Subgoal Proposition Using a Vision-Language Model

Jianhai Su
University of South Carolina
suj@email.sc.edu

Qi Zhang
University of South Carolina
qz5@cse.sc.edu

Abstract: Recent advances in large language models (LLMs) have inspired research on their potential for robots in real-world tasks. This study investigates whether the architecture of the vision-language model (VLM) Flamingo can help ground the knowledge of a pretrained causal LLM within an agent's experience when following instructions in long-horizon tasks within partially observable environments. To achieve this, we propose the VLM-based hierarchical reinforcement learning (HRL) agent that uses Flamingo's Perceiver Resampler and Cross-Attention mechanism with a causal LLM to capture mission status and suggest promising subgoals. We assess the impact of two design factors, namely the training mode of the pretrained LLM and history representation, through the evaluation of three training modes (*RandomInit*, *TuneAll*, and *FrozenAll*) and two history representations (*Full History* and *Abstract History*). The experiments conducted on the BabyAI platform demonstrate: 1) the VLM-HRL agent outperforms the baseline agent; 2) LLM pretraining is unhelpful for the VLM-HRL agent in environments with less-natural instructions and finetuning serves to undo pretraining; 3) history abstraction enhances learning efficiency and stability.

Keywords: Subgoal Proposition, Language Grounding, Vision-Language Model

1 Introduction

Language serves as the cornerstone of human learning, reasoning, and communication. In many real-world applications, for example, personal assistants, robots are required to process language by design. The advent of large language models (LLMs), like GPT-3 [1] and BERT [2], and their emergent capabilities [3], such as providing solutions via a reasoning process for mathematical problems [4], has motivated researchers to apply pretrained LLMs' knowledge to planning and interactions in robotics for long-horizon tasks specified by high-level natural language instructions. The challenge of effectively grounding the knowledge in pretrained LLMs within an agent's decision-making experience remains an open question, particularly in environments with less-natural language descriptions and less-natural visual observations, like logistics and circuit design.

To ground the knowledge in a pretrained LLM within the agent's experience, the agent must initially gather raw perception data from its available sensors to accurately represent its experiences. A critical aspect of perception is the agent's vision, which can capture essential information about both the agent's behavior and environmental dynamics through a sequence of images. The utilization of an image sequence to portray the agent's behavior and environmental dynamics over time offers two distinct advantages. First, it accommodates partially observable environments by consolidating partial observations, thereby providing the robot with a more comprehensive view of its surroundings. Second, a series of frames effectively represents the execution of a skill related to a subgoal, making it well-suited for hierarchical reinforcement learning settings that capture the semantics of high-level action executions. This inspires us to investigate how to enable the agent to reason subgoals for completing long-horizon tasks specified in natural language, by grounding the knowledge in a pretrained LLM within the agent's experience, i.e., a sequence of subgoals where each comprises its language description and its execution represented by a series of visual observations.

7th Conference on Robot Learning (CoRL 2023), Atlanta, USA.

Flamingo [5], a vision-language model (VLM) architecture that connects pretrained vision and language models to handle sequences of arbitrarily interleaved visual and textual data, suits our needs. The Perceiver Resampler and Cross-Attention mechanism in Flamingo allow for the association of a sequence of images with a sentence, which we hypothesize could facilitate the grounding process under investigation. This leads us to propose the VLM-HRL agent design that adopts Flamingo with a causal LLM inside to serve as the high-level policy in an HRL framework. Our research explores the impact of two design options on learning: Flamingo's LLM training modes, including *RandomInit* (using a randomly-initialized LLM), *TuneAll* (fine-tuning the pretrained LLM), and *FrozenAll* (freezing the pretrained LLM), as well as two history representations, *Full History* (using the entire visual observations) and *Abstract History* (utilizing key observations).

We focus on the BabyAI platform [6] for conducting experiments, which can generate instruction-following tasks in a lightweight partially observable 2-D grid environment but with less-natural language description for a task and less-natural visual observation. From the experiment results, we observe that the VLM-HRL agent (*RandomInit*, *Full History*) can solve tasks much more efficiently than the baseline agent from the BabyAI platform. The baseline agent uses a FiLM-LSTM-based actor-critic model [7] to output a primitive action at each low-level time step. We also discover that the *RandomInit* variant of the VLM-HRL agent outperforms the other two in terms of learning efficiency and stability. Furthermore, history abstraction in VLM-HRL, tailored for resource-constrained, long-horizon tasks, improves learning efficiency and stability.

In summary, the main contributions of this paper are summarized as follows:

- The use of the Flamingo-based VLM-HRL agent design to study the grounding of knowledge in pretrained LLMs within an agent's experience that comprises a list of subgoals, each containing a language description and a sequence of visual observations depicting its execution.
- The findings regarding the training mode of the LLM in Flamingo indicate that LLM pretraining is not helpful for the VLM-HRL agent in long-horizon instruction-following tasks within a partially observable environment featuring less-natural instructions and observations.
- The discovery of the benefits of history abstraction in the VLM-HRL learning.

2 Background

2.1 Goal-conditioned Reinforcement Learning

A goal-conditioned RL problem is formulated by a goal-augmented Markov decision process $\mathcal{M}_g = \langle \mathcal{S}, \mathcal{G}, \mathcal{A}, P, R, \gamma \rangle$ where $\mathcal{S}$, $\mathcal{A}$, P, and γ are standard. Reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow [r_{\min}, r_{\max}]$ is bounded and gives the reward of a state-action pair given some goal $g \in \mathcal{G}$. Under full observability, i.e., the state $s \in \mathcal{S}$ is observable, we aim to find some goal-conditioned policy $\pi : \mathcal{S} \times \mathcal{G} \rightarrow \mathcal{A}$ that, given a goal, maximizes the expected discounted cumulative reward across discrete timesteps $t \in \{0, 1, ...\}$. Under partial observability, $\mathcal{M}_g$ will be further augmented with two elements, Ω and $\mathcal{O}$, such that, at the time t, the agent receives an observation $o_t \in \Omega$ with probability $\mathcal{O}(s_t, a_t)$ where Ω and $\mathcal{O}$ are a set of observations and a set of conditional observation probabilities respectively.

2.2 Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning addresses the challenge of the exploding exploration space long-horizon tasks. The horizon of a task is the average number of actions needed by an agent to solve the task. HRL is based on Semi-Markov Decision Process [8], which is similar to an MDP but with action execution time. HRL breaks down a long-horizon reinforcement learning task into a hierarchy of subtasks, resulting in a shorter task horizon and more structured exploration during the HRL agent's training. The hierarchy of subtasks corresponds to a hierarchy of policies. In this work, a two-level hierarchy is employed: the high-level policy handles task-solving through optimal subtask selection, while the low-level policies are pre-trained to achieve specific subtask goals. We will use subtasks and subgoals interchangeably.

3 Problem Statement

We consider learning a goal-conditioned policy in a partially observable environment. Each goal $g \in \mathcal{G}$ is assumed to be uniquely identified by its language description. Similar to prior work [9], we assume a set of low-level subgoals $\mathcal{G}_l$ such that every task specified by an initial state-goal pair, i.e., (s_0, g), can be optimally accomplished by completing a sequence of subgoals. Further, we assume access to a function, $\Psi : \mathcal{S} \times \mathcal{G}_l \rightarrow \{\text{“Success”}, \text{“Failure”}\}$, that indicates if a subgoal is completed.

The structure of subgoals justifies the HRL approach that consists of a high-level policy and a low-level policy. In existing work, the high-level policy either takes as input the state assuming full observability, or encodes the entire history using a sequence encoder like Transformers [10]. In this work, we consider using Flamingo as the high-level policy, and, crucially, our VLM can takes as input an *abstract* of the entire history. Instead of considering one low-level policy, we assume access to a library of pretrained low-level policies, each of which serves to complete basic tasks (or subgoals) in the environment. In the following sections, we refer to each low-level policy as a skill.

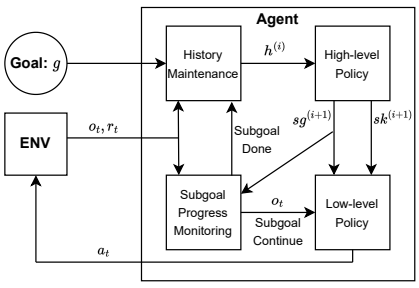
We define the i_{th} subgoal's history $sh^{(i)} = \left(sg^{(i)}, st^{(i)}, et^{(i)}, \{o_t\}_{t=st^{(i)}+1}^{et^{(i)}}, status^{(i)}, sr^{(i)}\right)$ where $sg^{(i)}$, $st^{(i)}$, $et^{(i)}$, $status^{(i)}$, and $sr^{(i)}$ are the i_{th} subgoal's description, starting timestep, ending timestep, final status and the reward received when it is done respectively. $status^{(i)}$ and $sr^{(i)}$ are defined as $\Psi(s_{t=et^{(i)}}|sg^{(i)})$ and $r_{t=et^{(i)}}$ respectively.

We then define the full history when the i_{th} subgoal is done as $h^{(i)} = \left(g, o_0, \{sh^{(\tau)}\}_{\tau=1}^{i}\right)$ where g, o_0 and $sh^{(\tau)}$ are the goal description of the intended task, the initial observation and the τ_{th} subgoal's history respectively. Then, we define history abstraction as a function $\overline{h}(\cdot)$ that maps a full history $h^{(i)}$ to an abstract $\overline{h}(h^{(i)})$ that is usually shorter, only keeping critical timesteps such as the initial timestep and the timesteps where a subgoal is completed/failed.

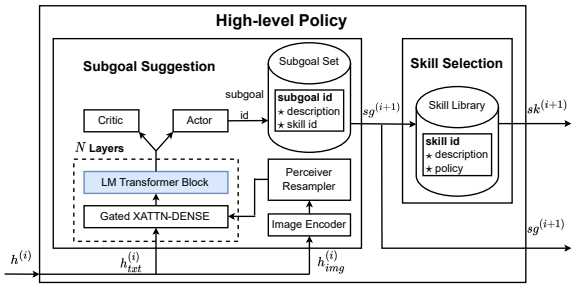
Given a goal g, a subgoal set $\mathcal{G}_l$ and a skill library, we aim to train a high-level policy $\pi(sg^{(i+1)}|h^{(i)})$ such that the expected discounted future reward $\sum_{i\geq 0} \mathbb{E}_{sg^{(i+1)}\sim\pi(\cdot|h^{(i)})}[\gamma^i sr^{(i+1)}]$ is maximized.

4 Approach

To leverage existing skills (pretrained policies for simple tasks) in tackling new tasks, we propose the VLM-HRL agent design, as shown in Figure 1a, that consists of a hierarchy of two levels of policies together with two auxiliary modules, *History Maintenance* and *Subgoal Progress Monitoring*. The high-level policy proposes the next subgoal and selects a skill for it. In contrast, the low-level policy applies the chosen skill to generate the agent's primitive actions until the subgoal is done.



(a) The agent with a hierarchy of high-level and low-level policy modules.



(b) The high-level policy module that uses a Flamingo-based Actor-Critic model to determine the next subgoal $sg^{(i+1)}$.

Figure 1: VLM-HRL Agent. Note: $h^{(i)}$ is the history up to the time step when the i_{th} sugboal is done, and $sg^{(i+1)}$ and $sk^{(i+1)}$ are the $(i+1)_{th}$ suggested subgoal and skill. The *LM Transformer Block* highlighted in blue indicates it is frozen during the training with the *FrozenAll* LLM mode. $h_{img}^{(i)}$ refers to the sequence of all visual observations in $h^{(i)}$ and $h_{txt}^{(i)}$ is a representation of $h^{(i)}$ in text sequence (more details can be found in Appendix A).

4.1 Subgoal Progress Monitoring and History Maintenance

Subgoal Progress Monitoring dispatches high-level/low-level action requests based on the current subgoal's status. It considers the subgoal as done when the step budget is exhausted or when an oracle determines its status based on the current observation and subgoal description. When a subgoal is done, it requests *History Maintenance* to update history and inform the high-level policy to generate the next subgoal. Otherwise, it instructs low-level policy to use the current skill and observation to generate a primitive action for progressing towards the subgoal.

History Maintenance, as shown Figure 5 in Appendix A, updates the current history with the summarization of the current subgoal history and sends it to high-level policy for proposing the next subgoal. In this work, for simplicity, we adopted a rule-based summarization that selects the first and the last observations of a subgoal's history. We believe that this summarization can capture essential information about a subgoal and effectively reduce redundant observations, particularly when the agent repeats ineffective primitive actions.

4.2 High-level and Low-level Policy Modules

High-level policy, as shown in Figure 1b, takes as the input the agent's current history $h^{(i)}$ and feeds it into *Subgoal Suggestion*, primarily consisting of a Flamingo model, to suggest the next subgoal id. It is then used to choose the next subgoal $sg^{(i+1)}$ and skill $sk^{(i+1)}$ from *Subgoal Set* and *Skill Library* respectively. The Flamingo model, housing a causal LLM within, facilitates the capture of the mission status: 1) Perceiver Resampler in Flamingo extracts visual semantics of the agent behavior during the execution of each subgoal; 2) Cross-Attention layer in Flamingo fuses a subgoal's description and the visual semantics of the agent's behavior to describe a subgoal execution; 3) the causal LLM combines all subgoals' execution semantics as the current mission status.

We design skills to incorporate a memory processing component that uses an LSTM layer to aggregate partial observations. Meanwhile, we have the VLM-HRL agent maintain the memory for the currently selected skill. When the high-level policy module suggests the next subgoal $sg^{(i+1)}$ and skill $sk^{(i+1)}$, the low-level policy module first clear the skill memory sm if $sg^{(i+1)}$ and $sg^{(i)}$ are different. After that, when the *Subgoal Continue* message is received, the current skill $sk^{(i+1)}$ is used to determine a primitive action and update the skill memory by plugging the current observation o_t and skill memory sm_{t-1} into the equation $a_t, sm_t = sk^{(i+1)}(o_t, sm_{t-1}, sg^{(i+1)})$.

4.3 Hierarchical Reinforcement Learning

We used *Proximal Policy Optimization* (PPO) [11] algorithm, as shown in Algorithm 1, to train the agent in a HRL framework. In each iteration, the experience collection collects N episodes instead of a fixed number of steps. The number of subgoals $Num\_SGS_j (j \in [1, N])$ proposed in one of these N episodes could be different from that in others. The experience collected from one episode, e.g., episode $j (j \in [1, N])$, contains the agent's history h_j, each high-level action, the state value after executing each high-level action, and each reward $sr^{(i)}$, where $i \in [1, Num\_SGS_j]$. During the model update, the batch of N episodes is randomly split into multiple mini-batches of M episodes. Each mini-batch is then used to calculate a loss for updating the model using Equation 1

$$loss = \frac{1}{MNS} \sum_{i=1}^{MNS} \frac{\sum_{j=1}^{M} loss_j^{(i)}}{\sum_{j=1, loss_j^{(i)} \neq 0}^{M} 1} \tag{1}$$

where $MNS = max([Num\_SGS_j]), j \in [1, M]$ is the maximum number of proposed subgoals of an episode in the mini-batch and $loss_j^{(i)}$ is the PPO loss associated with the i_{th} subgoal of the j_{th} episode if it exists in the mini-batch otherwise 0.

Algorithm 1 VLM-HRL Learning Using PPO

Require: max epochs K, the high-level policy π_θ with the parameter θ

for iteration=1,2,... **do**
  Collect experiences of N episodes using policy $\pi_{\theta_{old}}$.
  Compute advantage estimates for each subgoal and add to experiences.
  for epoch=1,2,...,K **do**
    Shuffle the batch of N experiences
    Split the batch into mini-batches
    for each mini-batch **do**
      Minimize $loss$ (Equation 1) wrt θ
      $\theta_{old} \leftarrow \theta$
    end for
  end for
end for

5 Experiment Setting

For the proposed agent design, we hypothesize that it could improve sample efficiency when compared to the baseline agent in BabyAI 1.1 [7]. We conduct experiments on the BabyAI platform [6] to test our hypothesis and study the impact of two factors on the sample efficiency, including the training mode of the LLM in the Flamingo model and the history representation.

5.1 BabyAI Platform

The BabyAI platform was designed to facilitate research on grounded language learning in a partially-observable 2-D gridworld. The platform originally comes with 19 levels, each of which represents a distribution of tasks at a certain difficulty level. The agent can navigate in the environment and move objects to complete a given task that is described in a natural-looking instruction, e.g., "put the blue ball next to the yellow box". The platform provides a utility function to determine if the instructed task is completed based on the current observation. It can play the role of the function Ψ we assumed to access for checking if a subgoal has been accomplished.

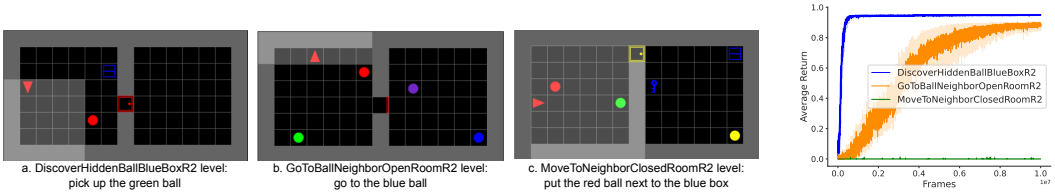
5.2 Skill Library, Subgoals and Basic Levels

Considering the agent's interaction with objects on BabyAI platform, we design 6 basic levels for the agent to acquire 6 basic skills (*GoTo*, *Pickup*, *DropNextTo*, *OpenBox*, *OpenDoor* and *PassDoor*). Each skill can accomplish a set of subgoals tied to valid objects. For example, the *OpenBox* skill targets boxes of any color. The total subgoals covered by these skills is 102. Task examples of basic levels and each skill's training configuration and performance can be found in Appendix B.

We designed three levels of varying complexity for the study, assessed by: 1) the number of objects in the environment, 2) the minimum subgoals required, and 3) the necessity of examining adjacent rooms for task completion. In Table 1, the levels—*DiscoverHiddenBallBlueBoxR2*, *GoToBallNeighborOpenRoomR2*, and *MoveToNeighborClosedRoomR2*—are categorized as low, medium, and high complexity. The relative complexities of designed levels are affirmed by baseline agent performance as illustrated in Figure 2b. Example tasks of these levels are shown in Figure 2a, with detailed descriptions in Appendix C. For each level, we assume domain knowledge access to specialize the 102 subgoals into a reduced set, as listed in Table 3 in Appendix D.

Table 1: Characteristics of three testing levels

Level Name	Complexity Degree	Number of Objects	Adjacent Room Exploration	Minimum Number of Needed Subgoals
DiscoverHiddenBallBlueBoxR2	low	3	No	2
GoToBallNeighborOpenRoomR2	medium	4	Yes	2
MoveToNeighborClosedRoomR2	high	5	Yes	4



(a) Example tasks. (b) Baseline performance.

Figure 2: Testing levels: example tasks and performance of baseline agent.

5.3 Implementation of Subgoal Suggestion

We implemented *Subgoal Suggestion* using flamingo-pytorch [12] and DistilGPT2 [13]. We configured the Flamingo model with one gated cross-attention layer for every three transformer blocks. For the policy headers, we adopted the same architecture as that used in the baseline agent.

5.4 Training Configuration for The VLM-HRL Agent

In all testing levels, the training configuration for VLM-HRL agent uses an entropy coefficient of 0.002, a PPO clipping threshold of 0.1, and a maximum number of 1 million primitive steps. While the learning rate may be selected from the list, $[1e{-}4, 3e{-}5, 1e{-}5]$. For each proposed subgoal, the agent applies the selected skill with a step budget of 16. For one training configuration, we trained the VLM-HRL agent in 5 runs of randomly picked unique seeds and reported the mean and standard error of the *Average Return* over 100 evaluation episodes. To study the impact of the knowledge of the pretrained LLM on learning efficiency, we investigated three training modes for the LLM in Flamingo during agent learning: 1) *FrozenAll* freezes the pretrained LLM (as shown in Figure 1b); 2) *RandomInit* trains a randomly initialized LLM; 3) *TuneAll* finetunes the pretrained LLM.

6 Experiment Results

6.1 Sample Efficiency: VLM-HRL Agent vs Baseline Agent

To validate the hypothesis introduced in Section 5, we investigated the VLM-HRL agent with a randomly initialized DistilGPT2 and a full history as a stepping stone. To compare with the baseline agent, we decided to report the *Average Return* of VLM-HRL agent versus the number of consumed frames. Since the algorithm 1 collects experiences of a fixed number of episodes every iteration, the logging timesteps can not be pre-set to a fixed number of frames. To mimic the reporting convention in RL literature, we decided to report the mean and standard error of *Average Return* of runs with M different seeds by using the *Average-of-Polylines* approach in Appendix E. The first row in Figure 9 in Appendix F shows that the VLM-HRL agent outperforms the baseline agent in all testing levels, indicating the effectiveness of the VLM-HRL learning.

6.2 The Impact of Training Mode of LLM and History Representation

We pick the best-performing agents in each LLM-training-mode group and compare them in Figure 3. The comparison between *FrozenAll* and *TuneAll* agents highlights that freezing the pretrained LLM does not prove beneficial for the decision-making of the VLM-HRL agent. Moreover, the TuneAll agent's underperformance in relation to the RandomInit agent indicates the role of fine-tuning in undoing the LLM pretraining, while also suggesting the limited utility of LLM pretraining for the VLM-HRL agent in long-horizon instruction-following tasks within environments featuring less natural instructions and observations, such as BabyAI. We hypothesize that the VLM-HRL agent may be able to utilize the knowledge in the pretraining LLM when working in environments with natural instructions and observations. We leave the investigation as future work.

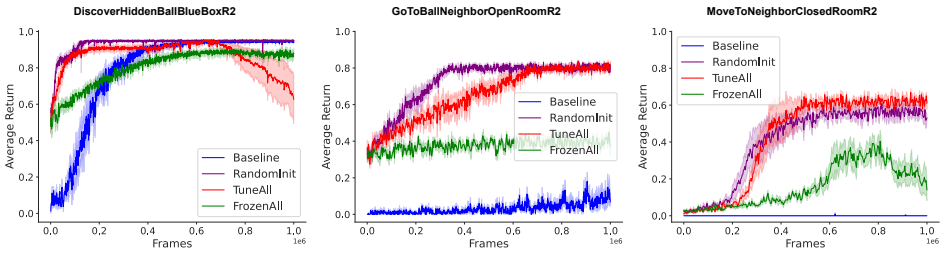


Figure 3: Average Return of the best-performing VLM-HRL agent variants in every testing level.

Using a full history will inevitably pose a challenge for the agent when facing long-horizon tasks where storing the full history is impossible or the cost (time and energy consumption) of proposing the next subgoal by processing the full history is not acceptable. This motivates us to study the impact of history representation on the learning capability of the VLM-HRL agent. We first pick the best-performing *RandomInit* agent and *TuneAll* agent in each testing level, and then train the counterpart agents with an abstract history. The poor performance of *FrozenAll* agent excludes itself from this study. As seen in Figure 4, history abstraction is beneficial for both *RandomInit* and *TuneAll* agents in terms of learning speed and learning stability respectively. We hypothesize that the advantages stem from eliminating redundant visual observations through history abstraction, thereby directing the agent's attention to crucial visual data. While the basic rule-based history abstraction has some benefits in accelerating the learning of *TuneAll* agents during early learning stages in the *GoToNeighborOpenRoomR2* and *MoveToNeighborClosedRoomR2* levels, it doesn't sustain the learning speed throughout the entire learning process. This suggests that a more sophisticated history abstraction method could be helpful for the agent to keep a fast and consistent learning pace.

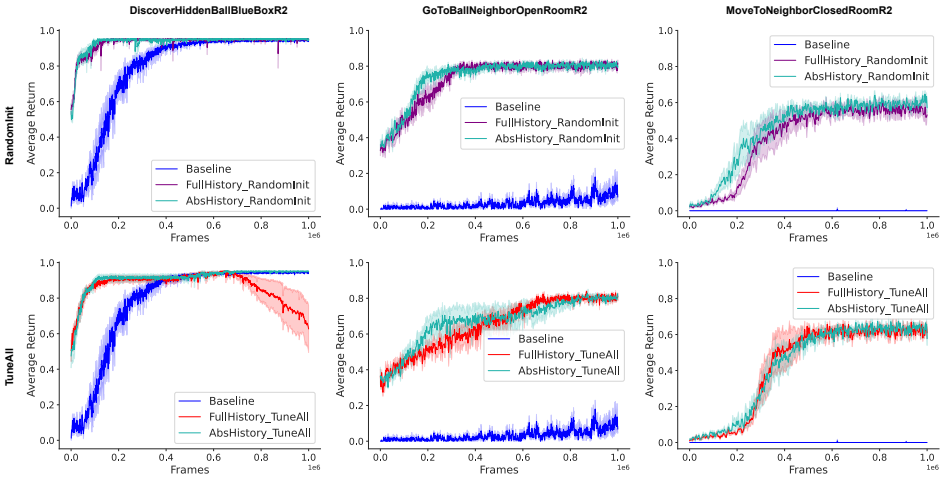


Figure 4: Performance comparison between VLM-HRL agents with full history and abstract history.

7 Related Work

7.1 Language-Informed HRL for Partially Observable Environment

Seeing the advance of representation learning of language, researchers seek to enhance reinforcement learning by incorporating natural language [14], harnessing its compositional, relational, and hierarchical qualities. Multiple works have exploited language for state representation [15, 16], reward shaping [9, 17], action generation [18], and goal representation [19].

Hierarchical Reinforcement Learning addresses long-horizon complex tasks by exploiting the hierarchical structure of tasks. [20] introduces the Hierarchical Deep Reinforcement Learning Network architecture, surpassing the vanilla DQN [21]. [22] devises a meta-learning approach to learn hierarchically structured policies. Both of them aim to jointly learn a set of low-level policies and a high-level policy that optimally selects a low-level policy at each high-level timestep. In contrast, our work assumes access to pretrained low-level policies, focusing on learning a high-level policy for proposing subgoals to complete long-horizon tasks in partially observable environments.

Addressing Partially Observable Markov Decision Processes (POMDPs) entails handling scenarios with only observations, concealing the underlying MDP state. [23] introduces the Deep Recurrent Q-Network, combining LSTM [24] with DQN to address partial observability. In contrast, [25] presents the Deep Transformer Q-Network, replacing recurrent layers with a transformer decoder. Additionally, [26] introduces the History Compression via Language Models framework, leveraging pretrained language models for history compression. All these works primarily address memory components and history maintenance without incorporating language understanding. Our work represents the agent's history by associating each subgoal (high-level language description) with corresponding visual observations using Flamingo.

7.2 Applications of LLMs and VLMs to Planning and Interactions for Robotics

Large language models have seen remarkable success in natural language processing and emerged as valuable tools in robotics research. These pretrained LLMs (and vision models) are leveraged for various applications, such as zero-shot planning [27, 28], semantic and structural exploration with intrinsic rewards [29, 30, 31], building task-specific priors from the task's metadata [32], and grounding human language corrections to robot states and environment dynamics [33]. Additionally, The optimized conversation capability of ChatGPT [34] can be utilized for the automatic generation of deployable robotic code by smoothly incorporating human feedback on code quality and safety, given task specifications and predefined behaviors [35]. SayCan [36] efficiently decomposes high-level tasks into subtasks, using affordances to connect LLMs with the physical environment. In contrast, our work assumes access to a predefined list of low-level policies and focuses on studying if the Flamingo architecture is able to learn a good high-level policy, successfully grounding the knowledge in a pretrained LLM within the agent's experience in a partially observable environment.

8 Conclusion

In this study, we explore the utilization of the architecture of the Flamingo vision-language model to enhance learning efficiency by proposing promising subgoals in HRL setting. With a predefined subgoal set and skill library, our VLM-HRL agent aims to improve sample efficiency in partially observable environments. Experimental results validate our hypothesis, demonstrating the superior performance of the *RandomInit* VLM-HRL agent over the baseline agent across all testing levels. Additionally, we investigate the impact of pretrained language model knowledge on the VLM-HRL agent's learning efficiency through three variants: *RandomInit*, *TuneAll*, and *FrozenAll*, employing different LLM training modes in Flamingo. The findings suggest that LLM pretraining offers limited benefits for the VLM-HRL agent in environments with less natural instructions and observations. Considering resource limitations and time constraints in long-horizon tasks, we propose substituting the full history with an abstract history to mitigate the storage and processing overhead of numerous visual observations. Our study evaluates the effect of history representation on the learning capability of VLM-HRL agents by comparing two groups, *Full History* and *Abstract History*. The results highlight the advantages of history abstraction, enhancing both learning efficiency and stability. Based on these observations, our future work will include: 1) examining how a pretrained LLM's knowledge influences the VLM-HRL agent's decision-making in environments with natural instructions and observations; 2) pursuing a sophisticated history abstraction with the aim of outperforming the simple one introduced in this work; 3) researching methods to allow the high-level policy to generate descriptive subgoals directly instead of proposing subgoal IDs.

Acknowledgments

This work is supported in part by NSF IIS-2154904.

References

[1] 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.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[4] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[5] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

[6] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations*, 2019.

[7] D. Y.-T. Hui, M. Chevalier-Boisvert, D. Bahdanau, and Y. Bengio. Babyai 1.1. *arXiv preprint arXiv:2007.12770*, 2020.

[8] 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.

[9] S. Mirchandani, S. Karamcheti, and D. Sadigh. Ella: Exploration through learned language abstraction. *Advances in Neural Information Processing Systems*, 34:29529–29540, 2021.

[10] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

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

[12] P. Wang and M. Ryu. Implementation of flamingo, state-of-the-art few-shot visual question answering attention net out of deepmind, in pytorch. https://github.com/lucidrains/flamingo-pytorch, 2022.

[13] V. Sanh. Distilgpt2 is an english-language model pre-trained with the supervision of the smallest version of generative pre-trained transformer 2. https://huggingface.co/distilgpt2, 2019.

[14] J. Luketina, N. Nardelli, G. Farquhar, J. Foerster, J. Andreas, E. Grefenstette, S. Whiteson, and T. Rocktäschel. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*, 2019.

[15] K. Narasimhan, R. Barzilay, and T. Jaakkola. Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 63:849–874, 2018.

[16] E. Schwartz, G. Tennenholtz, C. Tessler, and S. Mannor. Language is power: Representing states using natural language in reinforcement learning. *arXiv preprint arXiv:1910.02789*, 2019.

[17] P. Goyal, S. Niekum, and R. J. Mooney. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*, 2019.

[18] S. Yao, R. Rao, M. Hausknecht, and K. Narasimhan. Keep calm and explore: Language models for action generation in text-based games. *arXiv preprint arXiv:2010.02903*, 2020.

[19] 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.

[20] C. Tessler, S. Givony, T. Zahavy, D. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[22] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

[23] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[25] K. Esslinger, R. Platt, and C. Amato. Deep transformer q-networks for partially observable reinforcement learning. *arXiv preprint arXiv:2206.01078*, 2022.

[26] F. Paischer, T. Adler, V. Patil, A. Bitto-Nemling, M. Holzleitner, S. Lehner, H. Eghbal-Zadeh, and S. Hochreiter. History compression via language models in reinforcement learning. In *International Conference on Machine Learning*, pages 17156–17185. PMLR, 2022.

[27] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[28] 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.

[29] J. Mu, V. Zhong, R. Raileanu, M. Jiang, N. Goodman, T. Rocktäschel, and E. Grefenstette. Improving intrinsic exploration with language abstractions. *arXiv preprint arXiv:2202.08938*, 2022.

[30] A. Tam, N. Rabinowitz, A. Lampinen, N. A. Roy, S. Chan, D. Strouse, J. Wang, A. Banino, and F. Hill. Semantic exploration from language abstractions and pretrained representations. *Advances in Neural Information Processing Systems*, 35:25377–25389, 2022.

[31] 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.

[32] K. Choi, C. Cundy, S. Srivastava, and S. Ermon. Lmpriors: Pre-trained language models as task-specific priors. *arXiv preprint arXiv:2210.12530*, 2022.

[33] P. Sharma, B. Sundaralingam, V. Blukis, C. Paxton, T. Hermans, A. Torralba, J. Andreas, and D. Fox. Correcting robot plans with natural language feedback. *arXiv preprint arXiv:2204.05186*, 2022.

[34] OpenAI. Chatgpt. https://openai.com/blog/chatgpt, 2023.

[35] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor. Chatgpt for robotics: Design principles and model abilities. *2023*, 2023.

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

Appendix A History Maintenance and Sequence Formatting

History Maintenance initializes the history $h^{(0)} = (g, o_0)$ with the goal g and the initial observation o_0 when a mission starts. The inside of the History Maintenance is shown in Figure 5.

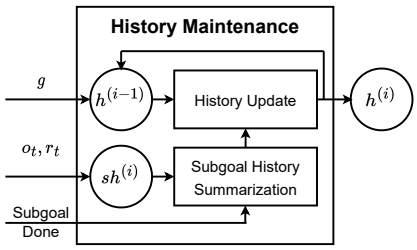


Figure 5: *History Maintenance* summarizes the history of the current subgoal, $sh^{(i)}$, when the current subgoal $sg^{(i)}$ is done, and then applies the summarized subgoal history to update the agent's history from $h^{(i-1)}$ to $h^{(i)}$, which will be used for generating the next subgoal $sg^{(i+1)}$.

Since the Flamingo model (the high-level policy) takes as input a textual part and a visual part, we first construct $h^{(i)}_{txt}$ and $h^{(i)}_{img}$ from the history $h^{(i)}$ and then feed them to the high-level policy. $h^{(i)}_{img}$ is the sequence of visual observations, encompassing all observations from $h^{(i)}$; $h^{(i)}_{txt}$ is a text sequence that serves as a representation of $h^{(i)}$ and follows the format as *"*$g|image|[Start]...sg^{(i)}|image...image|[status^{(i)}]$*"*, where

- g is the language description of the goal g.
- $image$ represents an observation. The first $image$ string refers to the initial observation o_0.
- [,], | and $Start$ are string literals.
- $sg^{(i)}$ is the language description of the i_{th} subgoal.
- $status^{(i)}$ is the language description of the i_{th} subgoal's status.

Appendix B Basic Levels and Skill Library

Figure 6 shows an example task for each of the 6 basic levels. For each level, there are 5 objects randomly positioned in the agent's starting room and there are two rooms in the environment. All levels except *PassDoorLocalR2* can be completed inside the agent's starting room. The *DropNextToLocalR2* level assumes the agent carries the object to drop at the beginning of a task.

To learn the skill for each basic level, we used PPO to train a policy by using the bag-of-word encoding for observations and the model architecture from BabyAI 1.1 [7], which is shown in Figure 7a. The bag-of-wording encoding represents the partial observation of 7x7 grid into a 7x7x3 tensor where each cell on the 7x7 grid is converted into a 1-D tensor, (object type, object color, door state). We used the same training configuration, as shown in Table 2 to learn a skill for each basic level. For each basic level, we trained the agent to learn three policies, with each policy undergoing training using a unique random seed, and then select the best policy as the skill representative and put it into the skill library. The training performance for each skill is shown in Figure 7b.

Appendix C Three Testing Levels

Three designed testing levels of different complexity degrees as listed below:

- *DiscoverHiddenBallBlueBoxR2*: the environment features a closed door of any color, a blue box, and red and green balls. The target ball, either red or green, is concealed within the box. The goal is to locate and collect the target ball.

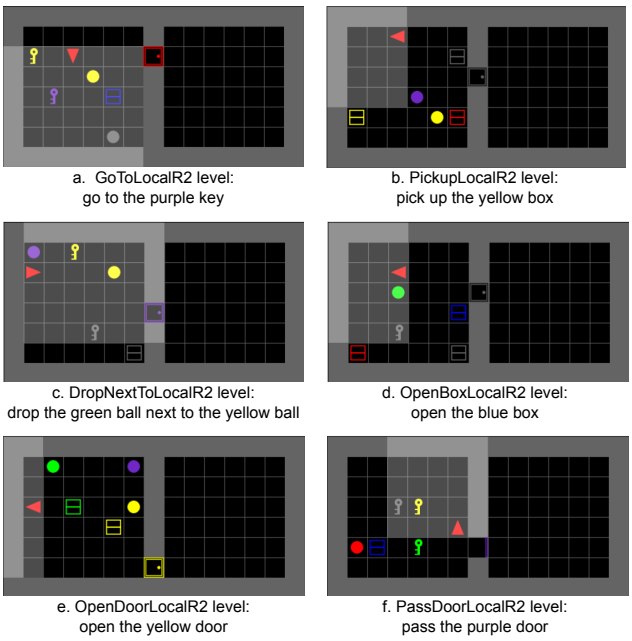
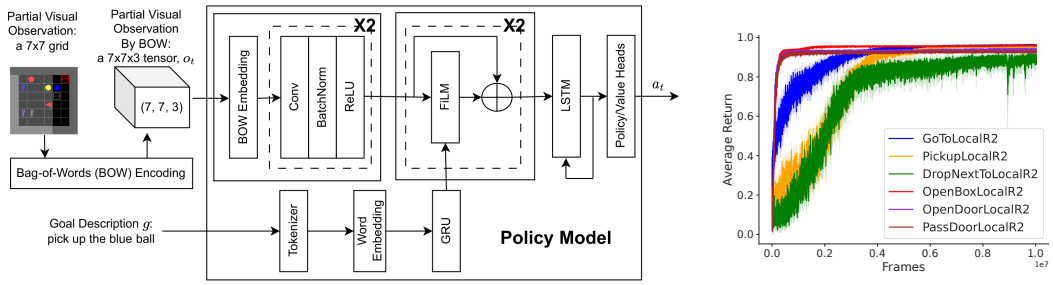


Figure 6: Example tasks of the 6 basic levels.



(a) The model architecture of skills.

(b) Skills' Performance.

Figure 7: Model architecture and performance of skills. Note: Bag-of-Words encoding converts each cell of the raw partial observation (a 7x7 grid) to a triple, (object type, object color, door state).

- *GoToBallNeighborOpenRoomR2*: the environment contains one door and four uniquely colored balls. The door is open and has a color of either red or green. The balls come in red, green, blue, and purple colors. Two of the four balls are in the agent's starting room, while the remaining two are situated in the adjacent room.
- *MoveToNeighborClosedRoomR2*: the environment includes a closed yellow door, a red ball, a green ball, a blue box, and a key of any color. One of the balls is the target object to be moved. The key and the non-target ball serve as distractors. The balls are positioned in the agent's starting room, while the key and the box are placed in the adjacent room. The goal is to locate the target ball and move it next to the blue box.

Appendix D Specialized Subgoal Set for Each Testing Level

For each testing level, its specialized subgoal set, as listed in Table 3, is crafted by selecting subgoals from the entire subgoal set (102 subgoals) after answering the following two questions:

- what objects and doors could possibly exist?
- how could these objects and doors be related to tasks in the level?

Table 2: Training configuration for learning a skill for each basic level

Model Architecture	arch: 'bow_endpool_res' instr_arch:'gru' image_dim:128 instr_dim:128 memory_dim:128
Experience Collection	procs: 16 frames_per_proc: 40 recurrence: 20 batch_size: 640 max_frames: 1e7
Model Update	lr: 1e-4 entropy_coef: 0.01 value_loss_coef:0.5 max_grad_norm:0.5 clip_eps:0.2

Table 3: Domain knowledge-based subgoal sets for 3 testing levels

Level Name	Designed Subgoal Set
DiscoverHiddenBallBlueBoxR2	[1] open the blue box [2] pick up the green ball [3] pick up the red ball
GoToBallNeighborOpenRoomR2	[1] pass the red door [2] pass the green door [3] go to the red ball [4] go to the green ball [5] go to the blue ball [6] go to the purple ball
MoveToNeighborClosedRoomR2	[1] open the yellow door [2] pass the yellow door [3] pick up the green ball [4] pick up the red ball [5] drop next to the blue box

Appendix E *Average-of-Polylines* for Performance Comparison

Average-of-Polylines approach intends to approximate the mean and standard error of an interested performance using runs of the same experiment with M different seeds, where the logging points (e.g., consumed steps in RL) of the performance during each run of the experiment are not prefixed.

- For the logged data points of each run i ($i = 1, 2, ..., M$), we fit them to a polyline by connecting two consecutive data points whose x values are next to each other.
- Find out the smallest number of data points across all runs and use it as the number of samples, N, to draw from the generated polylines above.
- Evenly draw from a given range $[x_{min}, x_{max}]$, e.g. $[0, 1e6]$, a list of N values, $x = [x_1, ..., x_N]$, where $x_1 = x_{min}$ and $x_N = x_{max}$.
- For each polyline i ($i = 1, 2, ..., M$), sample from it N points $[(x_1, y_1^{(i)}), ..., (x_N, y_N^{(i)})]$ using x.
- Plot the representative curve using the mean and standard error of each aligned group of points, $((x_k, y_k^{(1)}), (x_k, y_k^{(2)}), ..., (x_k, y_k^{(M)}))$ where $k = 1, ..., N$.

Figure 8 shows an example of applying the *Average-of-Polylines* approach to report the mean and standard error of *Average Return* of runs with 5 different seeds by the VLM-HRL agent that uses a

randomly-initialized language model, a full history and a fixed learning rate of $1e-4$ to solve tasks in the level DiscoverHiddenBallBlueBoxR2. The light blue curve represents the mean and standard error of *Average Return* of runs of five different seeds, where red, orange, pink, green, and purple curves correspond to the original data of the five runs. As a reference, the baseline's performance is plotted in dark blue.

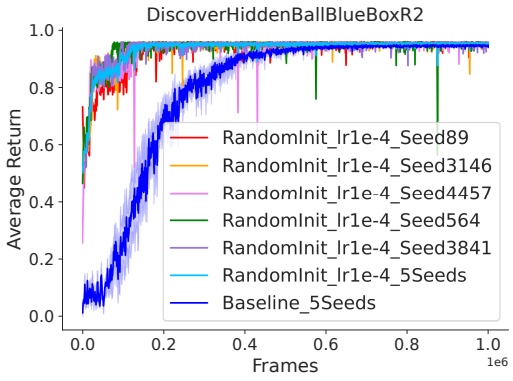


Figure 8: Merge *Average Return* of VLM-HRL agents trained in five runs of different seeds.

Appendix F VLM-HRL Agents Trained Using Different LLM Training Modes and Learning Rates

For each training mode of the LLM in Flamingo, we report the *Average Return* of agents in Figure 9:

- *FrozenAll* agents are not comparable to that of the other two groups. When compared to the baseline agent in *DiscoverHiddenBallBlueBoxR2*, all three *FrozenAll* agents show a good learning speed at start. But two of the three, which were trained with learning rates $3e-5$ and $1e-5$, have unstable learning processes, showing performances collapse later. The third agent that was trained with the learning rate $1e-5$ has a stable learning process but falls behind the baseline agent after the early stage. In *MoveToNeighborClosedRoomR2*, the *FrozenAll* agent trained with a learning rate of $1e-4$ learned a better policy than the baseline agent but its performance tends to deteriorate during the later stage of the learning process. For the rest two *FrozenAll* agents in *MoveToNeighborClosedRoomR2* and all three *FrozenAll* agents in *GoToNeighborOpenRoomR2*, they show a good starting point when compared to the baseline agent, but were not able to learn more under the given learning budget of one million frames. The results collectively indicate that freezing the pretrained LLM significantly restricts the agent's capability for solving downstream tasks.
- *TuneAll* agents have an unstable learning process in *DiscoverHiddenBallBlueBoxR2* and fail to compete with the baseline agent. But *TuneAll* agents are more capable than the baseline in learning to solve tasks of the other two testing levels, *GoToBallNeighborOpenRoomR2* and *MoveToNeighborClosedRoomR2*.
- *RandomInit* agents can solve tasks of all three testing levels. And there is at least one variant *RandomInit* agent that has a better learning speed than the baseline agent.

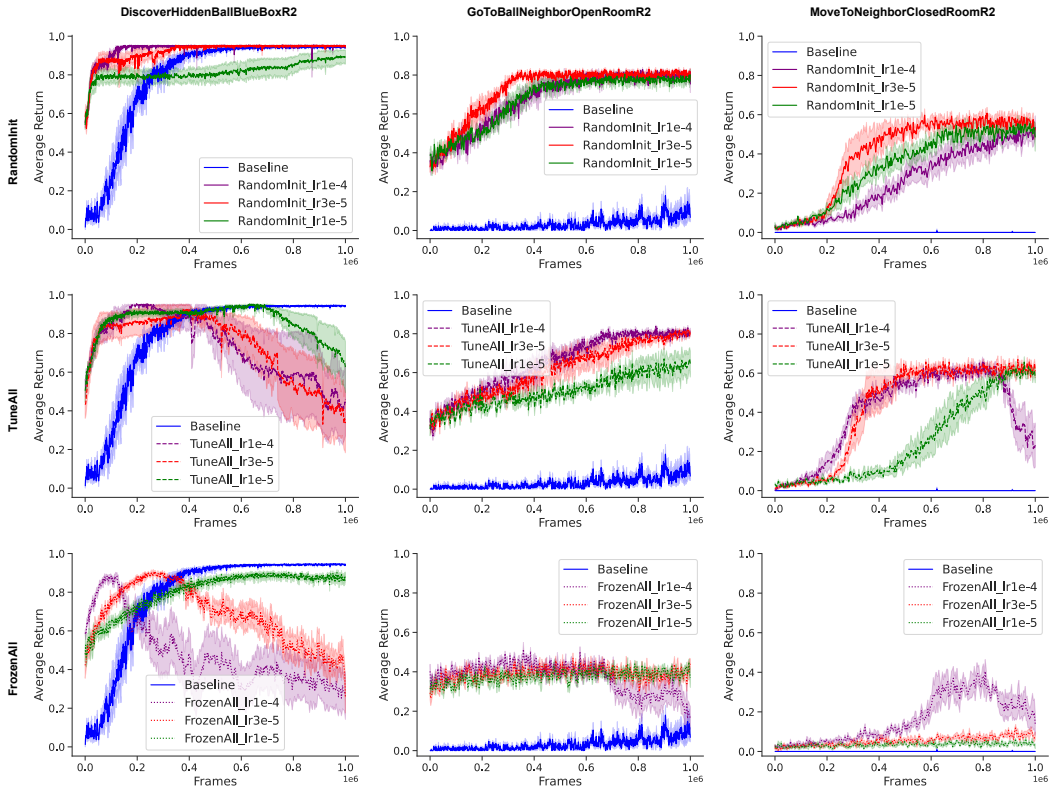


Figure 9: Average Return of VLM-HRL agents trained with different LLM training mode and learning rate in every testing level.