using RL and so we show their performance using the horizontal lines. Although they are comparable to our method, one benefit of your method is that is does not rely on the LLM during deployment.

4.2 SkillHack

The NetHack Learning Environment Küttler et al. [2020] is an RL environment based off the classic game of NetHack. It is notoriously difficult because of the large number on entities, actions, procedural generation and stochastic nature of the game. MiniHack Samvelyan et al. [2021] and SkillHack Matthews et al. [2022] are extensions on top of it to that enables creation of custom levels and tasks but more manageable than the full game while still retaining the complexities. The SkillHack suite contains 16 skills such as *PickUp, Navigate, Fight, Wear, Weild, Zap, Apply* etc.

Method	**Description**
LLM x HRL (ours)	Use LLMs to bias high-level action selection as explained in Section 3. Only receives reward in the end at task completion.
Vanilla HRL	A baseline hierarchical agent which has no guidance from the LLMs.
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 which provides the right sub-tasks in the correct sequence. Goecks et al. [2021]
SayCan w/o Aff	A SayCan Ahn et al. [2022] like architecure but without an affordance function and blindly trusting the LLM. This method will depend on LLM access during deployment

Table 1: This table lists the all methods we compared

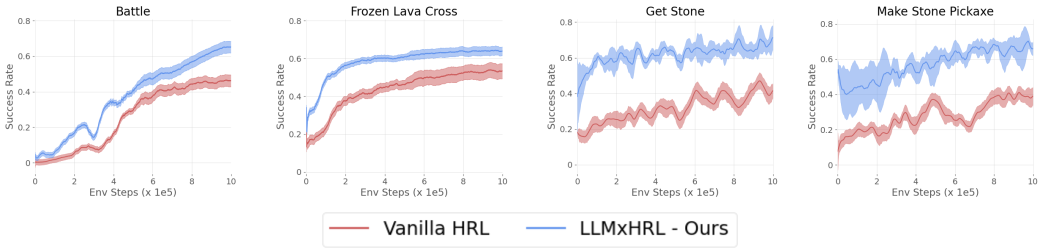


Figure 3: The 2 plots on the left show the success rate of different methods on the SkillHack - Battle and Frozen Lava Cross. The 2 plots on the right show the success rate of different methods on the Crafter - Get Stone and Make Stone Pickaxe

More details in the Appendix. These skills can be executed sequentially to achieve larger tasks. We consider two such tasks - *Battle, FrozenLavaCross*.

- In the *Battle* task, the needs to *PickUp* a randomly placed Sword, *Wield* the Sword and finally *Fight* and kill a Monster.
- In the *FrozenLavaCross* task, the 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 by *ApplyFrostHorn*. Finally, *NavigateLava* across your 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 are trained using IMPALA Espeholt et al. [2018] using 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 3, in both the tasks, *Battle and FrozenLavaCross*, our method clearly outperforms the HRL agent without LLM guidance.

4.3 Crafter

Crafter Hafner [2021] is a 2D version Minecraft which has the same complex dynamics but with easier observation space and faster simulation speeds. Similar to Minecraft, it involves collecting and buildign artifacts along an achievement tree. We modified the game slightly to make it easier by slowing down health degradation and fewer dangers to fight. We evaluate on 2 tasks that have a natural hierarchical structure - *MakeWoodPickaxe and MakeStonePickaxe*. More details in the Appendix. Similar to our other experiments we pretrain policies for multiple skills using PPO. The high level policy is then trained to select among these skills. The last two plots in 3 shows how our method performs better than the baseline HRL method.

4.4 uArm Real Robot Experiments

We also test on a real robot arm on a simpler tabular Q-learning version of our method. uArm Swift Pro uArm Developer [2018] is a open-source desktop robotic arm. We design two block manipulation tasks - *DeskCleanUp and SwapBlocks*. Similar to previous simulation experiments, we assume access to various skills which can be used to solve longer tasks. In our setup we process the visual inputs using a camera and the robot arm positions into an array of discrete values representing the state.From this simplified state, we are able to learn a high-level policy using tabular Q-learning. Like before, we calculate $p_{CS} = log\_softmax(F_{LLM})$ using an LLM and access to $f_{LLM}(l_{skill_i}, l_{goal\_inst}, l_{traj})$. Refer to the Appendix for more details.

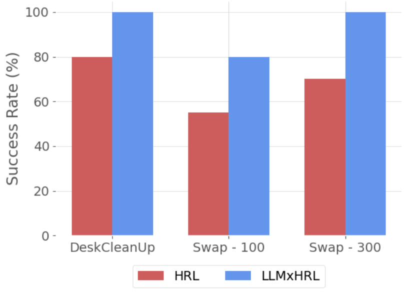


Figure 4: Robot Arm Results

Figure 4 show the results of our experiments. In the *DeskCleanUp* task, we have a 3 locations where we have a tray and 2 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 again have 3 locations (or zones) and 2 blocks in 2 random locations. The goal is to swap the position of blocks. In the Figure 4, *Swap - 100* denotes performance after 100 episodes and *Swap - 300*, after 300 episodes. We can see that using LLMs to guide agent exploration give us better performance in fewer trials.

5 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 which in simulation environments as well as a real robot. We show that out method perform better than baselines and does not require manual reward shaping. Moreover, once the agent is trained, we no longer depend on the LLM during deployment unlike some prior methods.

This work can be extended in several ways to make to more end to end. We Currently assume access to a function which provides us to 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 and one level or hierarchy to tackle longer tasks.

References

M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

M. Chevalier-Boisvert, L. Willems, and S. Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

Y. Du, O. Watkins, Z. Wang, C. Colas, T. Darrell, P. Abbeel, A. Gupta, and J. Andreas. Guiding pretraining in reinforcement learning with large language models. *arXiv preprint arXiv:2302.06692*, 2023.

L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

R. Fruit and A. Lazaric. Exploration-exploitation in mdps with options. In *Artificial Intelligence and Statistics*, pages 576–584. PMLR, 2017.

V. G. Goecks, N. Waytowich, D. Watkins-Valls, and B. Prakash. Combining learning from human feedback and knowledge engineering to solve hierarchical tasks in minecraft. *arXiv preprint arXiv:2112.03482*, 2021.

D. Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

H. Hu, D. Yarats, Q. Gong, Y. Tian, and M. Lewis. Hierarchical decision making by generating and following natural language instructions. *Advances in neural information processing systems*, 32, 2019.

W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

Y. Jiang, S. S. Gu, K. P. Murphy, and C. Finn. Language as an abstraction for hierarchical deep reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33: 7671–7684, 2020.

H. Le, N. Jiang, A. Agarwal, M. Dudik, Y. Yue, and H. Daumé III. Hierarchical imitation and reinforcement learning. In *International conference on machine learning*, pages 2917–2926. PMLR, 2018.

M. Matthews, M. Samvelyan, J. Parker-Holder, E. Grefenstette, and T. Rocktäschel. Hierarchical kickstarting for skill transfer in reinforcement learning, 2022. URL `https://arxiv.org/abs/2207.11584`.

B. Prakash, N. Waytowich, T. Oates, and T. Mohsenin. Interactive hierarchical guidance using language. *arXiv preprint arXiv:2110.04649*, 2021.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Kuttler, E. Grefenstette, and T. Rocktäschel. 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)*, 2021. URL `https://openreview.net/forum?id=skFwlyefkWJ`.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

uArm Developer. uArm-Python-SDK, 2018. URL `https://github.com/uArm-Developer/uArm-Python-SDK`.

G. Warnell, N. Waytowich, V. Lawhern, and P. Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. *AAAI Conference on Artificial Intelligence*, pages 1545–1553, 2018. URL `https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16200`.

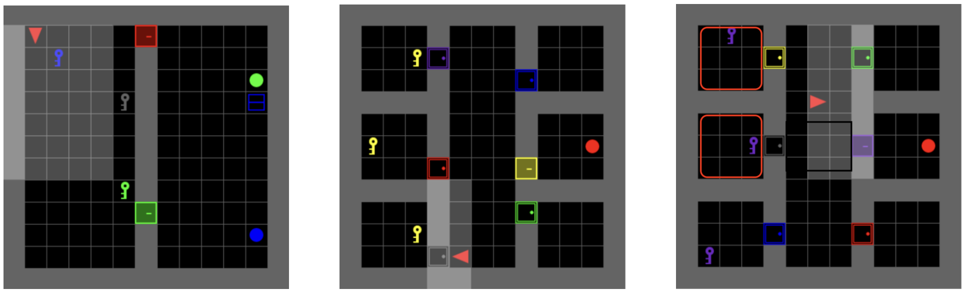
N. Waytowich, S. L. Barton, V. Lawhern, and G. Warnell. A narration-based reward shaping approach using grounded natural language commands, 2019.

A Appendix

A.1 Environment Details

A.1.1 MiniGrid

Minigrid is an open source gridworld environment Chevalier-Boisvert et al. [2018]. We use three tasks *UnlockReach, KeyCorridor v0 and KeyCorridor v1*. Figure 5 shows the grid layouts for the three tasks. The observation is an encoded version of the grid which capture the each cell type, color and an optional door/box state. We consider the fully observable version of these tasks, which means the observations consists of the full grid - 13x13 in our case.



(a) Goal: open the locked green door and go to the blue box

(b) Goal: pick up a purple key, then open the purple door and go to the red ball

(c) Goal: pick up a purple key, then open the purple door and go to the red ball, avoid the yellow and grey doors

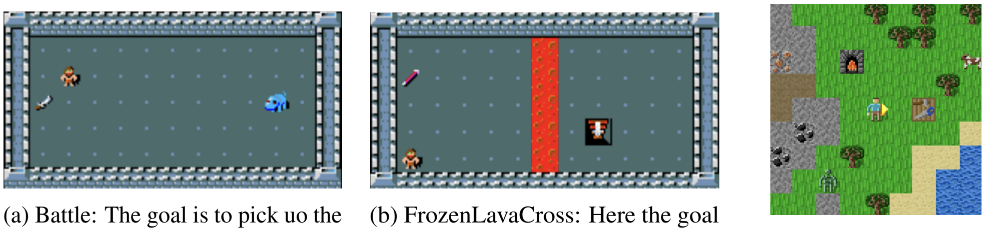
Figure 5: The agent is represented using the red triangle. Left: The UnlockReach task where the agent needs to get the right key and open a door and then go to the object in the right room. Middle: The KeyCorridor-v0 task where the agent needs to reach the red ball in one of the locked rooms on the right. It first needs to get a the key to from one of the rooms on the left. Right: Similar to v0 but the some of the rooms have defective keys shown in red. The agent does not see this, it only recieves this information in the text goal

A.1.2 SkillHack

SkillHack Matthews et al. [2022] is an extensions on top of Küttler et al. [2020] which where you can design custom levels and get visual/spacial states along with text descriptions. Figure 6 shows the tasks we test where each of them require solving multiple sub-tasks. The state consists of a 2D map along with inventory information and text describing the effect of each action. This is very convenient for out method as we need to interact with the LLM using language.

A.1.3 Crafter

Crafter Hafner [2021] is a 2D version Minecraft, Figure 6. it has a very simple state representation encoding the items on the map, an inventory and the health of the agent. It is fairly easy to translate this into text using a hang-coded function. 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 langauge phrases.



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

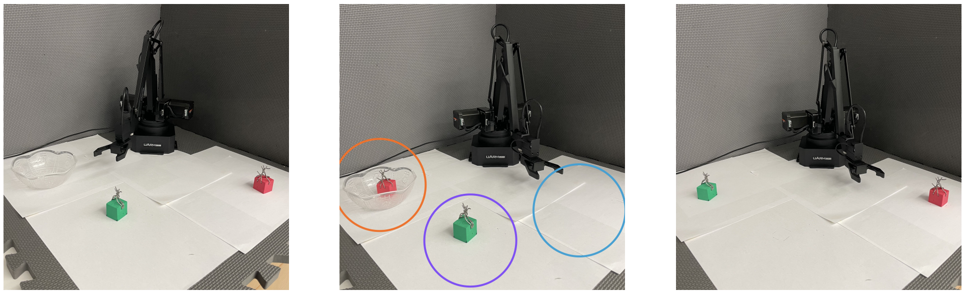
(b) FrozenLavaCross: Here the goal is to pick up the wand, navigate to the lava to zap, creating a bridge and then navigate to the exit

(c) Crafter Env: 2D Minecraft like environment

Figure 6: 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 craete new objects and tools

A.1.4 uArm Robot Arm

uArm Swift Pro is an open source research robotic arm. We use this to test the tabular Q-leraning version of our method using two tasks. The environment is setup where the table has 3 zones as shown in Figure 7 (b). The robot is able to pick and place objects from any of these zones. They are considered the low level skills and are hard-coded. However with more resources, these can be trained using RL making them more robust. In the *DeskCleanup* task, there is a tray in one of the zones as seen in Figure 7 (a) and (b). The goal is to pick up the blocks and place them in the tray. In the *BlockSwap* task the two blocks are placed in random zones Figure 7 (c). The goal is to swap the positions. This can only be done by utilizing the empty zone. We use an Intel Realsense camera to convert the visual information to a simplified discrete state for the tabular Q-learning. The state consists of 4 dicrete values, $[arm\_location, holding, red\_location, green\_location]$. We get the $arm\_location$ form the robot apis, which is one of the 3 zones. The $red\_location$ and $green\_location$ denote the location of the red and green blocks respectively (one of 3 zones). $holding$ denotes the block the arm is currently holding if any. We extract $holding, red\_location$ and $green\_location$ from the camera image. Instead of training or finetuning a separate object detection model we use CLIP and text phrases describing the objects on image patches. We then get the co-ordinates and map them to the right values in the state space.



(a) Desk Cleanup Task

(b) Desk Cleanup Task

(c) Block Swap Task

Figure 7: uArm: Robot arm environment

A.2 Prompt Design

We use the following structure for all out prompts.

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

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 as that the tasks we test on 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}.

A.2.1 MiniGrid Prompts

```
You are a 2D maze-solving agent with access to a variety of low-level
skills such as "pick:red:ball", "pick:red:key", "pick:green:ball", "
pick:green:key", "pick:blue:ball"...

Goal: open the locked green door and go to the blue box
So far:
Question: Should I pick:red:ball?
Answer: No

Goal: open the locked green door and go to the blue box
So far:
Question: Should I  goto:blue:box?
Answer: No

Goal: open the locked green door and go to the blue box
So far:  pick:green:key
Question: Should I unlock:green:door?
Answer: Yes

Goal: open the locked green door and go to the blue box
So far:  pick:green:key, unlock:green:door
Question: Should I  goto:red:box?
Answer: No

[..few more examples..]

Goal: open the locked green door and go to the blue box
So far:  pick:green:key, unlock:green:door
Question: Should I  goto:blue:box?
Answer:
```

A.2.2 SkillHack Prompts

The NetHack environment is originally a text based game., So luckily we get the language description of our action and observations from the game engine.

```
You are a NetHack Agent equipped with the following skills:

ApplyFrostHorn:  Use a frost horn to freeze some lava.
Eat: Eat an apple.
Fight: Hit a monster.
NavigateLava: Reach the staircase past random lava patches.
PickUp: Pick up a random item.
PutOn: Put on an amulet or towel.
TakeOff: Take off clothes.
Unlock: Use a key to unlock a locked door.
Wear: Wear a robe.
Wield: Wield a sword.
ZapWandOfCold: Use a wand of cold to freeze lava.

<list common item names>

Battle: PickUp a randomly placed Sword, Wield the Sword and finally
Fight and kill a Monster.

Goal: Battle the Monster
So far: I see a silver saber
Question: Should I TakeOff?
```

```
Answer: No

Goal: Battle the Monster
So far: picked up a silver saber
Question: Should I Wield?
Answer: Yes

Goal: Battle the Monster
So far: picked up a silver saber, weild a silver saber
Question: Should I ZapWandOfCold?
Answer: No

[..few more examples..]

Goal: Battle the Monster
So far: picked up sword, weild sword
Question: Should I Fight?
Answer:
```

```
You are a NetHack Agent equipped with the following skills:

[..same as above..]

Frozen Lava Cross: Either a Wand of Cold or a Frost Horn will spawn on
 the near side of a river of lava. PickUp the item and then create a
bridge across the lava with either ZapWandOfCold or by ApplyFrostHorn
. Finally, NavigateLava across your newly made bridge to reach the
staircase on the other side.

<list common item names>

Goal: Reach the staircase past the lava
So far: I see a wand
Question: Should I PickUp?
Answer: Yes

[..few more examples..]

Goal: Reach the staircase past the lava
So far:  picked up a wand, apply zap, the lava cools and solidifies
Question: Should I NavigateLava?
Answer:
```

A.2.3 Crafter Prompts

For this environment, the prompts are designed similar to Du et al. [2023].