Hypothesis, Verification, and Induction: Grounding Large Language Models with Self-Driven Skill Learning

Shaohui Peng[1], Xing Hu[2, 4], Qi Yi[2, 3], Rui Zhang[2], Jiaming Guo[2], Di Huang[2], Zikang Tian[2, 5], Ruizhi Chen[1], Zidong Du[2, 4], Qi Guo[2], Yunji Chen[2, 5], Ling Li[1, 5,*]

[1] Intelligent Software Research Center, Institute of Software, CAS, Beijing, China
[2] SKL of Processors, Institute of Computing Technology, CAS, Beijing, China
[3] University of Science and Technology of China, USTC, Hefei, China
[4] Shanghai Innovation Center for Processor Technologies, SHIC, Shanghai, China
[5] University of Chinese Academy of Sciences, UCAS, Beijing, China
pengshaohui@iscas.ac.cn

Abstract

Large language models (LLMs) show their powerful automatic reasoning and planning capability with a wealth of semantic knowledge about the human world. However, the grounding problem still hinders the applications of LLMs in the real-world environment. Existing studies try to fine-tune the LLM or utilize pre-defined behavior APIs to bridge the LLMs and the environment, which not only costs huge human efforts to customize for every single task but also weakens the generality strengths of LLMs. To autonomously ground the LLM onto the environment, we proposed the Hypothesis, Verification, and Induction (HYVIN) framework to automatically and progressively ground the LLM with self-driven skill learning. HYVIN first employs the LLM to propose the hypothesis of sub-goals to achieve tasks and then verify the feasibility of the hypothesis via interacting with the underlying environment. Once verified, HYVIN can then learn generalized skills with the guidance of these successfully grounded subgoals. These skills can be further utilized to accomplish more complex tasks that fail to pass the verification phase. Verified in the famous instruction following task set, BabyAI, HYVIN achieves comparable performance in the most challenging tasks compared with imitation learning methods that cost millions of demonstrations, proving the effectiveness of learned skills and showing the feasibility and efficiency of our framework.

Introduction

Large language models (LLMs) have shown their powerful capability in automatic reasoning and planning with a wealth of semantic knowledge about the human world (Wei et al. 2022, 2021; OpenAI 2023; Kojima et al. 2022). However, there still remains a large gap in adopting LLMs to automatically solve problems in specific environments. This is because of the misalignment between the LLM's semantic planning and the grounded-specific implementation, which is also known as the grounding problem (Ichter et al. 2022; Driess et al. 2023). Solving this problem can unlock the LLMs' capacity of understanding and affecting the real world, which is a solid step towards real-world applications of artificial intelligence.

To address the grounding problem, existing studies try to fine-tune the LLM to predict feasible actions (Carta et al. 2023; Li et al. 2022; Wang et al. 2022) or utilize a set of behavior APIs (i.e. low-level skills) that serve as a bridge between the LLMs and the environment (Ichter et al. 2022; Raman et al. 2022; Liang et al. 2022). On the one hand, fine-tuning LLMs is of low sample efficiency and may also damage the reasoning ability of LLMs. On the other hand, existing methods relying on behavior APIs often assume the APIs are pre-defined by the environment (Liang et al. 2022) or pre-trained using expert demonstrations (Ichter et al. 2022), which not only costs huge human efforts to customize for every single task but also weakens the generality strengths of LLMs. Therefore, how to *autonomously* ground the LLM onto the environment still remains an open problem and is the key challenge of LLM-based agents.

It is challenging to achieve the goal of autonomous grounding that maps the LLM's semantic plan to practical implementation, because of the following reasons: 1) The prerequisite of grounding, obtaining successful experiences, is difficult because of the sparse rewards in the physical world. 2) Even obtaining rare success experiences, the grounding is usually closely related to specific tasks without a shared API library, therefore is of low generality and invaluable for general tasks. To address these issues, we produce intrinsic rewards based on LLM-generated subgoals and their check functions, which increase successful experiences by alleviating the sparse reward issue. We then propose the language-aligned general skill learning methodology by forcing each skill to achieve a group of goals with similar semantic descriptions. These skills show good generality in solving other or even more complex tasks.

In summary, we propose a Hypothesis, Verification, and Induction (HYVIN) framework that intimately combines the LLM and the reinforcement learning process within the following key stages: 1) Hypothesis: the LLM not only acts as the planner by decomposing tasks into small subgoals but also provides the check functions so that RL agents can evaluate whether they can complete these subgoals. Such intrin-

*Corresponding author.

sic rewards from the LLM significantly alleviate the sparse reward issue. 2) Verification: with the subgoals and corresponding check functions, RL agents learn the policies of the subgoals based on intrinsic rewards and finally are verified through whether the tasks are accomplished. 3) Induction: RL agents cluster verified subgoals through semantic similarity and learn generalized skill policy upon them. With these general skills, the LLM can generate solutions for unseen or even more complex tasks through minimal and efficient interaction.

We validate the self-driven grounding framework in instruction following tasks, which is common and reasonable for LLM-based agents because of textural task instruction. Verified in BabyAI, a grid world platform to study language-grounded tasks, our automatic grounding framework achieves comparable performance in the most difficult tasks compared with imitation learning methods that cost millions of demonstrations. The experiment results not only proves the effectiveness of learned skills but also show the feasibility and efficiency of our framework.

Related Work

LLM-assisted Agents LLMs show their great power in automatic reasoning and planning with a wealth of semantic knowledge about the human world. Therefore, it is promising to involve LLMs in developing intelligent agents. The key challenge in LLM-assisted agents is how to ground the LLM's knowledge (in linguistic form) to the tasks at hand. Regarding this challenge, there are two mainstream methods: (1) utilizing a set of behaviour APIs with detailed linguistic annotations. Such APIs can be pre-defined by the environment (Liang et al. 2022) or pre-trained using expert demonstrations (Ichter et al. 2022; Huang et al. 2022; Yuan et al. 2023). For example, Code as Policies (Liang et al. 2022) uses the LLM to generate executable codes for accomplishing instructions which can invoke behaviour APIs under certain conditions. SayCan (Ichter et al. 2022) invites humans to rate the success of given demonstrations, which are utilized to train API policies and then derive an affordance function. Voyager (Wang et al. 2023) stores and retrieves executable code, which calls pre-implemented basic APIs, to handle complex scenarios. To better interact with the environment, some approaches (Huang et al. 2022; Raman et al. 2022) introduce the environment feedback to regenerate new plans, which can be seen as another kind of trial-and-error learning. Although these methods have made impressive progress by utilizing APIs, their applications are also limited by the behaviour APIs in that the agent can only accomplish tasks that can be solved by arranging these basic APIs. (2) fine-tuning the LLMs. The LLM can be fine-tuned to predict the agent's feasible action given the state descriptions. Such fine-tuning can be performed using expert demonstrations (Wang et al. 2022; Li et al. 2022) or online RL (Carta et al. 2023). However, fine-tuning a model as large as the LLM is quite time expensive and requires much training data.

Instruction Following In instruction following, an agent is given an instruction and the goal is to accomplish the task described by the instruction. Such a paradigm makes the agent able to assist human beings by following human instructions, which has wide real-world applications. The works for instruction following can be divided into three categories: (1) Semantic-parsing methods (Artzi and Zettlemoyer 2013; Misra et al. 2016), which directly parses the instruction into the agent's actions via lexical analysis and other pre-defined rules. These methods require great human efforts to design proper rules, and can not generalize to complex environments. (2) Learning-based methods, which directly train a language-conditioned policy to accomplish instructions (Peng et al. 2023). Many prior works require expert demonstrations in their training loops. For example, expert demonstrations are often used in policy imitation learning (Lynch and Sermanet 2021; Chaplot et al. 2018), hindsight instruction relabelling (Röder, Eppe, and Wermter 2022; Chen, Gupta, and Marino 2021), and learning the language-conditioned reward function (Bahdanau et al. 2019). Some works try to sidestep the need for expert demonstrations (Ranzato et al. 2021; Huang et al. 2023), but at the cost of much lower sample efficiency. All learning-based approaches are typically trained using hard-coded instruction templates, which can not provide diverse, ambiguous and long-term planning instructions as humans. Therefore, they can only deal with simple and low-level instructions such as pick-and-place tasks. (3) LLM-based methods, which use LLMs to assist the understanding and planning of instructions (Ichter et al. 2022; Liang et al. 2022; Raman et al. 2022). See the last paragraph for more details.

Preliminaries

Problem Formulation

We consider adopting an LLM-based agent to solve instruction following (IF) tasks.Each instruction $I \in T$ describes a task coarsely in the environment. Given the instruction, only when the agent accomplishes the task using primitive action set A can receive a positive reward from the environment. For example, in BabyAI, which is a famous instruction following task set, instructions like "Open the green door" or "Put the red box next to the blue ball" specify some macroscopic object manipulation tasks, while the agent needs to accomplish them in a grid world using primitive actions like "turn right", "move forward", "pick" and so on.

An LLM-based agent takes instructions I and environment observation o as input and outputs actions to accomplish tasks. As shown in Figure 1, the general framework of LLM-based agents contains a high-level *planner* and some low-level *skills* that can unitize the semantic knowledge in the LLM to accomplish instruction following tasks. Given the coarse instruction, the *planner* (often LLMs) will decompose it into a sub-instruction sequence or generate a program to solve it. At the same time, the low-level skills consist of pre-trained policies or pre-implemented scripts to execute the plan or program. Researchers usually assume the environment provides textual descriptions of state and task-related feedback to adapt to the LLM setting. With the immediate translation from the semantic output to execution in the environment through low-level APIs, the high-level

planner can get adequate feedback and iteratively refine its plan to accomplish tasks.

Current methods leverage semantic knowledge in the high-level planner to reason and decompose the coarse instruction, but meanwhile, bypass the grounding problem by translating the semantic plan into implementation through pre-defined low-level APIs. In order to maximize the use of priors in LLMs to reduce human effort, our framework aims to *automatically learn generalized skills in the environment to build low-level APIs to solve the grounding problem*.Besides, in this paper, we assume the LLM could call basic perception functions $P = \{p1, p2, ...\}$, like "GetObservedObjects()", to get environment status instead of predesigned textual observation and feedback mechanisms.

Challenges

The essential problem of building an LLM-based agent is to ground semantic knowledge of LLM in the environment. To automatically solve the grounding problem, there are two main challenges:

- How to obtain successful grounding experiences from scratch? Without pre-defined low-level APIs, the agent cannot interact with the environment to attempt the semantic plan directly and efficiently explore the reward of task accomplishment. To address the challenge, we make the LLM hypothesize the plan for each task and generate corresponding checks for each sub-step. Based on these intrinsic rewards provided by checks, we can quickly train small policies to execute and verify the plan, then collect successful trajectories as grounding experiences. Although such experience obtained through quick attempts may belong to simple tasks, we can also leverage them to enhance the grounding ability of the agent to accomplish more complex and long-term tasks progressively.
- How to efficiently train generalized low-level behavior APIs under the guidance of experience? The subgoals in successful experience are proposed by the LLM based on specific instruction, and cannot be applied to new scenarios as general behavior APIs. Inspired by inductive ideas in mathematics, we introduce a mechanism to group subgoals with similar semantics together, then train skill policies that can achieve a group of subgoals as generalized behavior APIs. Unlike standard online reinforcement learning, which suffers from data efficiency issues, we make skills training efficient through dense rewards provided by checks, and initial state restoration by the successful trajectories.

Method

In this section, we will first give an overview of our proposed HYVIN framework, which can automatically and progressively ground LLM in the environment.

Overview

As shown in Figure 2, HYVIN can be divided into four phases.

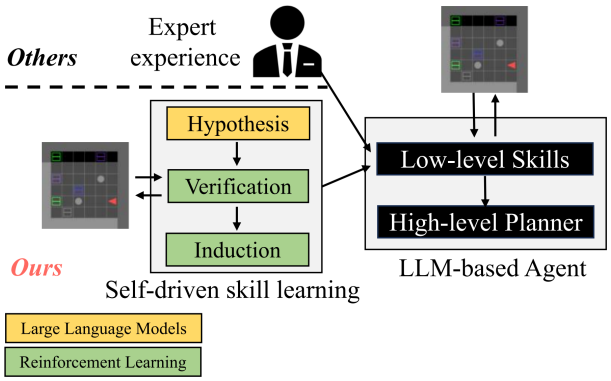

Figure 1: A general framework of LLM-based agents

- **Hypothesis**: For each instruction, LLM tries to decompose it into subgoals and generate check functions for each subgoal.
- **Verification**: Based on the reward provided by the check function, we train separate policies for each subgoal within limited steps until the task is accomplished to verify the feasibility of the hypothesis of LLM.
- **Induction**: We group the subgoals in successful hypotheses with similar semantics to train generalized skills reinforcement learning.
- **Deduction**: Based on learned skills as low-level actors, we use LLM as a few-shot high-level planner to generate programs to solve unseen and more complex tasks.

Hypothesis

The hypothesis phase aims to solve tasks separately regardless of generality to collect grounding experience. Considering the gap between the semantic knowledge in LLM and the environment, the hypothesis phase decomposes the task into several subgoals rather than directly giving the solution, and leaves correctness verification to the next process. As shown in Figure2(a), the hypothesis can be formed as $I \xrightarrow{Prompt} G, F$. We use LLM as the zero-shot planner, which takes an instruction I and necessary decomposition prompt as input, then outputs a subgoal sequence $G = \{g_1, g_2, ...\}$ and corresponding check functions $F = \{f_1, f_2, ...\}$. The subgoal g_i is a small instruction labeled with the explicit mark "Goal X" to facilitate further processing. Each check function $f_i : S \to \{0, 1\}$ is a program that checks the achievement of corresponding sub-goal g_i via invoking perception functions provided by the environment. To make the LLM output as we want, except for the instruction I, we add role definition, perception APIs descriptions, and explanation of the task space to the decomposition prompts.

Verification

After getting the subgoals and check functions, we need to verify their feasibility in the environment to collect successful grounding experiences.The feasibility of decomposition is verified by the consistency between achieved signals generated by check functions and the tasks accomplished sig-

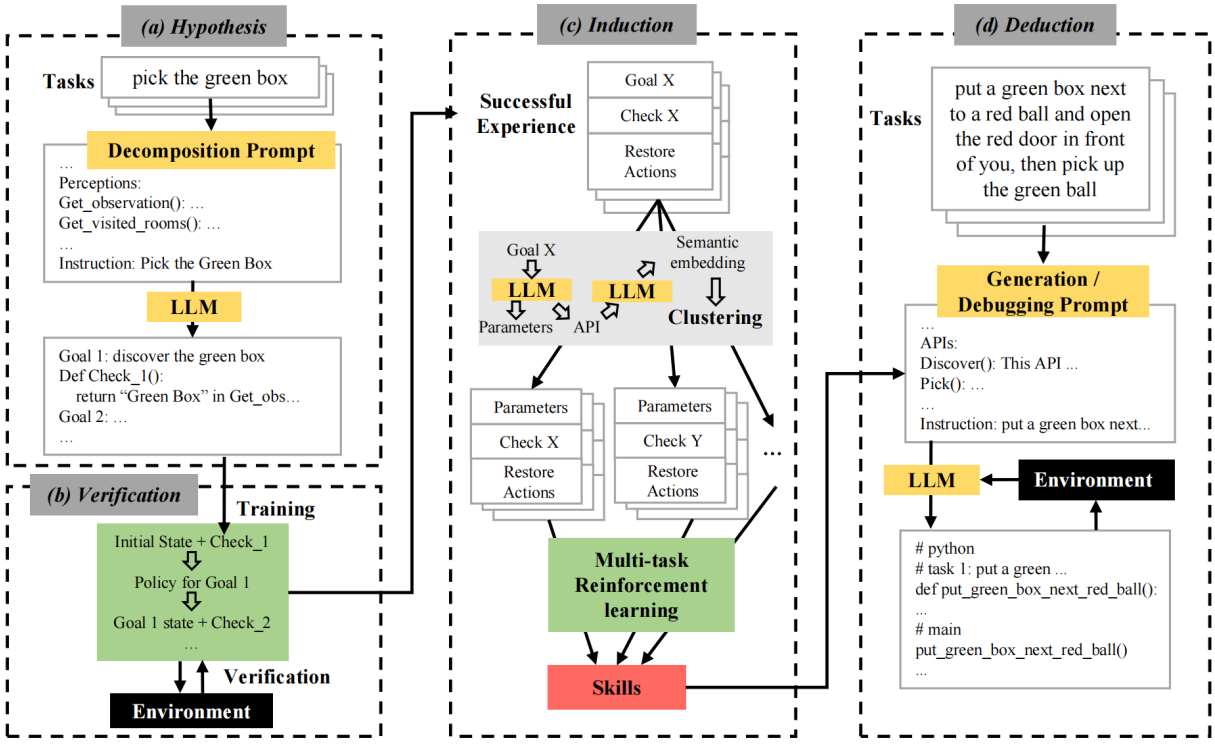

Figure 2: Overview of Hypothesis, Verification, and Induction (HYVIN) framework. (a) Hypothesis: try to decompose tasks; (b) Verification: leverage efficient interaction with the environment to verify the results of hypothesis; (c) Induction: group successful experience to train generalized skills; (d) Deduction: build few-shot planner to solve tasks using acquired skills.

nal. Specifically, as shown in Figure 2(b), we train independent policies for each subgoal based on the bool reward provided by its check function. Once the subgoal is achieved, we stop the training and save the action sequence. The saved action sequence can be used as the restoring mechanism to prepare the initial state for further skill training. Until all sub-goals are achieved and the task is also accomplished, we can verify the decomposition is successful. Meanwhile, the grounding experiences (including subgoal descriptions, check functions, and restore action sequences) are collected for skill learning in the future phase. Considering some complex and long-term tasks cannot be solved by direct decomposition, the above mechanism is only suitable for solving simple tasks within an acceptable environment interaction steps to collect experience. We set the maximum number of verification steps T_{verify} as a threshold to distinguish intractable complex tasks for the next stage to solve.

Induction

After collecting successful grounding experiences, the induction phase aims to discover and learn generalized skills from separate grounding trajectories of different instructions so that we can reuse them in more unseen and complex tasks.

Discovery As described above, we have collected task-solving experience through efficient hypothesis and verification, including subgoal descriptions, corresponding check functions, and start state restore action sequences. However, such successful experiences can only be used on specific instructions that are easy to decompose for LLM. To make the LLM-based agent able to solve more unseen and complex tasks, we must further abstract and learn skills to build the generalized low-level actor.To this end, we cluster the collected subgoals according to their semantics to ensure a certain generalized skill can accomplish a category of subgoals as shown in Figure 2(c). Specifically, we first use LLM to translate each subgoal description g_i into API description $g_{i,api}$ and parameter $g_{i,param}$. For example, the subgoal "discover the green box" is translated into the API "discover" and parameter "green box". Then we use the k-means algorithm to conduct unsupervised clustering based on the semantic distance between subgoal descriptions computed by the following cosine similarity:

$$C(g_{i,api}, g_{j,api}) = \frac{emb(g_{i,api}) \cdot emb(g_{j,api})}{||emb(g_{i,api})|| \cdot ||emb(g_{j,api})||}, \quad (1)$$

where $g_{i,api}$ is the API description and $emb(\cdot)$ is the embedding functions of LLM.

Training We have divided subgoals with similar semantics into different categories through the clustering process.Then we build reinforcement learning (RL) environments to train skills that can achieve a cluster of subgoals separately. Unlike common RL environments, the skill training environment is like a multitask learning scenario. Each subgoal of a cluster can be seen as a single task, the subgoal

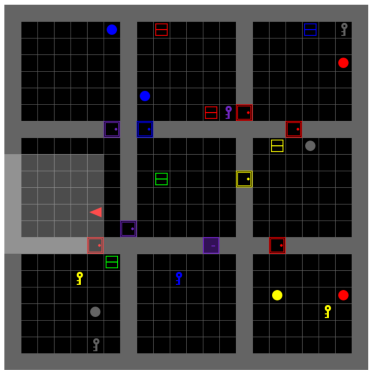

Figure 3: An example of BossLevel task in BabyAI

parameter $g_{i,param}$ is the task description, the reward is provided by the corresponding check function f_i, and the initial state is set by the saved restore action sequence. Trained in such a multiple tasks environment consists of subgoals that have the same semantics, the skill is supposed to have generalization in tasks with similar scenarios. Besides, subgoals belonging to the same cluster are divided into training and verification sets to monitor the generalization ability of the trained skill to prevent overfitting.

Deduction

Through the above process, we have overcome the challenges of obtaining successful grounding experience from scratch and training generalized skills efficiently. In other words, we have autonomously built the low-level skills for the LLM-based agent without human effort. To apply automatically learned skills to solve unseen and complex instructions following tasks, we next introduce the high-level planner in this section. As shown in Figure 2(d), we use LLM as a few-shot planner to generate programs to accomplish tasks. The process can be divided into a program generation phase and a debugging phase.

Generation The generation prompt for LLM contains the role definition and API descriptions (including skills and perception functions). Besides, considering the complexity of the task, the generation prompt follows a few-shot in-context learning paradigm. We include some skill API descriptions and a hand-written example that leverages the learned skills to solve a complex instruction-following task.

Debugging To better solve complex tasks, we also designed an interaction debugging process in our high-level planner. Besides the task instruction and the generated program to be modified, the debugging prompt also includes the error message and some general debugging suggestions to fix the possible bugs. Benefiting from the feasibility and robustness of adaptive learned skills, the basic error reporting mechanism based on illegal action detection can effectively improve the accuracy of the generated programs, which greatly reduces human effort.

Task	Method	Type	Demos	Success rates
GoTo	Original	IL	10K	99.8
	LID-Text	IL	10K	99.5
	ChatGPT	LLM	0	44
	HYVIN	LLM	0	99.9
	HYVIN-action	LLM	0	55.1
	HYVIN-no-skills	LLM	0	82.1
Pickup	Original	IL	10K	99.8
	LID-Text	IL	10K	99.8
	ChatGPT	LLM	0	0
	HYVIN	LLM	0	92.9
	HYVIN-action	LLM	0	47.6
	HYVIN-no-skills	LLM	0	73.8
PutNext	Original	IL	10K	97.7
	LID-Text	IL	10K	99.9
	ChatGPT	LLM	0	0
	HYVIN	LLM	0	91.9
	HYVIN-action	LLM	0	0
	HYVIN-no-skills	LLM	0	85.4
Open	Original	IL	1M	100
	ChatGPT	LLM	0	0
	HYVIN	LLM	0	92.4
	HYVIN-action	LLM	0	0
	HYVIN-no-skills	LLM	0	62.5
Synth	Original	IL	1M	87.7
	LISA	IL	100K	61.2
	ChatGPT	LLM	0	0
	HYVIN	LLM	0	78.9
	HYVIN-action	LLM	0	0
	HYVIN-no-skills	LLM	0	13.5
Boss	Original	IL	1M	77
	LISA	IL	100K	69.8
	ChatGPT	LLM	0	0
	HYVIN	LLM	0	75.9
	HYVIN-action	LLM	0	0
	HYVIN-no-skills	LLM	0	8.6
	HYVIN-GPT4	LLM	0	85.9

Table 1: Overall results. "IL" means "Imitation Learning", "LLM" means "LLM-based agent".

Results

Experiment Setting

Environment To evaluate the efficiency and effectiveness of our proposed framework that automatically discovers, learns, and applies skills, we test HYVIN on the BabyAI environment (Chevalier-Boisvert et al. 2019). BabyAI is a grid world environment for instruction following. Given the language instruction and a $7 \times 7 \times 3$ partial and local view, the agent must learn to accomplish various tasks of arbitrary difficulty levels. In this paper, we choose the following six levels of instruction with different types and difficulties (more details of the environment can be found in the Appendix): **GoToLocal**: Go to an object inside a single room. **PickupLocal**: Pick up an object inside a single room. **PutNextLocal**: Pick up an object and put it next to another object inside a single room. **Open**: Open a door in a 3×3 room maze, the door may in another room. **SynthSeq**: Union of all instructions from PutNext, Open, Goto, and PickUp and may with multiple commands. **BossLevel**: The hardest task of BabyAI as shown in Figure 3. The command can be any

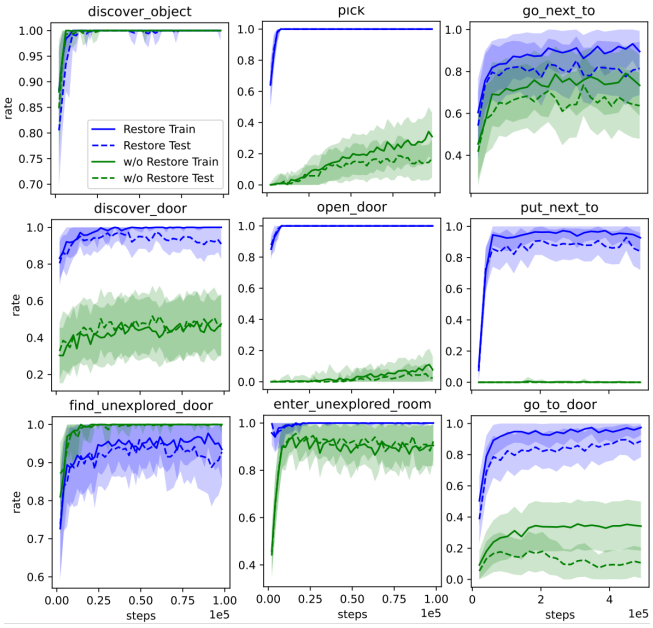

Figure 4: Training curves of skills. The blue curve represents skill learning through restoring the initial state, while the green curve doesn't. The dotted curve represents test performance in the held-out verification set.

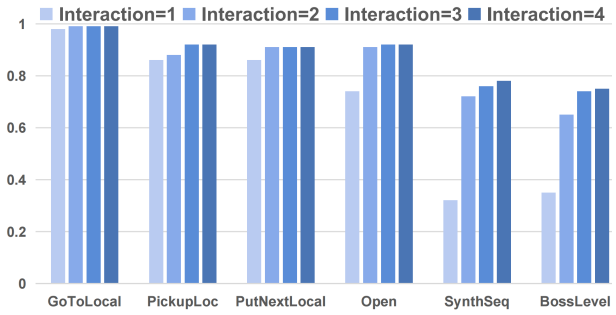

Figure 5: Successful rates of different interaction times

sentence drawn from the Baby Language grammar.

Baselines We verify the effectiveness of HYVIN by comparing it with several baselines, including Imitation Learning methods relying on expert demonstrations and a variant of our LLM-based agent: **ChatGPT** takes textual observation and obtains actions through dialog (following settings in GLAM (Carta et al. 2023)). **Original** is the baseline from the original BabyAI paper, which trained the GRU + CONV model with imitation learning using one million demonstration episodes for each level. **LID-Text (Li et al. 2022)** is an approach that first represents goals and observations as a sequence of embeddings, then uses a policy network initialized with a pre-trained LM and trained with demonstrations to predict the next action. **LISA (Garg et al. 2022)** is a hierarchical imitation learning framework that can learn diverse, interpretable skills from language-conditioned demonstrations. **HYVIN-action** is the variant of HYVIN that maintains the same high-level planner but employs primitive actions instead of the acquired skills. **HYVIN-no-skills** is the variant of HYVIN that only has hypothesis and verification phases without inductive skill learning. **HYVIN-GPT4** is the variant that uses GPT-4 instead of ChatGPT to enhance performance.

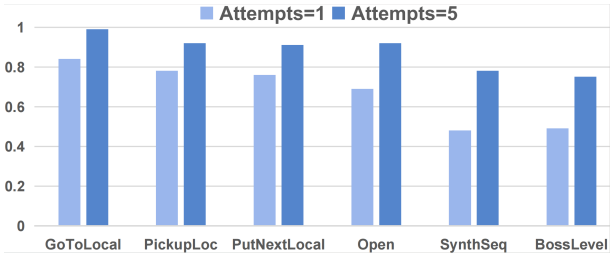

Figure 6: Successful rates of different attempts times

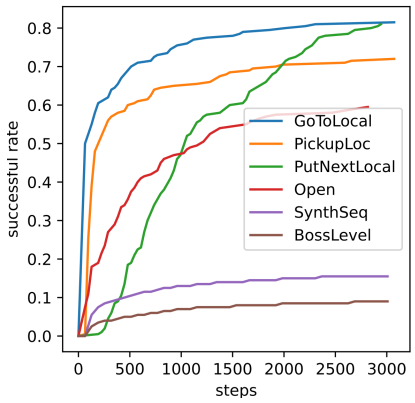

Figure 7: Task verification results

Implementation In this paper, we use ChatGPT (GPT-3.5-turbo) as the large language model to complete task decomposition, the semantic embedding of API, high-level planning, and debugging. More details on the prompt contents are shown in the Appendix. In the verification and skill learning phase, we use the standard model proposed in BabyAI, and train the policy using the PPO algorithm.

Overall Results Comparison

The main performance results are shown in Table 1. We separately compare HYVIN with baselines in each level task. For each level task, we randomly sample 100 instructions that never occurred in the skill training phase. Considering the randomness of ChatGPT's answers, we repeat the experiment of each instruction 3 times to get the average results. The results showed that HYVIN can achieve comparable performance using automatically learned skills rather than a large number of expert demonstrations, which shows the effectiveness of our framework. Besides, the results of ChatGPT, HYVIN-action, and HYVIN-no-skills showed it cannot directly solve BabyAI's tasks. The main cause is LLMs cannot cooperate with low-level primitive actions without proper grounding, which also emphasizes the importance of self-driven skill learning in the LLMs grounding scenario.

Ablation Results

Skill Learning Ablation We first investigate the data efficiency when learning skills through reinforcement learning as shown in Figure 4 . Although the difficulty varies because the clustering process only relies on the semantics of subgoals and ignored the difficulty, skills can be learned efficiently. The verification phase proves that the instruction has been decomposed into small enough subgoals and can be trained in limited steps. Therefore, the skill training environment consisting of verified subgoals can lead to an efficient skill training process. Besides, results also demonstrate that restoring actions that make the skill start at an expected state is important to learning efficiency. Without the start state reset, the learning efficiency drops obviously, and some skills even cannot be learned. Some learning curves show overfitting trends in the late period, which is different from the normal reinforcement learning process and Illustrates the role of the hold-out validation set. For some skills like "enter_unexplored_room", the green curves seem better than the blue ones. This is because without restoring the initial state, the difficulty decreases a lot since the agent can enter any room.

Deduction Ablation We also explored the effect of the interaction debugging times and multiple attempts of skills. The ablation results show the importance of interactive debugging and multiple attempts.

A. Interaction Times: Figure 5 shows the results of different interaction times between the high-level planner and the environment. For some tasks like "GoToLocal", the successful rate promotion is limited because of simplicity. For complex tasks like "BossLevel", repeat debugging can bring more than 40% promotion, which shows the adaptability and feasibility of learned skills. However, when the performance reaches some ceiling bound, more interaction seems useless.

B. Multiple Attempts: Different from pre-defined APIs with scripts, our learned skills are stochastic policies. Thus, we also investigate the effect of multiple attempts of skill policy on the final success rate. Figure 6 shows similar results with interaction times, the multiple attempts improve complex tasks greater.

Method Details

To show more insight into HYVIN, we show some key intermediate results. **Task Verification:** Figure 7 shows the verification results of different level tasks. In the implementation, we random sample 100 instructions from each level task, and set the verification steps threshold T_{verify} equals to 3000. The results prove our assumption, for some simple task levels, like "GoToLocal", "PickupLoc" and "PutNextLocal", the LLM can decompose the instruction into reasonable subgoals and check functions, so that the verification training be successful in limited steps. For hard levels which also include some simple tasks, direct decomposition can accomplish few instructions, which means it cannot solve complex and long-term tasks. We also show more details of **Skill Clustring** in Appendix.

Tools	$0 \rightarrow$ wood	wood $\rightarrow$ stone	stone $\rightarrow$ iron
	66.9%	25.8%	2.1%

Table 2: Verification in simple tasks with 50k steps

	HYVIN	DreamerV2
$0 \rightarrow$wood	92.3%	92.7%
$0 \rightarrow$wood pickaxe	**90.6%**	59.6%
$0 \rightarrow$stone	31.4%	**42.7%**
$0 \rightarrow$stone pickaxe	**32.5%**	0.2%
$0 \rightarrow$iron	**0.7%**	0%
$0 \rightarrow$iron pickaxe	0%	0%

Table 3: Deduction in complex tasks

A Complex Case Study

We make a case study on a more complex Minecraft-like environment, Crafter (Hafner 2021), to show the effectiveness of HYVIN. In the hypothesis and verification phases, HYVIN initially accomplishes simple tasks that only include one-stage complexity and collects grounding experiences, as shown in Table 2. Then in the induction phase, HYVIN learns some useful skills like collecting materials and making tools to solve complex tasks, as shown in Table 3. The results show that HYVIN significantly outperforms the SOTA learning-based method, DreamerV2, in Crafter.

Conclusion

In this paper, we propose a framework called Hypothesis, Verification, and Induction (HYVIN) to address the challenge of automatically grounding LLM onto specific environments. In order to alleviate the problem of grounding experience acquisition, we make the LLM not only decompose tasks but also generate intrinsic rewards to help RL agents efficiently verify the decomposition results. We also propose a language-aligned general skill learning methodology by forcing each skill to achieve a group of goals with similar semantic descriptions to enhance their generality. Compared with imitation learning methods that cost millions of demonstrations, HYVIN can achieve comparable performance in the hardest tasks in BabyAI. The ablation study also shows the flexibility and feasibility of learned skills in the interactions between the high-level planner and the environment.

We leave introducing multi-modal LLMs to extend the applications of HYVIN as future works. Besides, considering HYVIN only contains a single cycle of hypothesis, verification, and induction. It is an interesting and promising direction to design a mechanism of multiple cycles in HYVIN, allowing HYVIN to learn more powerful and diverse hierarchical skills to accomplish more flexible tasks.

Acknowledgements

This work is partially supported by the NSF of China (under Grant 92364202).

References

Artzi, Y.; and Zettlemoyer, L. 2013. Weakly Supervised Learning of Semantic Parsers for Mapping Instructions to Actions. *Trans. Assoc. Comput. Linguistics*, 1: 49–62.

Bahdanau, D.; Hill, F.; Leike, J.; Hughes, E.; Hosseini, S. A.; Kohli, P.; and Grefenstette, E. 2019. Learning to Understand Goal Specifications by Modelling Reward. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Carta, T.; Romac, C.; Wolf, T.; Lamprier, S.; Sigaud, O.; and Oudeyer, P. 2023. Grounding Large Language Models in Interactive Environments with Online Reinforcement Learning. *CoRR*, abs/2302.02662.

Chaplot, D. S.; Sathyendra, K. M.; Pasumarthi, R. K.; Rajagopal, D.; and Salakhutdinov, R. 2018. Gated-Attention Architectures for Task-Oriented Language Grounding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 2819–2826.

Chen, V.; Gupta, A.; and Marino, K. 2021. Ask Your Humans: Using Human Instructions to Improve Generalization in Reinforcement Learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*.

Chevalier-Boisvert, M.; Bahdanau, D.; Lahlou, S.; Willems, L.; Saharia, C.; Nguyen, T. H.; and Bengio, Y. 2019. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Driess, D.; Xia, F.; Sajjadi, M. S. M.; Lynch, C.; Chowdhery, A.; Ichter, B.; Wahid, A.; Tompson, J.; Vuong, Q. H.; Yu, T.; Huang, W.; Chebotar, Y.; Sermanet, P.; Duckworth, D.; Levine, S.; Vanhoucke, V.; Hausman, K.; Toussaint, M.; Greff, K.; Zeng, A.; Mordatch, I.; and Florence, P. R. 2023. PaLM-E: An Embodied Multimodal Language Model. *ArXiv*, abs/2303.03378.

Garg, D.; Vaidyanath, S.; Kim, K.; Song, J.; and Ermon, S. 2022. LISA: Learning Interpretable Skill Abstractions from Language. *ArXiv*, abs/2203.00054.

Hafner, D. 2021. Benchmarking the Spectrum of Agent Capabilities. *arXiv preprint arXiv:2109.06780*.

Huang, D.; Nan, Z.; Hu, X.; Jin, P.; Peng, S.; Wen, Y.; Zhang, R.; Du, Z.; Guo, Q.; Pu, Y.; and Chen, Y. 2023. ANPL: Towards Natural Programming with Interactive Decomposition.

Huang, W.; Xia, F.; Xiao, T.; Chan, H.; Liang, J.; Florence, P.; Zeng, A.; Tompson, J.; Mordatch, I.; Chebotar, Y.; Sermanet, P.; Jackson, T.; Brown, N.; Luu, L.; Levine, S.; Hausman, K.; and Ichter, B. 2022. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, 1769–1782.

Ichter, B.; Brohan, A.; Chebotar, Y.; Finn, C.; Hausman, K.; Herzog, A.; Ho, D.; Ibarz, J.; Irpan, A.; Jang, E.; Julian, R.; Kalashnikov, D.; Levine, S.; Lu, Y.; Parada, C.; Rao, K.; Sermanet, P.; Toshev, A.; Vanhoucke, V.; Xia, F.; Xiao, T.; Xu, P.; Yan, M.; Brown, N.; Ahn, M.; Cortes, O.; Sievers, N.; Tan, C.; Xu, S.; Reyes, D.; Rettinghouse, J.; Quiambao, J.; Pastor, P.; Luu, L.; Lee, K.; Kuang, Y.; Jesmonth, S.; Joshi, N. J.; Jeffrey, K.; Ruano, R. J.; Hsu, J.; Gopalakrishnan, K.; David, B.; Zeng, A.; and Fu, C. K. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, 287–318.

Kojima, T.; Gu, S. S.; Reid, M.; Matsuo, Y.; and Iwasawa, Y. 2022. Large Language Models are Zero-Shot Reasoners. *ArXiv*, abs/2205.11916.

Li, S.; Puig, X.; Paxton, C.; Du, Y.; Wang, C.; Fan, L.; Chen, T.; Huang, D.; Akyürek, E.; Anandkumar, A.; Andreas, J.; Mordatch, I.; Torralba, A.; and Zhu, Y. 2022. Pre-Trained Language Models for Interactive Decision-Making. In *NeurIPS*.

Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2022. Code as Policies: Language Model Programs for Embodied Control. *CoRR*, abs/2209.07753.

Lynch, C.; and Sermanet, P. 2021. Language Conditioned Imitation Learning Over Unstructured Data. In *Robotics: Science and Systems XVII, Virtual Event, July 12-16, 2021*.

Misra, D. K.; Sung, J.; Lee, K.; and Saxena, A. 2016. Tell me Dave: Context-sensitive grounding of natural language to manipulation instructions. *Int. J. Robotics Res.*, 35(1-3): 281–300.

OpenAI. 2023. GPT-4 Technical Report. *ArXiv*, abs/2303.08774.

Peng, S.; Hu, X.; Zhang, R.; Guo, J.; Yi, Q.; Chen, R.; Du, Z.; Li, L.; Guo, Q.; and Chen, Y. 2023. Conceptual Reinforcement Learning for Language-Conditioned Tasks. In *AAAI Conference on Artificial Intelligence*.

Raman, S. S.; Cohen, V.; Rosen, E.; Idrees, I.; Paulius, D.; and Tellex, S. 2022. Planning with Large Language Models via Corrective Re-prompting. *CoRR*, abs/2211.09935.

Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds. 2021. *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*.

Röder, F.; Eppe, M.; and Wermter, S. 2022. Grounding Hindsight Instructions in Multi-Goal Reinforcement Learning for Robotics. In *IEEE International Conference on Development and Learning, ICDL 2022, London, United Kingdom, September 12-15, 2022*, 170–177.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. *CoRR*, abs/2305.16291.

Wang, R.; Jansen, P. A.; Côté, M.; and Ammanabrolu, P. 2022. ScienceWorld: Is your Agent Smarter than a 5th

Grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, 11279–11298.

Wei, J.; Bosma, M.; Zhao, V.; Guu, K.; Yu, A. W.; Lester, B.; Du, N.; Dai, A. M.; and Le, Q. V. 2021. Finetuned Language Models Are Zero-Shot Learners. *ArXiv*, abs/2109.01652.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; hsin Chi, E. H.; Xia, F.; Le, Q.; and Zhou, D. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *ArXiv*, abs/2201.11903.

Yuan, H.; Zhang, C.; Wang, H.; Xie, F.; Cai, P.; Dong, H.; and Lu, Z. 2023. Plan4MC: Skill Reinforcement Learning and Planning for Open-World Minecraft Tasks. *CoRR*, abs/2303.16563.