SUB-GOAL DISTILLATION: A METHOD TO IMPROVE SMALL LANGUAGE AGENTS

Maryam Hashemzadeh*
Mila – Quebec AI Institute, Université de Montréal
maryam.hashemzadeh@mila.quebec

Elias Stengel-Eskin
UNC Chapel Hill
esteng@cs.unc.edu

Sarath Chandar†
Mila – Quebec AI Institute, Polytechnique Montréal, CIFAR AI Chair
sarath.chandar@mila.quebec

Marc-Alexandre Côté†
Microsoft Research, Montréal
macote@microsoft.com

ABSTRACT

While Large Language Models (LLMs) have demonstrated significant promise as agents in interactive tasks, their substantial computational requirements and restricted number of calls constrain their practical utility, especially in long-horizon interactive tasks such as decision-making or in scenarios involving continuous ongoing tasks. To address these constraints, we propose a method for transferring the performance of an LLM with billions of parameters to a much smaller language model (770M parameters). Our approach involves constructing a hierarchical agent comprising a *planning module*, which learns through Knowledge Distillation from an LLM to generate sub-goals, and an *execution module*, which learns to accomplish these sub-goals using elementary actions. In detail, we leverage an LLM to annotate an oracle path with a sequence of sub-goals towards completing a goal. Subsequently, we utilize this annotated data to fine-tune both the planning and execution modules. Importantly, neither module relies on real-time access to an LLM during inference, significantly reducing the overall cost associated with LLM interactions to a *fixed cost*. In ScienceWorld, a challenging and multi-task interactive text environment, our method surpasses standard imitation learning based solely on elementary actions by 16.7% (absolute). Our analysis highlights the efficiency of our approach compared to other LLM-based methods. Our code and annotated data for distillation can be found on GitHub[1].

1 INTRODUCTION

Recently, Large Language Models (LLMs) have found applications in various fields, including multi-task learning, decision making, answering questions, summarizing documents, translating languages, completing sentences, and serving as search assistants. They showcase a remarkable ability to make predictions based on input, enabling their use in generative AI applications to produce content based on input prompts (Devlin et al., 2018; Brown et al., 2020; Rae et al., 2021; Chowdhery et al., 2023; Scao et al., 2022; Patel & Pavlick, 2021; Han et al., 2021; Bommasani et al., 2021).

The promising advantage of LLMs is attributed to their training on extensive text datasets, resulting in impressive capabilities. This prior knowledge can be leveraged for action planning to solve tasks in robotics and reinforcement learning (Huang et al., 2022b; Brohan et al., 2023; Liang et al., 2023). Recent works have utilized in-context learning with LLMs to provide actions in autonomous decision-making agents and interactive environments (Mahowald et al., 2023; Yao et al., 2022; Schick et al., 2023; Shen et al., 2023; Nakano et al., 2021; Park et al., 2023; Lin et al., 2023; Brohan et al., 2023).

However, the extreme size of LLMs makes them computationally unaffordable for many applications. Moreover, closed-source models like ChatGPT (OpenAI, 2022) and GPT-4 (OpenAI, 2023) limit accessibility and reproducibility. Consequently, there is an increasing demand to find approaches that are less computationally intensive while still capitalizing on the knowledge embedded in LLMs. One prevalent technique is the use of Knowledge Distillation (KD) (Buciluă et al., 2006; Hinton et al., 2015), wherein a smaller model is trained with guidance from a larger model.

* Corresponding author: Maryam Hashemzadeh. †equal supervision.
[1] https://github.com/chandar-lab/SubGoal_Distillation_LLM

Through this approach, we can leverage the knowledge in an LLM to train a more compact model with a reduced number of parameters.

Distilling knowledge from LLMs offers significant advantages, allowing for the training of specialized local models rather than depending on an LLM as a general model. This approach not only enhances privacy, particularly for systems with security-sensitive considerations like co-pilot models, but also provides greater flexibility in tailoring models for specific tasks. Additionally, the use of a smaller model offers the advantage of versatility across diverse applications without size constraints, including device models and mobile apps. Another challenge with LLMs is their susceptibility to hallucinations. This tendency poses a hindrance to their effective execution of long-tail planning, especially in interactive decision-making scenarios.

In our research, we leverage the knowledge of LLMs to train an autonomous agent for effective decision-making in complex interactive text environments, utilizing small language models as our policy. Knowledge Distillation facilitates the training of smaller policies, allowing seamless integration of LLM knowledge. To address the challenges at hand, adopting a two-level planning approach proves beneficial for reducing hallucination – one for high-level reasoning to formulate sub-goals and another for low-level action planning to execute each sub-goal.

Figure 1 illustrates this concept in the task of freezing water from ScienceWorld (Wang et al., 2022a). The agent's subtasks involve navigating to the kitchen, finding a thermometer and a metal pot, pouring water into the pot, placing it in the freezer, and continuously monitoring its temperature until frozen. These constitute sub-goals generated by a high-level model, with each sub-goal subsequently executed by a low-level model. The generation of sub-goals empowers an autonomous agent to expedite learning for the current task and reuse similar sub-goals in various tasks to have more generalization.

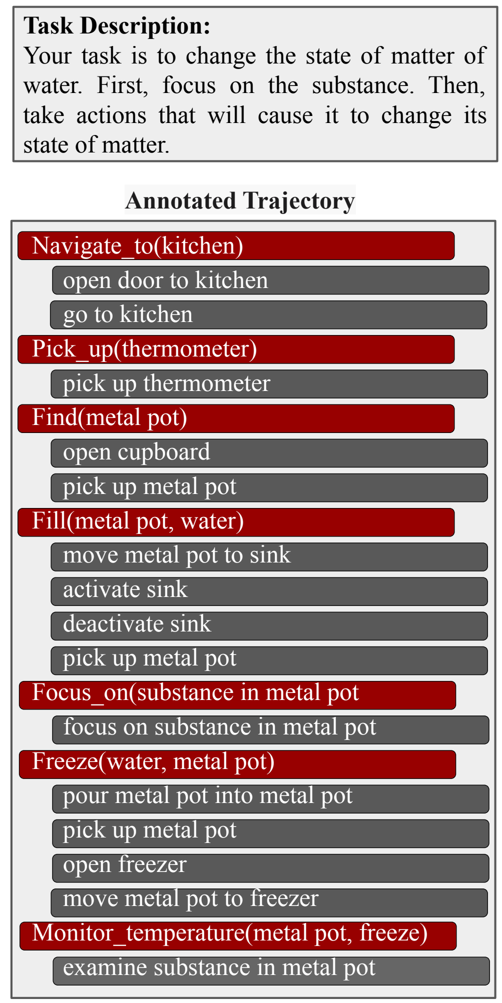

Figure 1: Example of annotating an expert trajectory with sub-goals for a particular variation of task 1-4 (*change-the-state-of-matter-of*). Looking only at the original trajectory (i.e., ignoring the red rows), we gather the expert ended up changing the state of *water* to be frozen. The expert had to navigate to the kitchen, find a thermometer and a metal pot, pour water into the pot, place it in the freezer, and continually monitor its temperature until frozen. Each of those milestones (highlighted in red) can be considered a sub-goal, encompassing a sequence of actions. Sub-goals can be shared across different tasks, facilitating generalization. We opted for a format that looks like function calls to encourage reusability (e.g., *fill(metal pot, water)*).

The contributions in this work are:

- We employ Knowledge Distillation from an LLM to train a high-level policy capable of generating sub-goals without making assumptions about the specific set of sub-goals. Notably, these sub-goals remain flexible, accommodating various sequences of actions.
- We demonstrate that employing Knowledge Distillation with hierarchical policies surpasses the performance achieved by both standalone imitation learning and its combination with in-context learning.
- We illustrate that this approach is more cost-effective in terms of the number of calls to an LLM compared to other methods utilizing in-context learning.
- We introduce an effective approach instead of using computational requirements of LLM and their restricted number of calls for using in interactive decision making tasks.

2 Related Work

Using LLMs for Action Planning Recent works have demonstrated the ability of LLMs to perform action planning for interactive decision making process without any additional training (Huang et al., 2022a). ReAct (Yao et al., 2022) proposes a way of prompting an LLM with interleave reasoning step and action taking step. That led the resolution of a variety of language-based reasoning and decision-making tasks. This approach empowers the model to construct high-level plans for effective action. Reflexion (Shinn et al., 2023) draws inspiration from reinforcement learning, employing a framework to reinforce language agents through linguistic feedback. At the end of each trial, it uses self-reflection to determine what went wrong with the task and keeps it in a memory. Then it leverages this information for the next trial.

Some works use a programmatic LLM prompt structure with available actions and objects in an environment to translate natural language commands into robot policy code via few-shot examples (Liang et al., 2023; Singh et al., 2023). Khot et al. (2022) introduced a decomposed prompting approach wherein a task is broken down into simpler sub-tasks, allowing for recursive handling. Subsequently, these sub-tasks are assigned to sub-task-specific LLMs, with both the decomposer and the sub-task LLMs with their own few-shot prompts. Sun et al. (2023) uses three steps, action mining, plan formulation, and plan execution to decompose a question into a sequence of actions by few-shot prompting of LLMs. In Prasad et al. (2023) tasks are decomposed explicitly by a separate LLM through prompting when an executor is unable to execute a given sub-task.

Imitation learning Some works employ imitation learning to train a language model as the agent's policy, as seen in offline decision transformers (Torabi et al., 2018). The inputs consist of states, actions, and reward-to-go values, which are fed into a transformer. This transformer then predicts actions in an autoregressive manner, utilizing a causal self-attention mask (Chen et al., 2021). Contextual Action Language Model (CALM) (Yao et al., 2020) is another work which uses a fine-tuned language model with oracle data to generate a set of candidate actions which are then passed to a policy network to select the best one. In Nakano et al. (2021), the authors fine-tune GPT-3 to address long-form questions within a web-browsing context. Human feedback is employed as a direct optimization measure for enhancing the quality of answers generated by the model.

Knowledge Distillation: Knowledge Distillation (KD) typically falls into two categories: black-box KD and white-box KD. In black-box KD, only the teacher's predictions are available for guidance, while in white-box KD, we have access to the teacher's parameters (Gou et al., 2021). Recently, black-box KD has gained widespread use for fine-tuning original models using self-instruct techniques, as proposed by Wang et al. (2022b), or for smaller models (Taori et al., 2023; Chiang et al., 2023; Peng et al., 2023) through the utilization of prompt-response pairs generated by LLMs. West et al. (2021) introduces symbolic KD from text rather than logits. This process involves the transfer of knowledge from a large, general model to a more compact commonsense model, facilitated by a commonsense corpus, yielding a commonsense knowledge graph and model. The work by Hsieh et al. (2023) trains a smaller model that outperform LLM using reasoning steps called rationales. They incorporated rationales as informative supervision to train smaller models with less training data.

Complex interactive text environments In text-based games, an agent interacts with the environment by reading and writing text while aiming towards the end game or solving a given task. Out of the recent frameworks that deals with generating and interfacing text-based games (Côté et al., 2018; Hausknecht et al., 2019; Shridhar et al., 2021; Murugesan et al., 2021), we use ScienceWorld (Wang et al., 2022a) which is very complicated by having a large set of objects, actions, and tasks.

3 Model

In this paper, we propose to train a hierarchical policy by combining KD from an LLM and imitation learning from expert trajectories. This section describes both modules in detail and we refer the reader to Figure 2 for a schematic view. We first formulate the problem as a POMDP (Section 3.1). Next, we describe what knowledge we are distilling from an LLM to guide the agent in accomplishing tasks (Section 3.2). Then, we detail how both the high-level and low-level policies of the hierarchical policy are trained (Section 3.3).

3.1 Problem Formulation

ScienceWorld (Wang et al., 2022a) can be defined as a partially observable Markov decision process (POMDP), where observations provide information solely on environmental changes induced by the current action. ScienceWorld is

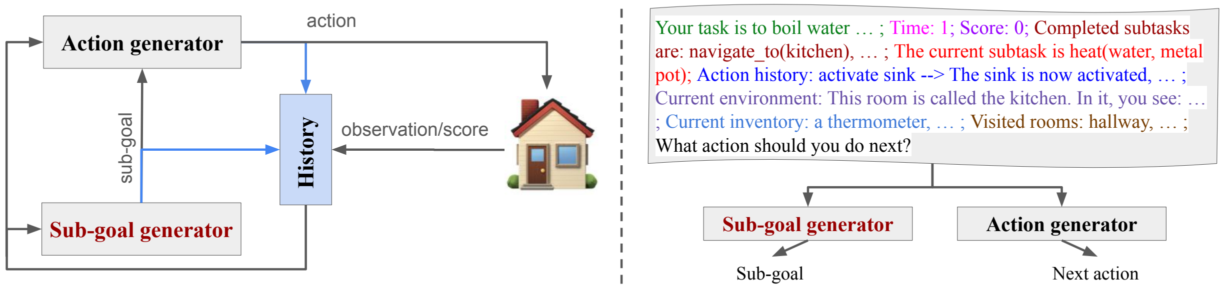

Figure 2: On the left, a schematic view of our approach is shown. There are two modules: the sub-goal generator and action generator. The sub-goal generator provides a sub-goal for the action generator, which predicts the next action given the current sub-goal and history. On the right, the inputs and outputs of both modules are illustrated. The input comprises different parts including task description, completed sub-goal, current sub-goal, a history of recent actions-observations, and more, each highlighted in a distinct color.

an interactive text environment meaning all task instructions, observations and actions are expressed in textual form. Importantly, both observations and rewards in this environment are conditioned by the ongoing task.

Given a language vocabulary V and an arbitrary maximum number of tokens N, an observation is defined such as $o \in \Omega \subset V^N$, a reward such as $r \in \mathrm{R}$ and an action as $a \in A \subset V^N$. Finally, a task or goal description is shown by $g \in G \subset V^N$.

We formalize the problem as a goal-augmented POMDP $M = (S, V, A, \Omega, G, T, R, O, \gamma)$ with S the state space, $A \subset V^N$ the action space, $\Omega \subset V^N$ the observation space, $G \subset V^N$ the goal space, $T : S \times A \times G \to S$ the goal-conditioned transition function, $R : S \times A \times G \to \mathrm{R}$ the goal-conditioned reward function, $O : S \to V^N$ an (unknown) observation function mapping a state to a textual description and γ the discount factor. We assume $\gamma = 1$ in our experiments.

3.2 Distilling Knowledge from an LLM

The initial step in training our policies is creating a dataset. This dataset should include sub-goals along with their corresponding aligned sequences of actions for each task.

To generate sub-goals along with their corresponding aligned sequences of actions we do the following steps. We assume access to a collection of expert trajectories. Then we prompt an LLM with two in-context examples. Each example is composed of a task description, a similar task as the one we wish to annotate, and its expert trajectory. The example also contains a set of sub-goals, with the sequences of actions linked to each sub-goal.

Given the two in-context examples and a new task description with its expert trajectory, the LLM is then instructed to generate a response. The response is a set of sub-goals with their associated list of actions. The generated list of actions is used to determine each sub-goal corresponds to which segment of the expert trajectory. It is important to note that these responses are collected only for the training tasks for which we assume having access to expert trajectories. Also, it is important to point out that the LLM is not generating any novel trajectories.

Figure 4 illustrates the prompt examples for task $1 - 1$ which is boiling a given substance. To ensure more uniform sub-goals that can generalize across tasks, we opted for a format that looks like function calls. Since that format was shown in the in-context examples, the LLM-generated sub-goals mimic this format as well making them easier to parse.

Since the expert trajectories for some tasks can be long (+100 actions), the generated sub-sequence of actions corresponding to each sub-goal may not align exactly with the expert trajectory. Sometimes, it might miss certain actions, while in other instances, it might include additional actions, especially when there are repeated actions in the trajectory. To address this, we use a trajectory alignment process that finds the minimal set of edits to go from the generated trajectory to the expert trajectory according to the Levenshtein distance. For each "remove" edit, i.e. the generated trajectory has superfluous actions, we simply remove those from the generated trajectory. On the other hand, for "add" edit, i.e. the generated trajectory is missing some actions, we prompt the LLM to generate a new sub-goal for those. An example is shown in Figure 3.

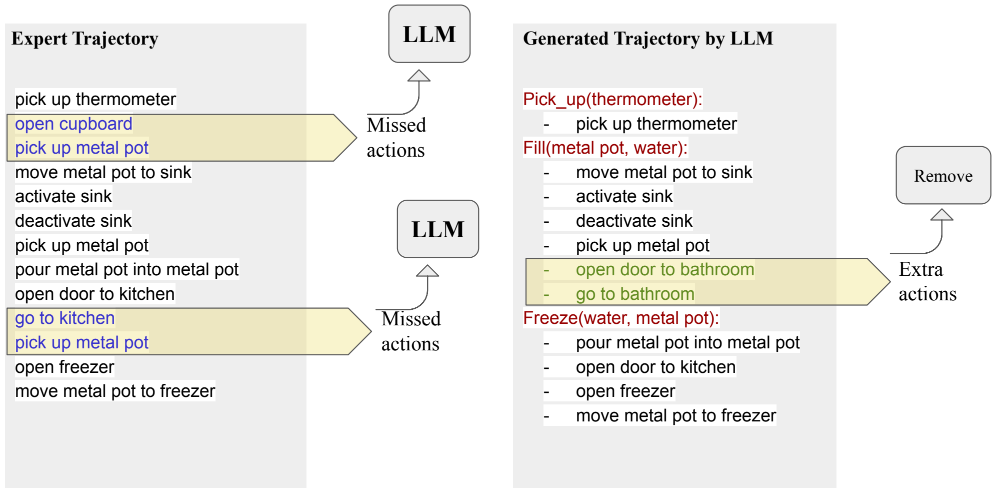

Figure 3: Example of a trajectory generated by the LLM deviating from the provided expert trajectory. In this example, which is for a boiling task, certain actions are omitted in the generated trajectory, indicated in blue in the left box. To address these missing actions, we group them into sequences and prompt the LLM to generate sub-goals for them. If the generated trajectory includes additional actions, such as the green actions in the right box, we simply remove them to align with the expert trajectory.

In the resulting annotated dataset, each data point follows the same format as used by Lin et al. (2023) but with the added mention of completed sub-goals and the current sub-goal. Precisely, it corresponds to:

- **Input**: task description, number of steps, current score, completed sub-goal, current sub-goal, a history of 10 recent actions-observations, current items in the room, inventory, and the visited rooms.
- **Target**: next action, next sub-goal.

3.3 Hierarchical Imitation Learning

With the dataset obtained from distilling knowledge from an LLM, we can now focus on training the policies.

Low-level policy: The low-level policy is a language model (LM) which is trained through imitation learning using the annotated dataset. The goal is to have a model much smaller than an LLM so it can fit on a single machine and run faster, ideally below a billion of parameters. This policy learns to predict the next action given the current task description, the 10 previous observation-action pairs, the previous completed sub-goals, and the current sub-goal. We refer to this policy as the **action generator**.

High-level policy: The high-level policy is another LM with a reasonable size. It is trained using the annotated dataset to generate the next sub-goal given the previous sub-goals and a short history, i.e. the last 10 actions and observations. So the high-level policy generates sub-goals while the low-level policy generate actions. Moreover, this policy conditions on the same input information as for the action generator. We call this policy the **sub-goal generator**.

Hierarchical policy: During inference, we first leverage the high-level policy to generate a sub-goal. This generated sub-goal is then fed into the action generator, allowing it to produce the next action aligned with the provided sub-goal. This sequential approach serves as a guiding cue for the action generator, particularly when the trajectory to achieve the goal is complex or long. Moreover, it serves to prevent the action generator from generating actions that might deviate the agent from the correct path, thereby improving the precision and relevance of the actions being generated.

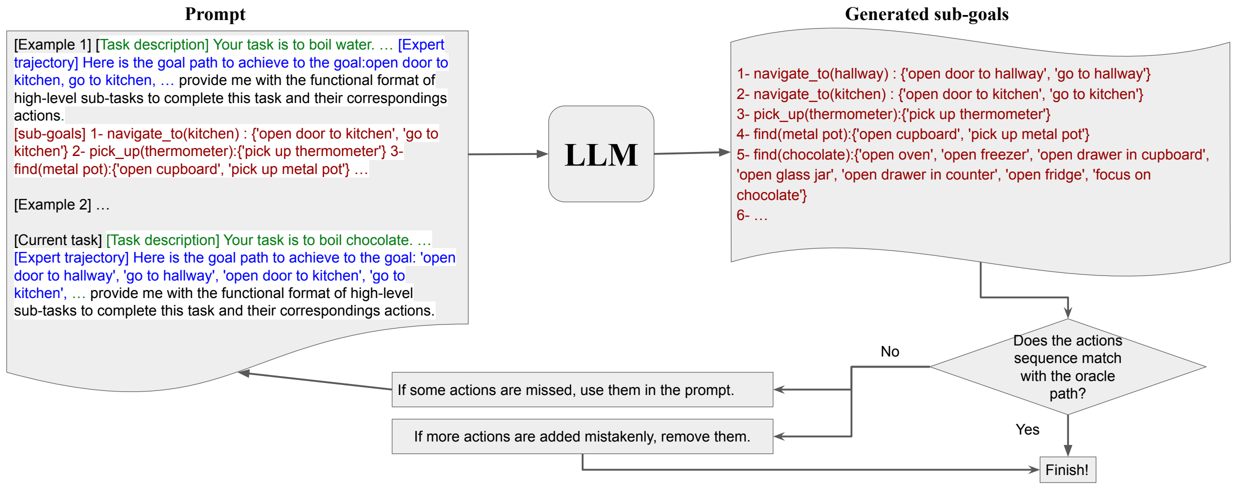

Figure 4: The figure demonstrates KD to generate sub-goals using an LLM. The LLM is presented with a prompt containing two in-context examples. Each example is composed of a task description in green and an expert trajectory detailing the steps to accomplish that task in blue. It also includes the expected set of sub-goals with their corresponding sequences of actions in red. Following this, we provide a new task description and trajectory, and we let the LLM generate the associated sub-goals and segmented actions.

4 Experiments

4.1 Environment

We chose ScienceWorld (Wang et al., 2022a) as the environment due to its complexity and the diverse range of tasks it encompasses. This environment is an interactive multi-task text-based game where the agent conducts elementary science experiments in a simulated environment. Each experiment is designed as a separate task. For example, *"Your task is to boil water. For compounds without a boiling point, combusting the substance is also acceptable. First, focus on the substance. Then, take actions that will cause it to change its state of matter"*. To complete a task, the agent must perform multiple actions and receives the result of each action as an observation and a score. The observations and actions are in text format. An observation describes the changes in the environment, and the score is a numerical value ranging from 0% to 100%, indicating the degree of completion of the current task through the current action.

Furthermore, ScienceWorld is a benchmark with 30 distinct tasks spanning 10 science domains which are widely different (Appendix A.4). For instance, in the "Changes of State" task, the agent is required to locate and use heating/freezing sources to alter the state of a substance (e.g., ice or chocolate). Conversely, in a task such as "Mendelian Genetics," the agent is tasked with determining whether a specified trait (e.g., white flower color) is dominant or recessive in a plant. These examples illustrate the substantial diversity across the domains, ranging from physical transformations to genetic analyses, underscoring the broad spectrum of challenges within ScienceWorld.

On top of that, ScienceWorld has 10 different locations, more than 200 object types, and 25 action templates which makes the search space very larger for the agent. Each type of task has different variations in which the task objects, the agent's initial location, and random contents of each room are altered.

4.2 Experimental Setup

The environment has separate sets of variations for train and test. In the test variations, the combinations of objects and conditions are not seen in the train set. Following the experimental setup in (Lin et al., 2023), if the number of variations is more than 10, we consider only the first 10 variations.

Our base models for both policies is a pre-trained FLAN-T5-LARGE (Chung et al., 2022) with 700M parameters. For the both polices, we used greedy decoding at inference. We also conduct an ablation study over different model sizes (Figure 5a). For fine-tuning the policies, we use all the training tasks and their variations (3600 games in total) from ScienceWorld. We vary the number of training epochs in function of the size of the models (see Appendix A.3).

Methods	SayCan*	ReAct*	Reflexion*	Swift-only	SwiftSage*	Ours
Overall Average	25.22	19.76	23.40	46.25	62.22	65.43
Solved Task Types	0/30	0/30	4/30	4/30	2/30	11/30
Short†	37.24	28.95	39.19	79.68	72.81	91.61
Medium	20.06	21.09	14.73	35.80	55.34	62.83
Long	18.66	11.23	16.27	25.36	57.99	45.35
Task 1-1	33.06	3.52	4.22	15.0	58.0	16.22
Task 3-3	99.56	76.19	72.54	59.5	66.9	5.6

Table 1: The table illustrates the overall average score (%) across all test tasks on the ScienceWorld benchmark for SayCan, ReAct, Reflexion, Swift-only, SwiftSage, and our algorithm (last column). The *Solved Task Types* row represents the number of task types for which an agent manages to solve all the test variations. The table also shows the average scores for tasks with a short, medium, and long length of expert trajectory. The rows *Task 1-1* and *Task 3-3* display the scores for each of them in which our approach does not work well in comparison with the other methods. The * denotes scores reported from (Lin et al., 2023) which all use ChatGPT (GPT-3.5).

4.3 Baseline Agents

We compare our approach with other works that leverage LLMs. Some rely only on prompting such as SayCan, ReAct, and Reflexion, but SwiftSage also do imitation learning. Here is a brief description of each method.

SayCan: the LLM initially offers a set of actions along with their respective ranks. Then, a value-based method is employed to re-rank these actions in order to determine the most rewarding action for execution (Brohan et al., 2023).

ReAct: the LLM generates actions by incorporating the provided prompt and the history of generated texts. It employs reasoning traces as intermediate thought steps during the action generation to refine a plan for the upcoming steps (Yao et al., 2022).

Reflexion: the language agent reflects the task feedback at each trial in the form of text and retains this information within an episodic memory. During the subsequent trial, it leverages the stored memory text to enhance its decision-making process (Shinn et al., 2023).

SwiftSage: this method comprises two components: Swift, a fine-tuned LM to predict actions, and Sage, a module that queries an LLM for planning when the performance of Swift is inadequate (as determined by some handcrafted rules) (Lin et al., 2023).

Swift-only: this is the Swift part of the SwiftSage method which only has the fine-tuned LM to predict the actions. We consider this method as a strong baseline and the most comparable to our approach as it relies on imitation learning without the need for querying an LLM during inference.

Note that all baselines use ChatGPT (GPT-3.5) as their LLM.

4.4 Results and Analysis

Main Results: Table 1 compares the performance of the baselines with our approach in the ScienceWorld. The score for each task type is the average score (in percent) obtained for 10 test variations. Our approach demonstrates an overall performance of **65.43%**, surpassing Swift-only by 16.71% (33.9% relative increase), and showing a slight improvement over SwiftSage of 3.3% (5.3% relative). Interestingly, our method is able to solve all test variations (i.e., gets an average score of 100%) for 11 out of the 30 task types. In contrast, SwiftSage solves them only for 2 task types, and Swift-only, only for 4 task types.

Additionally, we measured the performance of the agents with respect to the length of the tasks (a proxy for task complexity). The length of a task is determined by how many actions was needed by the expert to solve it.[2] Following Lin et al. (2023), we group the tasks into three categories: *Short* when the length is less than 20 actions, *Medium* when it falls between 20 and 50 (inclusively), and *Long* if above 50. As shown in Table 1, our approach outperforms

[2]Expert trajectories for test tasks were not seen during training.

other methods on short and medium tasks. On long tasks, we outperform all methods except SwiftSage, which has a substantial advantage here: The longer the task, the higher the chance it triggers one of the rules for Sage to take over.

As part of the comparison, there are other approaches that do not use a LLM including DRRN (He et al., 2016), KG-A2C (Ammanabrolu & Hausknecht, 2019), CALM (Yao et al., 2020), BC (Torabi et al., 2018), TDT (Chen et al., 2021). The results from (Wang et al., 2022a) show these approaches perform poorly, below 17%, in ScienceWorld. For this reason, we did not include them here and only focus on approaches comparable with us.

A key motivation for our approach is cost-effectiveness in terms of LLM queries. During training, we make one query to ChatGPT per task to identify the sub-goals within an expert trajectory. Sometimes mismatches occur between the expert trajectory and the actions assigned to each sub-goal by ChatGPT. When that is the case, we employ dynamic programming, with a maximum of 10 attempts per task. This contrasts with other baseline methods, where LLM is queried for each action, incurring considerably higher costs.

Why is it failing on some task types? The performance of our algorithm in some tasks are low, (see Table 5). In Table 1, the scores of two tasks are presented. One contributing factor is the variations in the test are very different from those in the training. For instance, the objects might be very different or the path to complete the task is very different and longer. The main culprit is the sub-goal generator which is not able to generate good sub-goals.

As a concrete example (Table 2), in the test variations for task 3-3, the agent needs to go to kitchen and then fill a jug with water. When looking at the transcript, we see the agent is able to go to kitchen but then when it arrives, the sub-goal generator issues a sub-goal which is not relevant, `FocusOn(fountain)`. The agent attempts to focus on the fountain which is a wrong action and the game terminates with a score of 0.

Another example is task 1-1 (Table 2) in which the agent should boil a substance. It should first find the substance but since the substance is in a totally different location than those seen during training, the sub-goal generator is not able to generate a good sub-goal for this step. Consequently the agent will do other actions and exhaust all the allocated time steps.

Example (task 3-3)		Example (task 1-1)	
With Sub-goal	**Expert Trajectory**	**With Sub-goal**	**Expert Trajectory**
NavigateTo(kitchen)	-	NavigateTo(kitchen)	-
- go to art studio	- go to art studio	- go to art studio	- go to art studio
- go to outside	- go to outside	- go to outside	- go to outside
- go to kitchen	- go to kitchen	- go to hallway	- go to hallway
FocusOn(fountain)	-	NavigateTo(bedroom)	-
-focus on fountain	- move jug to sink	-go to bedroom	- go to workshop
	- activate sink		- pick up metal potcontaining gallium
	- deactivate sink		
	- pick up jug		

Table 2: Two instances where the performance of our algorithm is low. The first column displays the trajectory generated with sub-goals, while the second column presents the expert trajectory. Sub-goals are highlighted in dark red, accompanied by their corresponding actions, and incorrect actions are marked in red.

The impact of scale: We conduct a comparison across various sizes of language models such as FLAN-T5-XL, FLAN-T5-BASE, and FLAN-T5-SMALL. Additionally, we evaluate T5-3B and T5-LARGE to determine the effectiveness of FLAN-T5 versus T5. The results are illustrated in Figure 5a. In our initial findings, we observed that FLAN-T5 outperforms T5 significantly. Moreover, our results reveal a positive correlation between the LM size and its performance – larger models generally yield better results. Intriguingly, we observe that for smaller models (FLAN-T5-SMALL and FLAN-T5-BASE), not conditioning on sub-goals works slightly better than including them. This might be indicative that the sub-goal generator is not expressive enough to generate meaningful and effective sub-goals which in turn impacts the action generator policy and leads to lower scores.

The impact of sub-goals: To study the impact of the sub-goal generator's size on the overall performance, we try pairing different sizes of sub-goal generator while limiting the action generator to be small. In Figure 5b, the average scores exhibit an upward trajectory. This can be attributed to the larger sub-goal generators producing more accurate and relevant sub-goals, subsequently empowering the action generator to generate more correct actions. See Table 6 for a complete breakdown of the score per task type and per model size.

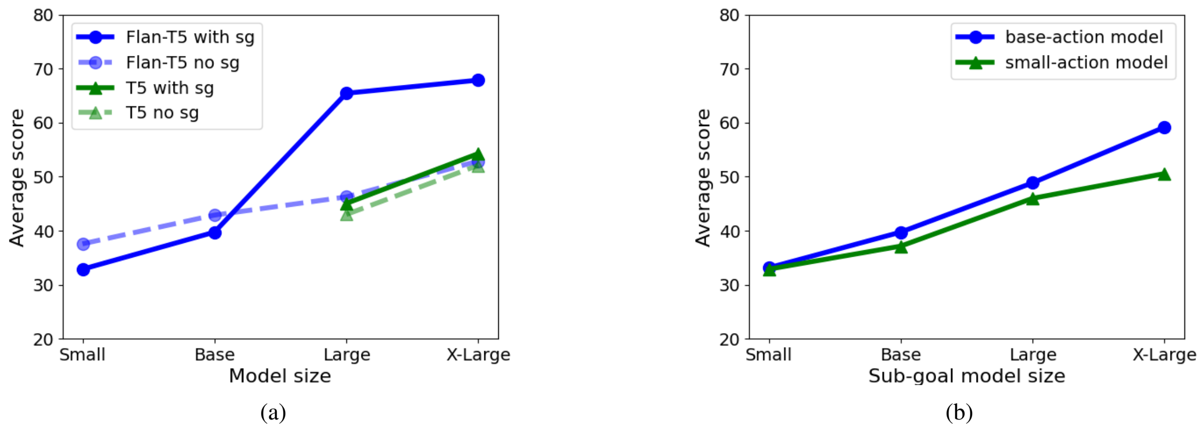

Figure 5: a) Average scores across different model sizes for FLAN-T5 and T5. For T5 model, X-Large refers to T5-3B. The larger models work better and FLAN-T5 performs also better than T5. Dashed lines represent models that are not conditioning on any sub-goals ("*no sg*") and equivalent to Swift-only. b) Average scores across different sizes of sub-goal generator while the action generator is kept to be base (blue) or small (green). Having larger sub-goal generators can significantly boost performance of small action generators.

Random			**Semi-random**		
first	10 steps	each	first	10 steps	each
39.1%	37.6%	6.4%	53.1%	43.3%	14.2%

Table 3: Average performance for randomly generated sub-goals. Sub-goals are selected randomly (or semi-randomly) at either the **first** step, every **10 steps**, or **each** step.

To further demonstrate the importance of the sub-goal, we generated random sub-goals and then fed them to the action generator. That yield an average score of **6.4%**, indicating that the action generator do condition on the sub-goals, subsequently, it cannot solve the tasks effectively. We conducted an additional experiment by altering the arguments of the sub-goals, as they have a functional format. If the argument corresponds to a location, we replaced it with another from the environment, and if it is an object, we replaced it with a randomly chosen object available at that step of the game. We named this approach *semi-random* sub-goals. The result for this experiment is **14.2%**, showing an increase in performance compared to the random sub-goals. Table 3 shows the average scores and Table 9 shows the score for each task.

Recovery from noisy sub-goals: We also assess the performance when both the action and sub-goal generators have been exposed to noisy sub-goals. More specifically, we consider two settings: applying noise 1) only at the first step, or 2) every 10 steps. In the first setting, the first sub-goal is (semi-)randomly selected, while the subsequent sub-goals are generated using the FLAN-T5-LARGE sub-goal generator. In the second experiment, a sub-goal is (semi-)randomly selected every 10 steps instead of using the sub-goal generator for all steps. Table 3 shows the overall scores for both settings and a breakdown per task types is presented in Table 10.

In both scenarios, semi-random selection (**53.1%** and **43.3%**) yields better results, as it closely resembles the sub-goals generated by the sub-goal generator. Some tasks achieve a score of 100, indicating successful recovery from noisy sub-goals. While overall scores are lower compared to using the FLAN-T5-LARGE sub-goal generator, it is still higher than using Swift only in the first setting and closely approaching it in the second setting (Appendix A.10).

Generalization on heldout task types: We select one or two task types from each science domain (see highlighted ones in Table 4) to train the action and sub-goal models. Then, we assessed their performance on the rest of the task types. We compared our algorithm against the Swift-only baseline. The average total scores are **40.63%** with sub-goals vs. **36.56%** for Swift-only. For unseen tasks, the scores are **27.72%** with sub-goals vs. **15.25%** for Swift-only. This suggests that using sub-goals helps improve generalization across unseen tasks. The scores for each task are presented in Table 11.

5 Discussion and Limitation

In contrast to SwiftSage, which relies on interactive usage of the ChatGPT API to handle planning, our approach makes use of a trained sub-goal generator to guide the action generator. Moreover, our framework empowers the agent to retrieve a nearly optimal trajectory by supplying the appropriate sub-goal. Nevertheless our framework has significantly reduced the frequency of API calls, which are both expensive and not universally accessible. ReAct, Reflexion, and SwiftSage require human annotations to correct sub-goals and predict a reasonable action. However in our approach, we do not need human help to predict sub-goals or provide precise prompts.

Generalization: In this work, our focus is on optimizing performance within the environment, and there might be a potential limitations when transitioning to entirely different scenarios. If we test it in a distinct environment, the performance may not be optimal, given the fine-tuning with data specific to the ScienceWorld environment. It's acknowledged that for generalization across diverse scenarios, an LLM may perform better, given its capacity to handle a broader range of inputs and contexts.

Goal Modification: When the agent encounters challenges in solving the current sub-goal, it will often find itself cycling through the same sub-goal for several steps. Consequently, the action generator repeats a sequence of actions mirroring recent ones. Sometimes the sub-goal generator will adjust the sub-goal slightly based on the input and that can be enough to get unstuck.

Ideally, we would like to avoid being stuck for several steps and learn to modify the sub-goal in the right way. One strategy involves online learning, where the controller is updated based on the reward from the environment. However, this approach carries the risk of catastrophic forgetting, necessitating additional measures such as loss modification and regularization to mitigate this risk. Another approach could involve incorporating an LLM alongside the controller. If the controller fails to produce effective actions, the LLM can suggest alternative sub-goals. This might have the risk of poor sub-goals and hallucinations which rewards might help but it is still challenging in such a sparse environment.

6 Conclusion

We introduce a straightforward yet highly effective approach for tackling complex text-based environments. Our framework leverages the knowledge of an LLM to extract sub-goals. A hierarchical policy of two LMs proposed: a high-level policy predicts a sub-goal, and a low-level policy, by using the predicted sub-goal, generates elementary actions. Through extensive experiments across 30 task types in ScienceWorld, our approach demonstrates increase performance compared to state-of-the-art baselines, including standard imitation learning and SwiftSage.

As future directions for this work, we aim to delve into further exploration of goal modification strategies when the agent encounters challenges in solving the current sub-goal. This could involve breaking down or transforming a sub-goal into a more achievable form. Another venue for future research involves extending this approach to a multi-module environment. In such scenarios, the sub-goal generator could leverage each module as an independent source to generate diverse and context-specific sub-goals. Exploring strategies for goal modification and online learning is another avenue we are keen to pursue.

Acknowledgments

Special thanks are due to Prasanna Parthasarathi for his invaluable insights, thoughtful brainstorming, and engaging discussions in the project. Also, thank you to Xingdi Yuan for initial discussions around knowledge distillation with LLM. Sarath Chandar is supported by the Canada CIFAR AI Chairs program, the Canada Research Chair in Lifelong Machine Learning, and the NSERC Discovery Grant. This research was enabled mostly by compute resources provided by Mila (mila.quebec) and partially by Microsoft.

References

Prithviraj Ammanabrolu and Matthew Hausknecht. Graph constrained reinforcement learning for natural language action spaces. In *International Conference on Learning Representations*, 2019.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2023.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. Pre-trained models: Past, present and future. *AI Open*, 2:225–250, 2021.

Matthew Hausknecht, Prithviraj Ammanabrolu, Côté Marc-Alexandre, and Yuan Xingdi. Interactive fiction games: A colossal adventure. *CoRR*, abs/1909.05398, 2019. URL http://arxiv.org/abs/1909.05398.

Ji He, Mari Ostendorf, Xiaodong He, Jianshu Chen, Jianfeng Gao, Lihong Li, and Li Deng. Deep reinforcement learning with a combinatorial action space for predicting popular reddit threads. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *arXiv preprint arXiv:2305.02301*, 2023.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pp. 9118–9147. PMLR, 2022a.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022b.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Prithviraj Ammanabrolu, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *arXiv preprint arXiv:2305.17390*, 2023.

Kyle Mahowald, Anna A Ivanova, Idan A Blank, Nancy Kanwisher, Joshua B Tenenbaum, and Evelina Fedorenko. Dissociating language and thought in large language models: a cognitive perspective. *arXiv preprint arXiv:2301.06627*, 2023.

Keerthiram Murugesan, Mattia Atzeni, Pavan Kapanipathi, Pushkar Shukla, Sadhana Kumaravel, Gerald Tesauro, Kartik Talamadupula, Mrinmaya Sachan, and Murray Campbell. Text-based RL Agents with Commonsense Knowledge: New Challenges, Environments and Baselines. In *Thirty Fifth AAAI Conference on Artificial Intelligence*, 2021.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

OpenAI. Openai: Introducing chatgpt, 2022.

OpenAI. Gpt-4 technical report, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

Roma Patel and Ellie Pavlick. Mapping language models to grounded conceptual spaces. In *International Conference on Learning Representations*, 2021.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models. *arXiv preprint arXiv:2311.05772*, 2023.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2010.03768.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11523–11530. IEEE, 2023.

Simeng Sun, Yang Liu, Shuohang Wang, Chenguang Zhu, and Mohit Iyyer. Pearl: Prompting large language models to plan and execute actions over long documents. *arXiv preprint arXiv:2305.14564*, 2023.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, 3(6):7, 2023.

Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 4950–4957, 2018.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*, 2022a.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022b.

Peter West, Chandra Bhagavatula, Jack Hessel, Jena D Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. Symbolic knowledge distillation: from general language models to commonsense models. *arXiv preprint arXiv:2110.07178*, 2021.

Shunyu Yao, Rohan Rao, Matthew Hausknecht, and Karthik Narasimhan. Keep calm and explore: Language models for action generation in text-based games. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8736–8754, 2020.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

A APPENDIX

A.1 FEW-SHOT PROMPT FOR THE LARGE LANGUAGE MODEL

We employed the ChatGPT API as the large language model in our study. The structure of the ChatGPT prompt is comprised of three main components. Firstly, there is a general description of the environment. The second part includes two examples, each containing the task description, an expert trajectory, and a set of sub-goals with their corresponding action sequences. Lastly, the prompt presents a new task description, along with an expert trajectory, then we ask the LLM to generate sub-goals for this new task.

Here is the first part:

Description of the Environment

You are a helpful assistant. You are in a simulated environment as an agent. A task and its description will be given to you. Suggest the best actions the agent can take based on the things you see and the items in your inventory to complete the task. Only use valid actions and objects. If you know what are around, then suggest the following actions. You are allowed to do the following actions with the objects. Open or close OBJ meaning open or close a container , Deactivate or activate OBJ meaning activate or deactivate a device, connect OBJ to OBJ meaning connect electrical components , disconnect OBJ meaning disconnect electrical components , use OBJ [on OBJ] meaning use a device/item , look around meaning describe the current room, look at OBJ meaning describe an object in detail, look in OBJ meaning describe a container's contents, read OBJ meaning read a note or book, move OBJ to OBJ meaning move an object to a container, pick up OBJ meaning move an object to the inventory, put down OBJ meaning drop an inventory item, pour OBJ into OBJ meaning pour a liquid into a container , dunk OBJ into OBJ meaning dunk a container into a liquid , mix OBJ meaning chemically mix a container , go to LOC meaning move to a new location , teleport to LOC meaning teleport to a specific room , eat OBJ meaning eat a food , flush OBJ meaning flush a toilet, focus on OBJ meaning signal intent on a task object, wait [DURATION] meaning take no action for some duration, task meaning describe current task, inventory meaning list agent's inventory, OBJ means objects. LOC means location. There are 10 locations centered around a house theme. These are: kitchen, bathroom, workshop, art studio, greenhouse, outside, bedroom, living room, foundry.

Here are the two examples for task $4-1$, which is *find-living-thing*. This is the second part of the prompt:

Two-shot examples of Task 4-1 (find-living-thing)

Example 1 [Task Description] Your task is to find a(n) living thing. First, focus on the thing. Then, move it to the red box in the kitchen.
[Expert trajectory] Here is the goal path to achieve to the goal: open door to greenhouse, go to greenhouse, open door to outside, go to outside, focus on dove, pick up dove, open door to kitchen, go to kitchen, move egg dove egg in inventory to red box
Based on the given goal path, provide me with the functional format of high-level sub-tasks to complete this task and their corresponding actions.
[sub-goals] 1- navigate_to(greenhouse): {'open door to greenhouse', 'go to greenhouse'} 2- navigate_to(outside): {'open door to outside', 'go to outside'} 3- Focus_on(dove): {'focus on dove'} 4- pick_up(dove): {'pick up dove'} 5- navigate_to(kitchen): {'open door to kitchen', 'go to kitchen'} 6- move(dove egg, red box): {'move dove egg in inventory to red box'}
Example 2 [Task Description] Your task is to find a(n) living thing. First, focus on the thing. Then, move it to the green box in the kitchen.
[Expert trajectory] Here is the goal path to achieve to the goal: open door to kitchen, go to kitchen, open door to outside, go to outside, focus on egg turtle, pick up egg turtle, open door to kitchen, go to kitchen, move egg turtle egg in inventory to green box
Based on the given goal path, provide me with the functional format of high-level sub-tasks to complete this task and their corresponding actions.
[sub-goals] 1- navigate_to(kitchen): {'open door to kitchen', 'go to kitchen'} 2- navigate_to(outside): {'open door to outside', 'go to outside'} 3- Focus_on(egg turtle): {'focus on egg turtle'} 4- pick_up(egg turtle): {'pick up egg turtle'} 5- navigate_to(kitchen): {'open door to kitchen', 'go to kitchen'} 6- move(egg turtle, green box): {'move egg turtle in inventory to green box'}

The third part is just a new task description with an expert trajectory:

Request for sub-goal generating

[Task Description] Your task is to find a(n) living thing. First, focus on the thing. Then, move it to the green box in the living room.
[Expert trajectory] Here is the goal path to achieve to the goal: open door to hallway, go to hallway, open door to greenhouse, go to greenhouse, open door to outside, go to outside, focus on baby baby beaver, pick up baby baby beaver, open door to greenhouse, go to greenhouse, open door to hallway, go to hallway, open door to living room, go to living room, move baby baby beaver in inventory to green box
Based on the given goal path, provide me with the functional format of high-level sub-tasks to complete this task and their corresponding actions.

A.2 Input Prompt for the Policies

Here, we show the inputs and outputs for both the action generator and sub-goal generator. We followed the format used by SwiftSage (Lin et al., 2023), incorporating sub-goals and making minor textual adjustments accordingly.

Input for the action generator

[Task Description] Your task is to find a(n) living thing. First, focus on the thing. Then, move it to the red box in the kitchen;
Time: 4; Score: 16; $< /s >$
Completed subtasks are: navigate_to(greenhouse), navigate_to(outside). The current subtask is Focus_on(dove);$< /s >$
Action history: $< extra\_id\_4 >$ look around (+0) –¿ N/A — $< extra\_id\_3 >$ go to greenhouse (+16) –¿ You move to the greenhouse. — $< extra\_id\_2 >$ open door to outside (+0) –¿ The door is already open. — $< extra\_id\_1 >$ go to outside (+0) –¿ You move to the outside. — $< /s >$
Current environment: This outside location is called the outside. Here you see: — the agent — a substance called air — an axe — a blue jay egg — a butterfly egg — a dove egg — a fire pit — a fountain (containing a substance called water) — the ground — a substance called wood — You also see: — A door to the foundry — A door to the greenhouse — A door to the kitchen — $< /s >$;
Current inventory: In your inventory, you see: — an orange — $< /s >$;
Visited rooms: hallway, greenhouse, outside $< /s >$;
What action should you do next? $< /s >$

Output for the action generator

focus on dove

Input for the sub-goal generator

[Task Description] Your task is to find a(n) living thing. First, focus on the thing. Then, move it to the red box in the kitchen;
Time: 4; Score: 16; $< /s >$
The previous subtasks are: navigate_to(greenhouse). The current subtask is navigate_to(outside) ;$< /s >$
Action history: $< extra\_id\_4 >$ look around (+0) –¿ N/A — $< extra\_id\_3 >$ go to greenhouse (+16) –¿ You move to the greenhouse. — $< extra\_id\_2 >$ open door to outside (+0) –¿ The door is already open. — $< extra\_id\_1 >$ go to outside (+0) –¿ You move to the outside. — $< /s >$
Current environment: This outside location is called the outside. Here you see: — the agent — a substance called air — an axe — a blue jay egg — a butterfly egg — a dove egg — a fire pit — a fountain (containing a substance called water) — the ground — a substance called wood — You also see: — A door to the foundry — A door to the greenhouse — A door to the kitchen — $< /s >$;
Current inventory: In your inventory, you see: — an orange — $< /s >$;
Visited rooms: hallway, greenhouse, outside $< /s >$;
What subtask should you do next? $< /s >$

Output for the sub-goal generator

Focus_on(dove)

Task Type	Topic	Task Name	# Variations
1-1	Matter	boil	30
1-2	Matter	melt	30
1-3	Matter	freeze	30
1-4	Matter	change-the-state-of-matter-of	30
2-1	Measurement	use-thermometer	540
2-2	Measurement	measure-melting-point-known-substance	436
2-3	Measurement	measure-melting-point-unknown-substance	300
3-1	Electricity	power-component	20
3-2	Electricity	power-component-renewable-vs-nonrenewable-energy	20
3-3	Electricity	test-conductivity	900
3-4	Electricity	test-conductivity-of-unknown-substances	600
4-1	Classification	find-living-thing	300
4-2	Classification	find-non-living-thing	300
4-3	Classification	find-plant	300
4-4	Classification	find-animal	300
5-1	Biology	grow-plant	126
5-2	Biology	grow-fruit	126
6-1	Chemistry	chemistry-mix	32
6-2	Chemistry	chemistry-mix-paint-secondary-color	36
6-3	Chemistry	chemistry-mix-paint-tertiary-color	36
7-1	Biology	lifespan-longest-lived	125
7-2	Biology	lifespan-shortest-lived	125
7-3	Biology	lifespan-longest-lived-then-shortest-lived	125
8-1	Biology	identify-life-stages-1	14
8-2	Biology	identify-life-stages-2	10
9-1	Forces	inclined-plane-determine-angle	168
9-2	Forces	inclined-plane-friction-named-surfaces	1386
9-3	Forces	inclined-plane-friction-unnamed-surfaces	162
10-1	Biology	mendelian-genetics-known-plant	120
10-2	Biology	mendelian-genetics-unknown-plant	480

Table 4: ScienceWorld's tasks names and numbers of variations. The highlighted rows show training task types for the generalization experiment.

A.3 Implementation Details

We set the maximum number of steps per episode to 100. Additionally, we implemented an alternative termination condition: if the scores remain unchanged over the last 50 steps, we stop the episode. This prevents the agent from getting stuck in a repetitive loop of actions that do not yield any changes, such as navigating between rooms. We selected this threshold to ensure that it is not too restrictive, considering that certain tasks with lengthy trajectories may require a long sequence of actions to achieve a reward.

The learning rate in all of the experiments is $1e-4$. The values for $max\_source\_length = 1024$ and for $max\_target\_length = 30$. The batch size for training is 8. We set the number of epochs to 20 during training. In the evaluation phase, checkpoints were selected based on their loss values, encompassing the checkpoint with the lowest loss and the subsequent three checkpoints. The final choice for test was determined among these checkpoints, with priority given to the one demonstrating the highest score in the test set.

For both the action generator and sub-goal generator, we used greedy decoding. When the generated actions is invalid we attempt to find the closes match from the list of admissible commands provided by ScienceWorld.

A.4 Dataset Statistics

In Table 4, we provide tasks' names and their variations for all of the tasks in ScienceWorld (Wang et al., 2022a). For each task, variations are partitioned into 50% training, 25% development, and 25% test sets. In the development and test sets, variations include substances, animals, or plants that are not seen in the training.

A.5 Scores of Each Task

The average score of each task of the ScienceWorld is shown in Table 5. The methods are SayCan, ReAct, Reflexion, Swift-only, SwiftSage and our algorithm. In all of them ChatGPT is used as the LLM. The language models for Swift-only, SwiftSage and our algorithm are FLAN-T5-LARGE. For the methods, SayCan, ReAct, Reflexion, and SwiftSage we used the results from Lin et al. (2023). However, we reproduced the results for Swift-only and we found a lower performance than what was reported in the paper. So here we presented our scores.

The scores for SayCan, ReAct, Reflexion, and SwiftSage when they use GPT4 as the LLM are higher according to the results reported in Lin et al. (2023). However, due to limited access to GPT-4, we utilized ChatGPT, and thus, we present the results obtained using ChatGPT.

Task Type	Length	SayCan	ReAct	Reflexion	Swift-only	SwiftSage	Ours
1-1	107.7	0	3.52	4.22	15.0	58.0	16.22
1-2	78.6	0	13.70	10.61	24.4	58.5	0.2
1-3	88.9	0	7.78	7.78	32.2	38.5	30.33
1-4	75.2	0	9.88	0.92	57.4	62.5	65.0
2-1	21.4	1.7	7.19	5.92	9.4	47.9	98.55
2-2	35.2	14.1	6.10	28.59	6.7	53.3	46.0
2-3	65.0	93.7	22.37	22.37	5.7	48.6	45.0
3-1	13.6	19.3	56.0	100.0	70.0	72.7	100
3-2	20.8	8.7	54.33	17.45	48.3	50.3	78.2
3-3	25.6	22.0	76.19	72.54	59.5	66.9	5.6
3-4	29.0	36.4	36.4	70.22	69.0	78.1	46.0
4-1	14.6	11.7	26.67	64.93	100.0	100.0	100
4-2	8.8	76.0	80.0	87.27	100.0	97.5	100
4-3	12.6	11.4	53.33	16.42	94.4	58.3	100
4-4	14.6	9.5	27.50	100.0	100.0	100.0	100
5-1	69.5	11.3	11.1	5.8	13.4	57.5	100
5-2	79.6	75.0	18.8	47.6	44.6	50.9	65
6-1	33.6	13.5	35.0	22.4	26.2	43.2	66.87
6-2	15.1	25.0	20.0	10.0	53.3	63.3	100
6-3	23.0	58.4	16.7	40.0	11.1	27.4	100
7-1	7.0	75.0	37.5	75.0	83.3	75.0	100
7-2	7.0	100.0	50.0	75.0	100.0	60.0	100
7-3	8.0	31.7	31.7	28.1	77.8	68.3	100
8-1	40.0	5.6	4.2	2.8	33.0	75.6	60.0
8-2	16.3	12.8	7.0	8.2	8.0	33.0	25.0
9-1	97.0	38.0	28.5	100.0	73.3	54.0	52.5
9-2	84.9	4.2	10.0	17.5	73.3	63.3	53.6
9-3	123.1	0.0	0.0	1.7	53.3	77.0	64.4
10-1	130.1	1.3	24.5	1.3	17.0	76.0	30.76
10-2	132.1	0.3	11.7	6.0	17.0	51.1	30.76
Average	49.26	25.22	19.76	23.40	49.22	62.22	65.43

Table 5: The table illustrates the results for each task of the ScienceWorld for SayCan, ReAct, Reflexion, Swift-only, SwiftSage and our algorithm. Each row shows the average score of the test variations for a task type. Column *Length* shows the average lengths of the expert trajectories.

A.6 Task Scores Across Various Model Sizes

In Table 6, the average scores for models of different sizes are presented. In this experiment, the *base* model was employed for the action generator, while the size of the sub-goal generator was varied across small, base, large, and x-large. The first four columns of the table display their respective scores. Additionally, using the *small* model for the action generator, the experiment was repeated with various sizes of the sub-goal generator, and the results are shown in the second set of four columns. The last column illustrates the performance when both the action generator and the sub-goal generator are x-large, achieving the highest score.

Action generator	Base				Small				X-Large
Sg generator / Tasks	Small	Base	Large	X-Large	Small	Base	Large	X-Large	X-Large
1-1	0.2	0.0	0.0	4.2	0.0	0.0	0.0	0.0	16.55
1-2	0.0	0.0	0.0	0.22	0.0	0.0	0.44	0.22	17.11
1-3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1-4	0.1	0.0	0.0	7.77	0.33	0.0	7.77	19.44	20.44
2-1	10.0	20.0	100	90.0	30.0	10.0	80.0	70.0	100
2-2	13.2	20.0	20.0	77.5	0.0	20.0	20.0	65.7	77.5
2-3	0.0	20.0	0.0	64.0	0.0	20.0	20.0	60.0	45.0
3-1	0.0	78.0	84.0	79.4	0.0	48.0	77.8	71.2	100
3-2	10.6	36.4	47.0	77.2	0.0	27.8	49.0	56.4	77.2
3-3	26.0	20.0	22.0	6.4	7.3	10.5	3.5	1.5	28.0
3-4	40.0	30.0	40.0	30.0	50.31	30.0	60.0	30.0	47.05
4-1	100	100	100	100	100	100	100	90	100
4-2	100	100	100	100	100	100	100	100	100
4-3	100	100	92.5	93	100	100	92.5	94.0	100
4-4	100	100	100	100	90.9	100	100	100	100
5-1	66.2	56.18	100	72.7	82.8	67.1	100	72.7	100
5-2	31.8	7.45	39.6	69.4	35.8	0.0	47.1	45.7	70.0
6-1	3.12	31.25	43.62	36.62	12.5	28.12	38.75	34.62	56.25
6-2	20.0	10.0	66.66	74.44	18.88	32.22	33.33	35.55	91.11
6-3	23.0	13.33	33.66	52.33	1.1	6.33	18.22	17.88	54.88
7-1	80.0	100	100	100	90.9	100	100	100	100
7-2	20.0	78.12	80.0	100	40.0	50.0	50.0	60.0	100
7-3	100	100	100	100	70.0	70.0	70.0	70.0	100
8-1	11.6	18.4	60.0	40.0	0.0	18.4	60.0	40.0	60.0
8-2	0.0	0	0.0	42.5	0.0	0.0	0.0	25.0	67.5
9-1	28.0	35.65	24	70	42.0	44.0	33.0	74.0	84.0
9-2	26.0	32.6	23	65	35.6	44.0	32.0	70.0	82.0
9-3	27.0	29.89	29	66	30.0	43.0	31.0	69.0	83.0
10-1	27.0	26.9	30.1	28.2	24.9	25.2	25.2	25.2	28.2
10-2	30.2	27.75	30.4	27.7	23.0	19.5	29.9	18.9	30.1
Average	33.13	39.73	48.85	59.15	32.87	37.13	45.98	50.56	67.86

Table 6: The table displays the average scores for the models of different sizes. The first four columns depict scores when a model with a base size is used as the action generator. The subsequent four columns illustrate scores when the action generator is small size. The last column shows the results when both the action generator and sub-goal generator are x-large.

A.7 Scores for Different Model Sizes Without sub-goals

Table 7 presents task scores without the utilization of sub-goals, akin to the Swift-only method but employing language models of varying sizes. All models are FLAN-T5.

Task Type	Small	Base	Large	X-Large
1-1	0.4	0.0	14.0	14.0
1-2	11.8	0.0	22.88	22.0
1-3	0.0	0.0	24.55	20.0
1-4	7.6	0.0	0.88	0.88
2-1	40.0	80.0	70.32	91.66
2-2	43.2	45.6	6.56	45.82
2-3	40.0	43.0	5.76	41.0
3-1	59.8	85.2	85.2	74
3-2	41.4	51.0	41.1	43.4
3-3	19.2	9.4	64.88	33.29
3-4	50.0	33.0	81.26	58.75
4-1	100	95.0	100	100
4-2	100	97.7	100	100
4-3	80.0	92.0	80.0	81.25
4-4	100	95.0	100	100
5-1	64.2	64.4	24.0	60.42
5-2	34.5	35.0	43.57	42.53
6-1	12.75	29.62	26.0	58.12
6-2	0.0	11.11	36.0	34.11
6-3	3.6	0.0	11.33	7.77
7-1	90.9	100	100	100
7-2	40.0	100	100	100
7-3	74.7	78.34	87.6	100
8-1	8.2	35.2	46.0	34.4
8-2	0.0	0.0	8.0	41.0
9-1	20.0	20.0	40.7	40.0
9-2	19.0	20.0	43.6	43.0
9-3	20.0	20.0	44.52	44.0
10-1	22.8	23.0	19.66	23.0
10-2	22.9	22.9	20.22	23.0
Average	37.56	42.88	46.25	52.88

Table 7: Scores for each task without the utilization of sub-goals across various model sizes. All models are FLAN-T5.

A.8 Results for T5 Language Model With and Without sub-goals

The results for the T5 model, including T5-LARGE and T5-3B, are presented in Table 8. The outcomes are shown for both scenarios—with and without the integration of sub-goals. In the experiments where sub-goals were employed, equivalent sizes were utilized for both the action generator and sub-goal generator.

A.9 Results for random and semi-random sub-goals

We generated random sub-goals from the sub-goal space and used that instead of the sub-goal generator. The action generator remains the fine-tuned FLAN-T5-LARGE, as before. The results are displayed in Table 9.

To further investigate the impact of the sub-goals, we conducted another experiment where we retained the sub-goals and solely modified their arguments, typically are objects or locations. Despite this adjustment, performance remained low, although slightly improved compared to random sub-goals.

	No sub-goals		With sub-goals	
Task Type	Large	X-large	Large	X-large
1-1	16.66	0.2	0.55	18.0
1-2	0.0	56.0	5	19.66
1-3	0.0	8.55	4.77	0.0
1-4	0.0	30.77	16.22	38.0
2-1	9.1	91.66	70.0	70.0
2-2	37.8	41.0	20.0	27.7
2-3	10.0	30.0	34.0	44.0
3-1	85.2	78.6	39.2	48.6
3-2	46.0	55.2	64.8	30.6
3-3	50.2	14.61	1.0	31.76
3-4	30.0	35.71	40.0	50.0
4-1	100	100	100	100
4-2	100	100	100	100
4-3	90.0	93.7	100	100
4-4	100	100	100	74.33
5-1	100	64.2	91.5	100
5-2	8.9	37.8	2.1	100
6-1	41.62	43.62	21.87	31.25
6-2	15.55	100	55.55	73.33
6-3	4.77	80.0	29.2	35.0
7-1	90.0	100	100	100
7-2	100	100	30.0	100
7-3	80.0	60.0	100	100
8-1	55.4	35.2	60.0	69.0
8-2	0.0	0.0	0.0	0.0
9-1	20.0	20.0	36.0	33.33
9-2	21.0	20.0	34.2	36.1
9-3	22.0	20.0	37.1	39.0
10-1	27.9	23.0	29.69	30.4
10-2	28.0	22.9	28.75	28.0
Average	43.00	52.09	45.05	54.26

Table 8: Scores for the T5 model are depicted under two conditions: without sub-goals and with sub-goals.

A.10 Sub-goal Retrieving

Here are the scores representing the ability to retrieve sub-goals for the sub-goal generators. There are two sets of the experiments: 1- Random/semi-random first sub-goals, 2-Random/semi-random sub-goals during interactions. The scores for each task and experiment are presented in Table 10.

A.11 Generalization experiment

In this experiment, we trained the model on a subset of tasks and evaluated it on test variations from all tasks. For each scientific topic, we selected one or two tasks for training and reserved the remaining tasks for evaluation. We compared our algorithm against the Swift-only baseline. The scores for each task are displayed in Table 11.

Task Type	Random Sub-goal	Semi-random Sub-goal
1-1	0	0.5
1-2	0	1
1-3	0	2.22
1-4	0	0.6
2-1	0	3
2-2	0	3.1
2-3	0	2
3-1	0	2.8
3-2	0	4
3-3	5.5	8
3-4	0	1
4-1	0	0
4-2	58.3	81.4
4-3	0	23.4
4-4	0	2.5
5-1	6.6	10.4
5-2	21.9	60
6-1	0	21
6-2	15.55	23.33
6-3	6.44	12
7-1	10	60
7-2	30	92
7-3	0	0
8-1	0	0
8-2	0	0
9-1	0	5.5
9-2	0	5
9-3	0	4
10-1	0	14
10-2	0	20
Average	6.413	14.15

Table 9: Scores for the random and semi-random sub-goals with fine-tuned FLAN-T5-LARGE action generator.

Task Type	Semi-random First Sub-goal	Random First Sub-goal	Semi-random Sub-goal at 10 steps	Random Sub-goal at 10 steps
1-1	36.2	33	8.22	8.8
1-2	19.33	20.11	31.44	30.55
1-3	20.77	28.22	33	25.4
1-4	73	26.44	26.88	16
2-1	40	31.2	28.4	48.4
2-2	16.4	17.2	12.2	28.1
2-3	17	18	12	28
3-1	78	32.8	78	50.6
3-2	78.2	28.6	56.2	62.4
3-3	6	6.5	13.4	10
3-4	12	50.5	0	36.8
4-1	100	70.8	92.5	90
4-2	100	82.5	100	97.3
4-3	80	57.5	85.5	81.5
4-4	100	79.8	97.4	60
5-1	100	51.7	53.1	33
5-2	41	23.8	31.8	5
6-1	31.65	42.5	26	24.25
6-2	75.55	77.77	54.44	71.11
6-3	52.88	44.77	32.88	17.6
7-1	100	80	100	70
7-2	100	90	100	80
7-3	80	70	70	50
8-1	0	20	20	0
8-2	25	0	0	0
9-1	52	15	27	31
9-2	50	14	26	29
9-3	51	16	25	28
10-1	27.5	20.9	29.5	1.8
10-2	29.6	24.6	29	13.6
Average	53.10	39.14	43.33	37.61

Table 10: Scores for retrieving sub-goals are displayed. The first two columns depict the random and semi-random generation of the first sub-goals only. The second pair of columns illustrates the scores obtained when sub-goals are generated randomly or semi-randomly every 10 steps.

Task Type	With sub-goal	Swift-only
1-1	6	5
1-2	1.3	0
1-3	2.2	0
1-4	9.33	74.33
2-1	38.9	100
2-2	20	33.9
2-3	20	0
3-1	35.4	100
3-2	14.6	0
3-3	16.9	19.6
3-4	20	15.5
4-1	100	100
4-2	95	100
4-3	39.2	0
4-4	65	53
5-1	90.7	81.8
5-2	50	2
6-1	69.75	23.75
6-2	62.22	30
6-3	38.77	2.5
7-1	100	100
7-2	100	100
7-3	10	16.6
8-1	40	64.4
8-2	0	0
9-1	44.5	20
9-2	40	19
9-3	41	5
10-1	20.6	28.6
10-2	28.7	1.9
Average score	40.63	36.56
Average score seen tasks	50.51	52.85
Average score unseen tasks	27.72	15.25

Table 11: The table displays the scores for the generalization experiment. The highlighted tasks in yellow called "seen tasks" which are the ones selected for the training.