
SkillAct: Using Skill Abstractions Improves LLM Agents

Anthony Z. Liu^{*1} Jongwook Choi^{*1} Sungryull Sohn² Yao Fu¹ Jaekyom Kim² Dong-ki Kim² Xinhe Wang¹
Jaewon Yu¹ Honglak Lee^{1,2}

Abstract

Complex sequential decision-making tasks often require hierarchical thinking and abstraction: breaking down these tasks into simpler subtasks that can be solved with reusable behaviors, or *skills*. In this work, we show that large language models (LLMs) can benefit from using skill abstractions to solve interactive tasks successfully. We propose a simple prompting approach named *SkillAct*, which can extend existing prompting approaches. In addition, we demonstrate that these skill abstractions can be *learned* from few-shot demonstrations by prompting LLMs. We demonstrate that *SkillAct* improves the performance of existing approaches such as ReAct on the interactive task benchmark ALFWorld.

1. Introduction

Humans solve complex tasks by breaking them down into simpler tasks, then decide how to finish these simpler tasks by *planning*. Often these simpler tasks appear in many different tasks. Cognitively, humans benefit from learning abstractions for shared tasks, which we refer to as *skills* in this work. For example, consider a set of household tasks, such as “*clean the dirty dishes*”, or “*serve coffee*”. In these tasks, an agent will need find and retrieve some target objects (dishes or mug) in the scene. A skill such as “finding a target object” would be useful, as it can be shared by many different tasks.

Automated approaches that use large language models (LLMs) to solve these complex interactive tasks (LLM Agents) have shown strong capabilities in multiple domains (Brohan et al., 2023; Nakano et al., 2021; Huang et al., 2022). In these tasks, agents leverage LLM’s strong domain knowledge, and its planning abilities by prompting LLMs to enumerate high-level plans on how to solve the

¹University of Michigan, Ann Arbor, USA ²LG AI Research, Ann Arbor, USA. Correspondence to: Anthony Liu <anth-liu@umich.edu>.

task (e.g., prompting “*What steps should I take to clean the dirty dishes?*”).

Recently, many prompt-based agents using LLMs have emerged, further improving task solving capabilities (Yao et al., 2022; Wang et al., 2023; Yin et al., 2023; Shinn et al., 2024; Niu et al., 2024). Notably, ReAct (Yao et al., 2022) prompts LLMs to “think” while solving tasks, enabling the agent to reason about the tasks given, track the agent’s progress, and update the current plan as necessary. Voyager (Wang et al., 2023) prompts LLMs to produce and learn a *skill library*. In Voyager, as the task is to solve a code-based environment, skills are represented in the form of executable code functions (in Python) that are updated and re-used as the agent explores the environment.

Our work takes a more general view of skills than these prior works do. Compared to Voyager, skills can also be applied to non-code settings by encoding skill information in natural language — we simply define skills with skill name, instructions, and examples of execution (Figure 1). This can be seen as a more relaxed, loose version than code skills as in Voyager, allowing more flexibility to be utilized by LLMs and the ease of computation without needing to execute through an interpreter that requires strict syntactic and semantic correctness for code. Furthermore, we show that by simply appending appropriate skill descriptions to an LLM agent prompt, such as *SkillAct*, we can increase the performance of the LLM agent. As this approach simply modifies the LLM prompt, it can be used to extend other LLM agents for other domains. In addition, these skills can be “learned” from data: LLMs can produce useful skills by analyzing few-shot demonstrations of the target tasks.

Our work makes the following contributions:

- We show that prompting LLMs with *skills*, or descriptions of shared behaviors that are useful solving tasks, can improve performance on interactive text-based planning and reasoning tasks such as ALFWorld (Shridhar et al., 2020b).
- We present an approach for extracting skills from demonstrations without expert supervision, and show these are comparable to skills written by human experts.

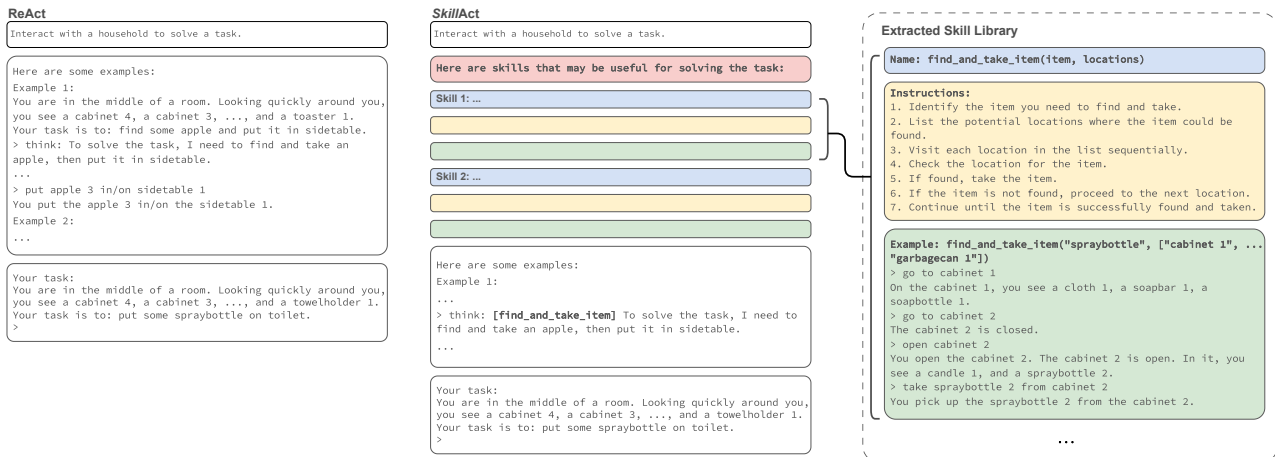


Figure 1. A comparison of prompting approaches SkillAct vs. ReAct on the ALFWorld domain. SkillAct extends existing prompt approaches by inserting descriptions of *skills* into the prompt. Skills help LLMs solve tasks by providing useful abstractions over useful behaviors. The skill shown in the figure was written by GPT4-turbo (Achiam et al., 2023) using the provided demonstrations.

2. SkillAct

Our approach, *SkillAct*, is a simple method which provides a large language model (LLM) agent with descriptions of *skills*: abstractions over useful and reusable behaviors for the target tasks. In the following, we (§2.1) define skills, (§2.2) describe how we can prompt LLMs with skills, and (§2.3) describe how to extract skills from demonstrations.

2.1. Skills

Consider a decision-making problem in a language domain, where at time step t , given an observation o_t from the environment, the agent must predict the next action a_t toward its final goal. A sequence of the agent’s observations and actions, $\tau = (o_1, a_1, \dots, o_N, a_N)$, is called a *trajectory*.

We designate “skills” to be *abstractions* over partitions of agent trajectories that (1) summarize high-level actions that an agent should take, consisting of multiple actions from the trajectory: $(o_i, a_i, \dots, o_{i+\ell}, a_{i+\ell})$, and (2) can be reused in multiple different tasks. Skills are also often referred to as “options” in the options framework (Sutton et al., 1999).

For instance, in ALFWorld, agents are given household tasks that require many steps, context understanding, and reasoning to solve, such as “*find some apple and put it in sidetable*”. However, these tasks also share multiple potential “skills”. For instance, one crucial skill present in every ALFWorld task is the need to find objects within the household. An agent that could consistently use a “find” skill could hence more consistently solve ALFWorld tasks.

2.2. Prompting LLMs with Skills

In this work, we represent skills in a textual form so that it can easily guide an LLM agent. We define each skill as a tuple $\mathbf{z} = (\phi, I, e)$, where ϕ is the name of the skill, I is the

instructions for the skill, and e is the skill examples, a list of trajectory segments where the skill is used. We present an example of skills in the third column of Figure 1. The name ϕ and instructions I of the skill can guide the LLM on how to use and generalize the given skill for a given task. The examples e serve as few-shot demonstrations to the LLM on how to execute the provided skills in the task.

We maintain a set of extracted skills as a *skill library* $\{\mathbf{z}_1, \dots, \mathbf{z}_k\} = \{(\phi_1, I_1, e_1), \dots, (\phi_k, I_k, e_k)\}$, which can be incorporated into the prompt for an LLM policy, as illustrated in the second column of Figure 1: We call the resulting prompt-based LLM agent as *SkillAct*, which is provided with the available skill library and a few additional examples of using skills to solve some tasks. This contrasts with simple prompt-based LLM agents, such as ReAct (Yao et al., 2022), which are only provided with few-shot demonstrations without any skill abstractions.

2.3. Extracting Skills from Few-Shot Demonstrations

To make the incorporation of the skill library effective, having a good set of skills is important. Due to the large-scale training data in diverse domains, LLMs often possess *common-sense* knowledge to form skills that can be helpful in solving decision-making tasks, such as the household tasks from ALFWorld. However, to leverage such ingrained skills for the completion of tasks, these skills should be “grounded” to the task (e.g., specific actions and contexts are required in tasks such as ALFWorld). A limited form of grounding can be done by showing an LLM few-shot demonstrations or expert trajectories, which are assumed to be given in past LLM agent works (Yao et al., 2022; Brohan et al., 2023; Huang et al., 2022). This grounding is limited as the trajectories will differ from the inference task, and the agent will need to adapt skills during inference.

In this work, we show that skills can be automatically “ex-

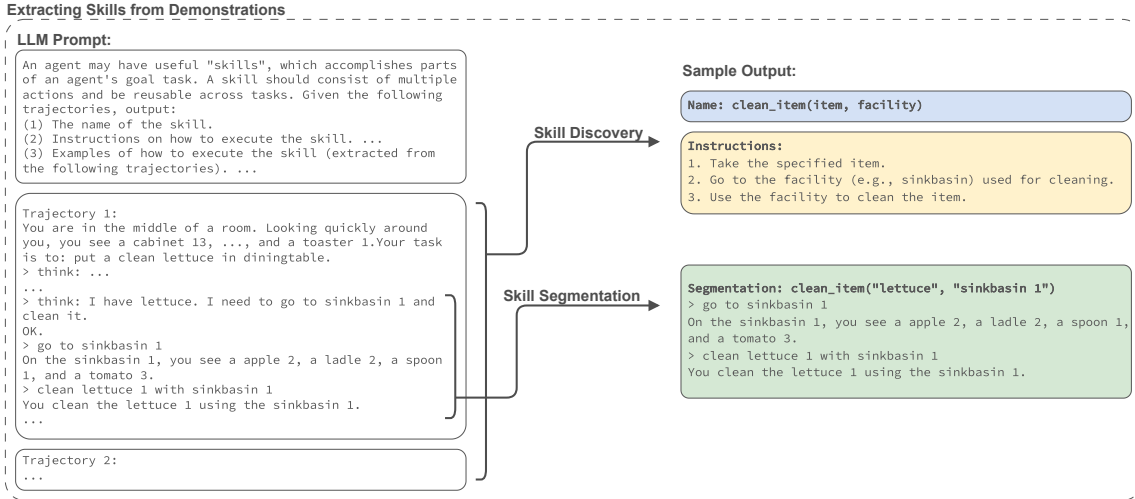


Figure 2. Skills can be extracted or “learned” using LLMs through prompting. We describe our process for extracting skills in Section 2.3. Through LLM’s prior domain knowledge and reading comprehension ability, they are able to extract skills that are similar to the expert skills without supervision (skills shown in Appendices B and C).

tracted” from demonstrations simply with a prompt-based approach, as in Figure 2. In the prompt for the LLM, we provide the definition of *skills* and instruct the LLM to output skills in the specified format: name, instruction, and example trajectory, i.e., (ϕ, I, e) . We include the prompts we used in Appendix D.

Finally, in order to make sure the LLM to make use of such skills, we “relabel” some steps in the demonstration to reference the skills using an LLM. The prompt for this process is described in Appendix E.

3. Experiments

3.1. Benchmark

ALFWorld. We evaluate on the ALFWorld environment (Shridhar et al., 2020b), a text-based simulated household environment, designed around the embodied agent ALFRED environment (Shridhar et al., 2020a). In this environment, the agent must find and manipulate objects in a household according to a given task, e.g., “*find some apple and put it in the sidetable*” as shown in Figure 1. ALFWorld is a difficult environment, as the agent needs to achieve various subgoals within the tasks by planning. However, across different tasks many subgoals may be shared — where learning shared skills can help to solve.

In our experiments, we use the same annotated randomly sampled trajectories as ReAct (Yao et al., 2022): 2 trajectories for each of the 6 task types. These trajectories contain “think” actions annotated by the ReAct authors.

3.2. Methods

Baseline. We test our approach, SkillAct, against a baseline ReAct (Yao et al., 2022). ReAct prompts the LLM to additionally output “think” steps, which help the LLM reason and better plan its actions. Our approach (SkillAct) also contains a similar think prompts, and can be thought of adding skills to ReAct.

SkillAct. We test two versions of SkillAct — (1) **SkillAct**: skills are extracted from the ReAct-annotated trajectories using GPT4-turbo, and (2) **SkillAct-Expert**: skills are written by an expert (human) annotator. The skill examples are also taken from the ReAct annotated trajectories. In both versions, we also use GPT4-turbo to relabel the ReAct trajectories to include reference to the provided skills.

3.3. Results

Skill Extraction Quality. We show an example of an extracted skill in Figure 2, and all extracted skills using GPT4-turbo¹ in Appendix B. We find that the extracted skills are similar to the expert skills, even more general than the expert skills (e.g. suggesting `clean_item(item, facility)`, for general cleaning facilities compared to the expert’s skill, `clean(object)`, which assumes the sink 1 is always the cleaning facility). The extracted skills also tended to be more verbose, likely a result of the prompt instructing the LLM to describe skills in a “step by step” manner.

ALFWorld: Success Rate. We show the success rate (SR)

¹We also tried extracting skills with weaker models, such as GPT3.5-turbo. However, we found they had poor output quality; often producing skills that are overly vague, or producing just single trivial skill.

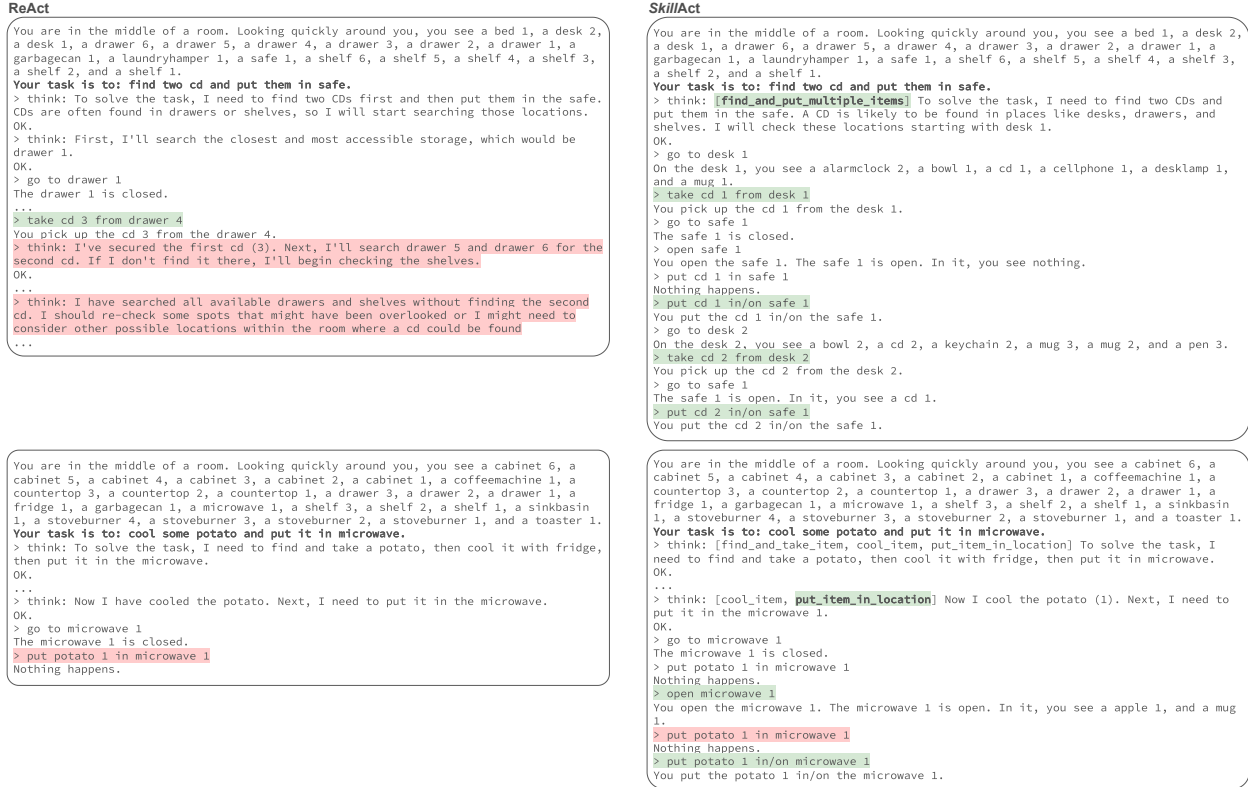


Figure 3. Sampled trajectories of two tasks with SkillAct with extracted skills vs. ReAct on ALFWorld. In these trajectories, ReAct makes errors specific to the ALFWorld environment (1) trying to take multiple objects at once and failing to find them and (2) using the incorrect action syntax. Skills help correct these errors by re-emphasizing how to correctly act through instructions and examples.

LLM	AGENT	ALFWORLD SR (\uparrow %)
GPT3.5-TURBO	REACT	54.5%
	SKILLACT	54.9%
	SKILLACT-EXPERT	59.0%
GPT4-TURBO	REACT	79.7%
	SKILLACT	85.1%
	SKILLACT-EXPERT	88.0%

Table 1. We evaluate each approach on ALFWorld with different base language models GPT3.5-turbo-instruct and GPT4-turbo (Achiam et al., 2023).

of each base LLM and approach in Table 1. For each LLM, SkillAct with expert skills achieves the best SR (59% and 88% SR for GPT3.5-turbo and GPT4-turbo respectively).

With extracted skills, SkillAct still achieves a success rate higher than ReAct (an increase of +0.4% and +5.4% SR). We hypothesize the relatively lower performance increase in GPT-3.5-turbo may be due to the LLM’s weaker ability to comprehend and utilize the skills.

ALFWorld: Qualitative Analysis. We analyzed ReAct compared with SkillAct (extracted skills) by sampling trajectories where ReAct failed and SkillAct succeeded. We visualize two examples from these trajectories in Figure 3.

From our analysis, we see that both ReAct and SkillAct

rarely make mistakes in planning: deciding which skills or subgoals to pursue to accomplish the task. However, ReAct makes more “plan execution” mistakes, mistakes when outputting actions to execute the plan. In Figure 3, we see that ReAct makes a mistake in trying to take multiple objects at once (ALFWorld agents can only hold one at a time), failing to find the objects, and outputting the wrong action syntax for the put action.

We also analyzed the trajectories of SkillAct with extracted vs. expert skills, shown in Figure 4. We find that extracted skills tend to cause the LLM to make more plan execution mistakes. We hypothesize that these may be due to the verbosity of extracted skills obscuring some important details that an LLM may need to solve the task.

4. Conclusion

In this work we showed how *skills* — descriptions of reusable behaviors in tasks — can improve an LLM agent’s performance in the ALFWorld environment by simply adding them to other LLM agent prompts. We also demonstrated how to extract and learn these skills through LLM prompting. We find the quality of these skills that are automatically extracted by LLMs can match the ones written by human experts, and result in similar performance boost on the ALFWorld domain.

References

- Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., Ho, D., Ibarz, J., Irpan, A., Jang, E., Julian, R., 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.
- Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pp. 9118–9147. PMLR, 2022.
- Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
- Niu, R., Li, J., Wang, S., Fu, Y., Hu, X., Leng, X., Kong, H., Chang, Y., and Wang, Q. Screenagent: A vision language model-driven computer control agent. *arXiv preprint arXiv:2402.07945*, 2024.
- Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.
- Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L., and Fox, D. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10740–10749, 2020a.
- Shridhar, M., Yuan, X., Côté, M.-A., Bisk, Y., Trischler, A., and Hausknecht, M. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020b.
- Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.
- Wang, G., Xie, Y., Jiang, Y., Mandlkar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.
- Yin, D., Brahman, F., Ravichander, A., Chandu, K., Chang, K.-W., Choi, Y., and Lin, B. Y. Lumos: Learning agents with unified data, modular design, and open-source llms. *arXiv preprint arXiv:2311.05657*, 2023.

Appendix

A. Additional ALFWorld Qualitative Analysis

SkillAct Extracted Skills

You are in the middle of a room. Looking quickly around you, you see a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 3, a countertop 2, a countertop 1, a drawer 3, a drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a shelf 3, a shelf 2, a shelf 1, a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1. **Your task is to: heat some apple and put it in garbagecan.**

> To solve the task, I need to find and take an apple, then heat it with the microwave, and finally put it in the garbagecan.

Nothing happens.

> think: [find_and_take_item] First I need to find an apple. An apple is more likely to appear in **fridge (1), countertop (1-3), or drawer (1-3)**. I can check these locations one by one, starting with fridge 1.

OK.

> go to fridge 1

The fridge 1 is closed.

> open fridge 1

You open the fridge 1. The fridge 1 is open. In it, you see a bowl 1, a bread 1, a cup 1, a lettuce 2, and a potato 1.

> go to countertop 1

On the countertop 1, you see a cellphone 1, a mug 2, a pan 2, and a tomato 1.

...

> It appears that an apple is not found in any of the inspected locations. Please let me know how you would like to proceed or if there is any other location you would like me to check further.

Nothing happens.

SkillAct Expert Skills

You are in the middle of a room. Looking quickly around you, you see a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 3, a countertop 2, a countertop 1, a drawer 3, a drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a shelf 3, a shelf 2, a shelf 1, a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1. **Your task is to: heat some apple and put it in garbagecan.**

> think: [find, heat, place] To solve the task, I need to find and take an apple, then heat it with microwave, then put it in garbagecan.

OK.

> think: [find] First I need to find an apple. An apple is more likely to appear in **fridge (1), countertop (1-3), drawer (1-3), cabinet (1-6), garbagecan (1)**. I can check one by one, starting with fridge 1.

OK.

> go to fridge 1

The fridge 1 is closed.

...

> open cabinet 6

You open the cabinet 6. The cabinet 6 is open. In it, you see nothing.

> go to garbagecan 1

On the garbagecan 1, you see a apple 2, and a apple 1.

> think: [find, take] Now I find an apple (1). Next, I need to take it.

OK.

> take apple 1 from garbagecan 1

You pick up the apple 1 from the garbagecan 1.

Figure 4. A trajectory of SkillAct with *extracted* skills vs. *expert* skills on the same task. In the trajectories, SkillAct with extracted skills (left) fails to enumerate through all possible object locations.

B. ALFWorld Extracted Skills

```

find_and_take_item(item, locations)
instructions:
1. Identify the item you need to find and take.
2. List the potential locations where the item could be found.
3. Visit each location in the list sequentially.
4. Check the location for the item.
5. If found, take the item.
6. If the item is not found, proceed to the next location.
7. Continue until the item is successfully found and taken.
example: find_and_take_item("spraybottle", ["cabinet 1", "cabinet 2", "cabinet 3", "
cabinet 4", "countertop 1", "toilet 1", "sinkbasin 1", "sinkbasin 2", "garbagecan 1"
])
> go to cabinet 1
On the cabinet 1, you see a cloth 1, a soapbar 1, a soapbottle 1.
> go to cabinet 2
The cabinet 2 is closed.
> open cabinet 2
You open the cabinet 2. The cabinet 2 is open. In it, you see a candle 1, and a
spraybottle 2.
> take spraybottle 2 from cabinet 2
You pick up the spraybottle 2 from the cabinet 2.

put_item_in_location(item, location)
instructions:
1. Identify the item and the desired location.
2. Carry the item to the specified location.
3. Place the item in/on the location.
example: put_item_in_location("spraybottle", "toilet 1")
> go to toilet 1
On the toilet 1, you see a soapbottle 2.
> put spraybottle 2 in/on toilet 1
You put the spraybottle 2 in/on the toilet 1.

clean_item(item, facility)
instructions:

```

```
1. Take the specified item.
2. Go to the facility (e.g., sinkbasin) used for cleaning.
3. Use the facility to clean the item.
example: clean_item("lettuce", "sinkbasin 1")
> go to sinkbasin 1
On the sinkbasin 1, you see a apple 2, a ladle 2, a spoon 1, and a tomato 3.
> clean lettuce 1 with sinkbasin 1
You clean the lettuce 1 using the sinkbasin 1.

heat_item(item, appliance)
instructions:
1. Take the specified item.
2. Go to the relevant appliance (e.g., microwave) used for heating.
3. Use the appliance to heat the item.
example: heat_item("egg", "microwave 1")
> go to microwave 1
The microwave 1 is closed.
> heat egg 2 with microwave 1
You heat the egg 2 using the microwave 1.

cool_item(item, appliance)
instructions:
1. Take the specified item.
2. Go to the relevant appliance (e.g., fridge) used for cooling.
3. Use the appliance to cool the item.
example: cool_item("pan", "fridge 1")
> go to fridge 1
The fridge 1 is closed.
> cool pan 1 with fridge 1
You cool the pan 1 using the fridge 1.

find_and_use_item(item, locations)
instructions:
1. Identify the item you need to find and use.
2. List the potential locations where the item could be found.
3. Visit each location in the list sequentially.
4. Check the location for the item.
5. If found, use the item.
6. If the item is not found, proceed to the next location.
7. Continue until the item is successfully found and used.
example: find_and_use_item("desk lamp", ["desk 1", "sidetable 1", "sidetable 2"])
> go to desk 1
On the desk 1, you see a alarmclock 2, a book 1, a cellphone 1, a keychain 1, a laptop 2,
a laptop 1, and a pen 3.
> go to sidetable 1
On the sidetable 1, you see a cd 1, a pen 1, and a pencil 1.
> go to sidetable 2
On the sidetable 2, you see a alarmclock 1, a desk lamp 1, and a pen 2.
> use desk lamp 1
You turn on the desk lamp 1.

find_and_put_multiple_items(items, locations, container)
instructions:
1. Identify the items and the locations where they might be found.
2. Visit each location sequentially to find each item.
3. Take each item once found.
4. Carry each item to the specified container.
5. Place the items in/on the container.
example: find_and_put_multiple_items(["creditcard 2", "creditcard 3"], ["countertop 1",
"dresser 1"])
> go to countertop 1
```

```

On the countertop 1, you see a cellphone 2, a creditcard 4, a creditcard 3, a creditcard 2, a mirror 1, a pencil 2, and a pencil 1. 98
> take creditcard 2 from countertop 1 99
You pick up the creditcard 2 from the countertop 1. 100
> go to dresser 1 101
On the dresser 1, you see a mug 1, and a television 1. 102
> put creditcard 2 in/on dresser 1 103
You put the creditcard 2 in/on the dresser 1. 104
> go to countertop 1 105
On the countertop 1, you see a cellphone 2, a creditcard 4, a creditcard 3, a mirror 1, a pencil 2, and a pencil 1. 106
> take creditcard 3 from countertop 1 107
You pick up the creditcard 3 from the countertop 1. 108
> go to dresser 1 109
On the dresser 1, you see a creditcard 2, a mug 1, and a television 1. 110
> put creditcard 3 in/on dresser 1 111
You put the creditcard 3 in/on the dresser 1. 112

```

C. ALFWorld Expert Skills

```

find(object): 1
The agent should visit each location in order until the object is seen. If a location is 2
closed, such as a drawer, or fridge, open the location to see inside.
In addition, I should visit locations where the object is likely to appear. For example, 3
an apple is more likely to be on the countertop.
find(apple) 4
> think: First I need to find an apple. An apple is more likely to appear in fridges (1), 5
diningtables (1-3), sidetables (1), countertops (1), sinkbasins (1), garbagecan (1).
I can check one by one, starting with fridge 1.
OK. 6
> go to fridge 1 7
The fridge 1 is closed. 8
> open fridge 1 9
You open the fridge 1. The fridge 1 is open. In it, you see a lettuce 2, a mug 2, a 10
potato 2, and a tomato 1.
> go to diningtable 1 11
On the diningtable 1, you see nothing. 12
> go to diningtable 2 13
On the diningtable 2, you see a pineapple 1. 14
> go to diningtable 3 15
On the diningtable 3, you see a bowl 2, a bowl 1, a bread 2, and a bread 1. 16
> go to sidetable 1 17
On the sidetable 1, you see a cup 1, a lettuce 1, a peppershaker 3, a potato 1, and a 18
saltshaker 1.
> go to countertop 1 19
On the countertop 1, you see a butterknife 1, a dish sponge 1, and a soapbottle 1. 20
> go to sinkbasin 1 21
On the sinkbasin 1, you see a cup 2, a egg 1, a mug 1, and a potato 3. 22
> go to garbagecan 1 23
On the garbagecan 1, you see a apple 3, and a egg 3. 24
> think: I found apple 3. Next, I can take the apple 25
> take apple 3 from garbagecan 1 26
You pick up the apple 3 from the garbagecan 1. 27
28
place(object, location): 29
The agent should go to the target location. Then, place the current object at the 30
location.
place(apple 3, sidetable 1) 31
> think: I have an apple. I need to put it in/on a sidetable. 32
OK. 33
> go to sidetable 1 34
On the sidetable 1, you see a cup 1, a lettuce 1, a peppershaker 3, a potato 1, and a 35
saltshaker 1.
> put apple 3 in/on sidetable 1 36

```



```
You put the apple 3 in/on the sidetable 1. 37
clean(object): 38
The agent should go to the sinkbasin. Then, the agent should use the "clean" action with 39
the sinkbasin. 40
clean(lettuce 1) 41
> think: I have lettuce. I need to go to sinkbasin 1 and clean it. 42
OK. 43
> go to sinkbasin 1 44
On the sinkbasin 1, you see a apple 2, a ladle 2, a spoon 1, and a tomato 3. 45
> clean lettuce 1 with sinkbasin 1 46
You clean the lettuce 1 using the sinkbasin 1. 47
heat(object): 48
The agent should go to the microwave. Then, the agent should use the "heat" action with 49
the microwave. 50
heat(egg 2) 51
> think: I have an egg 2. I need go to microwave 1 and heat it. 52
OK. 53
> go to microwave 1 54
The microwave 1 is closed. 55
> heat egg 2 with microwave 1 56
You heat the egg 2 using the microwave 1. 57
cool(object): 58
The should go to the fridge. Then, the agent should use the "cool" action with the fridge 59
cool(mug 3) 60
> think: I have a mug 3. Next, I need to go to fridge 1 and cool it. 61
OK. 62
> go to fridge 1 63
The fridge 1 is closed. 64
> cool mug 3 with fridge 1 65
You cool the mug 3 using the fridge 1. 66
look(object): 67
The agent should find a desklamp find(desklamp). Then, I can use the "use" action with 68
the desklamp. 69
look(bowl 1) 70
> think: I have a bowl 1. Next, I need to find a desklamp. A desklamp is more likely to 71
appear in desk (1), sidetable (1-2), shelf (1-5), bed (1), drawer (1-3). I can check 72
one by one, starting with desk 1. 73
OK. 74
> go to desk 1 75
On the desk 1, you see a alarmclock 2, a book 1, a cellphone 1, a keychain 1, a laptop 2, 76
a laptop 1, and a pen 3. 77
> go to sidetable 1 78
On the sidetable 1, you see a cd 1, a pen 1, and a pencil 1. 79
> go to sidetable 2 80
On the sidetable 2, you see a alarmclock 1, a desklamp 1, and a pen 2. 81
> think: Now I find a desklamp (1). Next, I need to use it. 82
OK. 83
> use desklamp 1
You turn on the desklamp 1.
```

D. Skill Extraction Prompt

```
You are a helpful assistant who can help build an automated agent 1
interacting with an environment. 2
3
Your task is to identify what "skills" would be useful in accomplishing tasks. 4
5
Here are some exemplar tasks the household agent can be given in this domain: 6
7
```

- put some spraybottle on toilet.
- find some apple and put it in sidetable.
- ...
- put two creditcard in dresser.
- put two cellphone in sofa.

The agent may have useful "skills":

- A skill accomplishes useful parts of the goal task by outputting certain text actions.
- A skill should consist of multiple actions executed sequentially (usually more than two actions).
- A skill should be "reusable" across different tasks and generalizable.
- A skill should not intersect with other skills.

Your task is to identify what skills would be useful in accomplishing tasks.

For each trajectory given in the following, try to summarize what the agent is doing in a high-level abstraction of skill.

(1) skill_name(arg1, arg2, ...)

(2) instructions: the instruction of a skill should explain:

- step by step explanation on how to use this skill
- tips on unexpected steps or advice on using this skill

(3) example: Part of the provided trajectories that show how to use the skill.

Indicate the skill being used and the arguments used for this example skill.

You must answer in the following format:

- For each skill discovered, place the text in a markdown fence.
- (1) In the first line, briefly give the name of the skill and arguments
- (2) In the second line, write step by step instructions, and tips on how to use this skill.
- (3) Write one example extracted from the example trajectory, and indicate the arguments for this skill.

Only include parts of the trajectory that are relevant for this skill.

```
```skill
{{ skill_name }}
instructions: {{ instructions }}
example: {{ skill_name(example_arguments) }}
> {{ action }}
{{ observation }}
> {{ action }}
{{ observation }}
```
```

E. Demonstration Relabeling Prompt

You may want to use some of the following skills relevant for this task:

[[insert list of skills here ...]]

Given the following context:

You are in the middle of a room. Looking quickly around you, you see a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 1, a diningtable 3, a diningtable 2, a diningtable 1, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a sidetable 1, a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.

Your task is to: find some apple and put it in sidetable.

Modify the this "think" action by adding a citation on which skills may be relevant in the text.

SkillAct: Using Skill Abstractions Improves LLM Agents

IMPORTANT: Only output the modified think action and nothing else.

Example: change "> think: (action text)" to "> think: [skill11, skill12] (action text)".

> think: To solve the task, I need to find and take an apple, then put it in sidetable.

11

12

13

14