StateAct: State Tracking and Reasoning for Acting and Planning with Large Language Models

Anonymous ACL submission

Abstract

Planning and acting to solve 'real' tasks using large language models (LLMs) in interactive environments has become a new frontier for AI methods. While recent advances allowed LLMs to interact with online tools, solve robotics tasks and many more, long range reasoning tasks remain a problem for LLMs. Existing methods to address this issue are very resource intensive and require additional data or human crafted rules, instead, we propose a simple method based on few-shot in-context-learning alone to enhance 'chain-of-thought' with state-tracking for planning and acting with LLMs. We show that our method establishes the new state-of-the-art on Alfworld for in-context-learning methods (**+14%** over the previous best few-shot in-context-learning method) and performs on par with methods that use additional training data and additional tools such as code-execution. We also demonstrate that our enhanced 'chain-of-states' allows the agent to both solve longer horizon problems and to be more efficient in number of steps required to solve a task. Finally, we also conduct ablation studies and show that 'chain-of-thoughts' helps state-tracking accuracy, while a json-structure harms overall performance. We open-source our code and annotations at **anonymous URL**.

1 Introduction

Using the in-build world- and commonsense-knowledge[1] of large language models (LLMs), such as GPT-3, Mixtral, Gemini (Brown et al., 2020; Jiang et al., 2024; Anil et al., 2023) to perform interactive reasoning tasks has become a frontier in AI research, with "AI Agents" now able to solve a range of multi-modal complex tasks (Durante et al., 2024). These range from solving (simulated) robotics tasks (Puig et al., 2018; Shridhar et al., 2021) and digital tasks such as online shopping (Yao et al., 2023a) and navigating operating systems (Liu et al., 2023), to playing a variety of games (Côté et al., 2019; Liu et al., 2023).

While LLMs are increasingly successful in solving AI agent tasks, Li et al. (2023b) show that LLMs struggle with long range understanding and Coelho et al. (2024) show that LLMs often focus on the beginning of the textual history, further amplifying the problem. Existing efforts to address these issues are resource intensive, Wu et al. (2024) required human expert annotations of rules, Sun et al. (2023) a code execution environment with carefully crafted code-based prompts, Fu et al. (2024) use additional training data together with retrieval augmented generation (RAG) to help the AI agent.

To overcome these challenges, we introduce "StateAct" a novel LLM agent based on few-shot in-context-learning that tracks the goal by 'reminding' the agent of the goal and explicitly keeps track of the agent's state (such as location and inventory). We also utilise 'chain-of-thought' (i.e. explicit reasoning (Wei et al., 2023)) as an optional addition. At the core of the method lies a simple annotation with goal-, state- and reasoning- traces of the few-shot examples, that we call 'chain-of-states'.

Our method establishes a new state-of-the-art for Alfworld (Shridhar et al., 2021) for few-shot in-context-learning based methods, outperforming the previous best method by 14% and even outperforms methods that use external tools by 2.5%. Furthermore, while our method does not require additional data or external tools and only minimal and easy human annotations, we perform on part with the current state-of-the-art that is much more resource intensive and requires additional data and expert human annotations. Comparing to in-context-learning methods, we also demonstrate that our method is both more efficient in terms of number of steps to solve a task and is able to solve longer-horizon tasks, validating the hypothesis that explicit state-

[1]Commonsense- and world- knowledge as explored by Lauscher et al. (2020) for example.

tracking and goal-tracking help with solving longer horizon reasoning and planning tasks.

2 Background

AI agents have historically used reinforcement learning (RL) to solve tasks (Sutton and Barto, 2018). With the dawn of LLMs works such as Li et al. (2022); Nottingham et al. (2023) combined LLMs and RL and trained additional policies or value functions to make predictions.

2.1 In-context-learning approaches

Huang et al. (2022a,b); Singh et al. (2022) were among the first to use LLMs directly to act in an interactive environment, their method produces agent actions as output after receiving environment observations as input.

ReAct (Yao et al., 2023b) took this work further by combining 'acting' (Huang et al., 2022a) and 'chain-of-thought' (Wei et al., 2023). ReAct establishes state-of-the-art for in-context-learning only based approaches and while it is a very scalable method, the performance (i.e. success rate) is still limited.

ExpeL (Zhao et al., 2023) uses additional training data to generate 'insights' and 'success trajectories' during training. At inference time they look-up the closest 'success trajectories' as few-shot examples to the agent (as opposed to fixed few-shot examples that we use) and augment them with these 'insights' to perform the final inference. They achieve 59% on Alfworld using retrieved 'success trajectories' (as few shot examples) + 'insights' and 50% using the same few-shot examples as ReAct + 'insights'.

2.1.1 In-context-learning and additional tools

The current state-of-the-art for in-context-learning based approaches in combination with additional tools is AdaPlanner (Sun et al., 2023). They introduce a code-based prompt (Li et al., 2023a) and use code-execution as an additional tool to execute the LLM generations to feed them back into the next prompt. The short-coming of Adaplanner is that it requires very complex human crafted prompts that are hard to scale to new environments as well as the additional step of requiring code-execution.

2.2 State tracking in LLM-based agents

Concurrent work to ours, AutoGuide (Fu et al., 2024), uses ReAct as the base agent and additional training data to create 'state-aware' text-based guidelines for the LLM-agent, they then use a type of retrieval augmented generation (RAG) process to guide the decision making process. They embed the current observation as a summary (this is what they call 'state-aware') and use a LLM to 'look up' the relevant 'state-aware' guideline, which is then fed into a final LLM to generate an action. Using training data and LLM-based RAG they achieve 79.1% on top of a ReAct agent on Alfworld. Their training and RAG approach could be used complimentary to our StateAct LLM agent[2].

Chen et al. (2024) propose state-tracking as a way to help the agent solve the task, without training data. Their method differs to ours two-fold. Firstly, they employ a complex sequence of components working together (an LLM-based attention over the observations, a LLM-based compilation of a complex state and finally a prediction of a program). Secondly, their system involves execution of actual programs. Our method on the hand requires a straight-forward extension of 'chain-of-thought' and uses a single LLM call to produce the state, thought and action and we do not require program execution.

2.2.1 Fine-tuning approaches

Previous fine-tuning approaches did not significantly enhance performance (Zhou et al., 2024; Yao et al., 2023b; Shridhar et al., 2021). A concurrent work, however, ActRe (Yang et al., 2024b) achieves 83% by fine-tuning gpt-3.5-1106 on additional training data.

2.3 "Multi-Agent Conversation" approaches

A new trend is to use multiple LLMs concurrently to 'chat' to one another to produce a result. A recent work in this direction by (Wu et al., 2023) achieves 67% on Alfworld.

2.4 Joined rule and LLM based agents

StateFlow (Wu et al., 2024) uses Finite State Machines (FSMs) combined with LLMs to solve Alfworld. These FSMs are human-expert crafted states, transitions and rule-based heuristics, where the LLM is asked to perform limited tasks in each of the given states. While their performance is 82% on Alfworld, we believe this result is more comparable to a rule-based 'experts' than an LLM-based agent (notably close to 100% can be achieved on

[2]The cost of using RAG at every generation step could be significantly higher than our method, however.

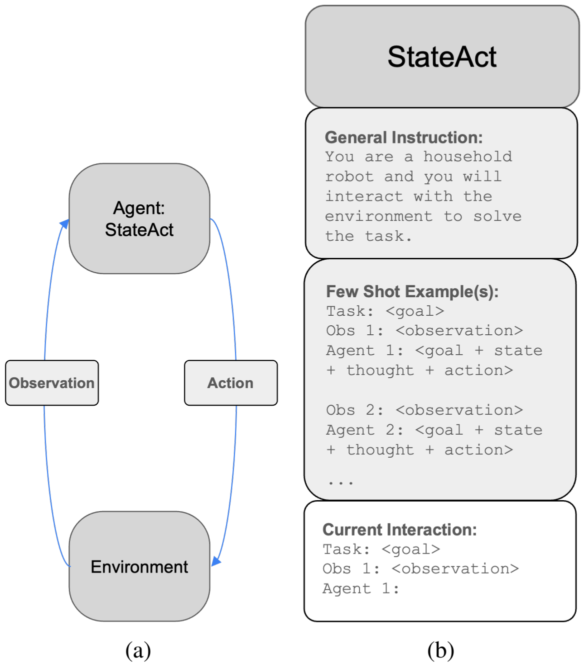

Figure 1: StateAct interacts with the environment directly, without additional tools or resources (a). A skeleton of the StateAct prompt (b).

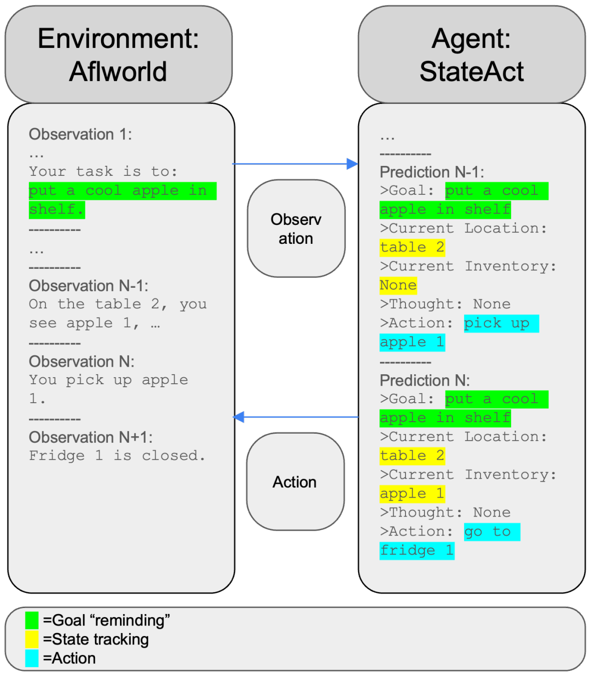

Figure 2: An example of how StateAct is working. The depicted traces are from the Alfworld environment.

Alfworld using hand-crafted 'experts' (Shridhar et al., 2021)).

2.5 Approaches with multiple trials

Works such as Reflexion (Shinn et al., 2023) or work by Sun et al. (2023); Yang et al. (2024a) are methods that perform several evaluations on the same test-environments and perform learning loops (during evaluation) on these test-set environments to improve the score. While the score necessarily goes up (as only the failed environments are re-run) we do not dive into this, as this does not compare to the actual agent performance and 'reflection' can be used on top of most LLM-based agents (e.g. Reflexion uses ReAct as the base agent).

3 Method

StateAct is an LLM-based AI agent that works on top of pre-trained large language models. StateAct takes the *textual* 'observation' from the environment and after a single call to the pre-trained LLM returns the 'action' back to the environment (without the use of additional tools or resources), see Figure 1a.

StateAct utilises in-context-learning (Brown et al., 2020; Wei et al., 2023) to make the agent interact with the environment. At the core of the approach is a prompt that consists of few-shot examples of successful 'interaction traces' as well as the current 'interaction trace' up to the current step in the environment, see Figure 1b. An 'interaction trace' consists of alternating 'observations' from the environment and desired (or actual) 'outputs' from the LLM. In the case of StateAct the LLM is tasked to generate the 'goal', 'state', 'thought' and 'action'. The 'action' is then extracted and passed to the environment to produce the next observation, see Figure 2. For example a sample of the trace for step n in Alfworld:

Observation n:

```
On the countertop 1, you see a tomato 1.
```

LLM Output n:

```
>Goal: Put a clean tomato in fridge
Current Location: countertop 1
Current Inventory: None
Thought: I find the tomato, now I need to
take it.
Action: take tomato 1
```

In the above example the agent produces the action '`take tomato 1`', in response to the observation '`On the countertop 1, you see a tomato 1`'.

3.1 Goal-reminders and state- tracking using LLMs

In order to make StateAct more precise we introduce simple notation. By denoting π as the policy of an AI agent, in the standard case at time step t the policy predicts action a_t, given the history of observations and actions $[o_t, a_{t-1}, ..., a_0, o_0]$.

$$\pi(a_t | o_t, a_{t-1}, ..., a_0, o_0) \quad (1)$$

Where a_t is the action produced by the agent at step t and o_t is the observation produced by the environment at step t after receiving action a_t as input. Usually, the first observation o_0 also contains the 'goal' description for the given environment.

For our case we need to enhance the policy to incorporate the 'state'. Similar to previous work (Yao et al., 2023b) we introduce the *context* vector, c_t. The context vector contains the action, as well as the other additional predictions of the agent, i.e. $c_t = (g_0, s_t, r_t, a_t)$. Where g_0 is the goal and always remains the same (for a given environment) and uses the goal extracted from o_0, s_t represents the predicted state at time step t, r_t represents 'chain-of-thought' style 'reasoning' at time step t, and a_t represents the action at time step t, as before. The new policy π then becomes:

$$\pi_{contextual}(c_t | o_t, c_{t-1}, ..., c_0, o_0) \quad (2)$$

In our case the LLM acts as $\pi_{contextual}$ and produces the context vector at every time step.

4 Experimental setup

Our aim is to study long-range acting, planning and reasoning capabilities of LLM-based agents. To achieve this, in line with previous work, we turn to simulated environments as an evaluation framework and to API-based state-of-the-art large language models. Specifically, we use Alfworld (Shridhar et al., 2021), a household robotics environment, and Webshop (Yao et al., 2023a), an online shopping environment, as simulated environments. As LLM we use OpenAI's[3] `gpt-3.5-turbo-1106`.

4.1 Alfworld

Alfworld (Shridhar et al., 2021) is based on a 3D, visual, household robotics environment called Alfred (Shridhar et al., 2020), which was translated into a text-based environment for ease of use for language based AI models, see Figure 3. Alfworld has a total of 135 test-set examples and six environment types. It features long-time horizons, partial observability, an out-of-distribution evaluation set

[3]`https://openai.com`, last accessed June 2024.

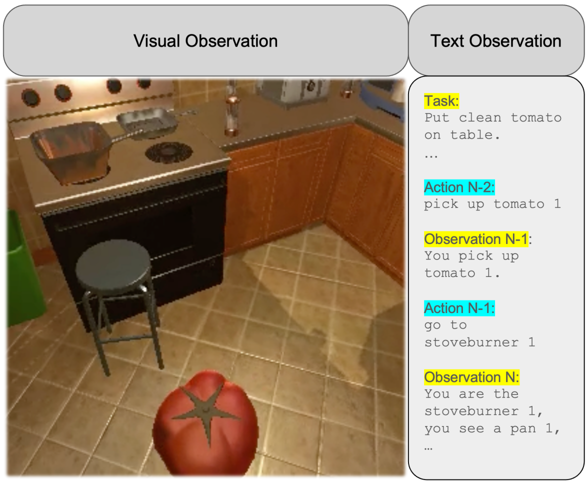

Figure 3: An example textual interaction in Alfworld (right) and corresponding 3D rendering (left).

and text-based interactions. Alfworld simulates a household environment with a household assistant robot tasked with solving problems, e.g. `clean an apple and put it on a table`. The robot (or agent) then needs to perform a series of 'high-level' operations to accomplish the tasks, e.g. '`go to fridge 1`', '`open fridge 1`'. At every step the environment provides a textual observation or feedback that the command has failed, e.g. '`You open the fridge 1`', '`You see apple 1`'. The underlying text engine is based on Textworld (Côté et al., 2019). See Appendix A for a complete list of commands and details on environments.

4.1.1 Alfworld correction

In our research we identified that Alfworld has a specific syntactic feature for the `put` command, namely `put <object> in/on <place>`, where "in/on" needs to be written exactly this way and using only "in" or only "on" produces a failed command. We observed this issue with LLMs on this environment and we propose a simple fix for it. We map: 1. "`put <object> in <place>`" and 2. "`put <object> on <place>`" to the command accepted by Alfworld, namely "`put <object> in/on <place>`".

Methods such as AdaPlanner (Sun et al., 2023) have avoided this issue because they use code-based prompts and regex parsers. However, methods such as ReAct (Yao et al., 2023b) and ExpeL (Zhao et al., 2023) have been affected, lowering their potential performance. In our work, we also report the results for ReAct using *corrections*.

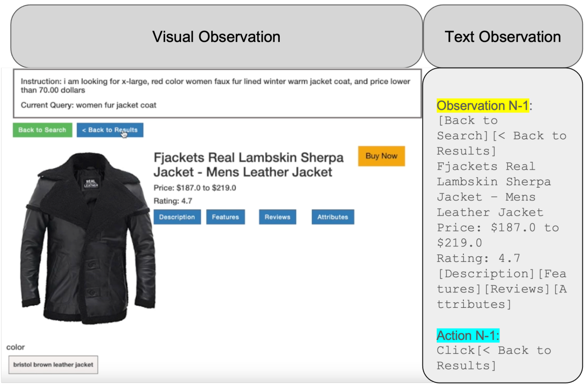

Figure 4: An example textual interaction in Webshop (right) and corresponding e-commerce website rendering (left).

4.2 Webshop

Webshop (Yao et al., 2023a) is a simulation of an online shopping experience. Given a task, e.g. "`I want a blue water-proof winter jacket, less than $100`", the agent needs to search a product catalogue, browse through the search results and select the most fitting product, select the attributes, e.g. `colour`, `size`, and then buy the product. In line with previous work we use the text-based version of Webshop, where all descriptions of the website are given in text form, see Figure 4. Webshop features a realistic large-scale product catalogue, a search engine and very varied product attributes depending on the category of product. See Appendix B for more details. In total the test set consists of 500 examples and each one is of the type "search and buy a product". Overall, Webshop has a maximum of 15 steps and two commands: 1. `search[<query>]`, 2. `click[<button>]`.

4.3 In context learning

Since ReAct (Yao et al., 2023b) forms the underlying agent for many current (Zhao et al., 2023) and state-of-the-art approaches (Fu et al., 2024), we use the same few-shot 'interaction traces' as ReAct. The main reason is to have a fair comparison and isolate additional effect - such as performance change from different in-context examples.

In total, Alfworld has six types of tasks and ReAct uses two in-context examples per task type to prompt the language models. On average each ReAct example ranges from 352 words to 591 words (590 token to 935 tokens). For our study we reuse the observations, thoughts and actions and annotate these examples further with goal 'reminders' and state tracking, which results in a range from 484 to 911 words (807 tokens to 1458 tokens) per example. During our annotation we discovered minor errors in the ReAct prompts and fixed them as well. We release all our annotations with our code release. In comparison, AdaPlanner (Sun et al., 2023), uses a different code based approach and the prompt has 1104 words (2015 tokens) on average.

We use the two-shot examples from ReAct for Alfworld and the one-shot example from ReAct for Webshop for all our experiments.

Method	Version	AW	WS
ReAct	gpt-3.5-turbo-1106	**40.7**	**18.2**
ReAct	gpt-3.5-turbo-0125	30.37	14.6

Table 1: ReAct success rate on Alfworld (AW) and Webshop (WS) using available gpt-3.5 models.

4.4 Models

In line with previous work we focus our attention on the API based LLMs to compare performance. Many OpenAI models have become deprecated. Notably, all models from ReAct and AdaPlanner(Sun et al., 2023) `davinci-002`, `gpt-3.5-turbo-0301` and `gpt-3.5-turbo-0613` are deprecated now. Therefore, we re-implemented ReAct and ran the experiments to determine the most suitable model, see Table 1. We establish that `gpt-3.5-turbo-1106` is the best performing (from the ones that remain available) on ReAct and we therefore chose this one. Furthermore, we did not opt for GPT-4 level models as these are prohibitively expensive[4]. Furthermore, we use temperature 0 for all experiments and sample only the top 1 response, see Appendix C the exact settings.

4.5 Metrics

In terms of metrics we use the pre-defined metrics of Alfworld and Webshop, namely success rate (SR). Success is a binary metric per each environment in the respective test sets (135 and 500 respectively). Success in Alfworld means the agent has successfully complete the whole task. In Webshop it means the agent has bought an item that has a hundred percent match with the desired item based on a partially hidden list of attributes of the shopping item (e.g. the colour, size, price, etc.).

[4] A single evaluation run on alfworld costs approx. $8 using gpt-3.5 and ReAct, gpt-4 would cost 10+ times more.

5 Results

5.1 Alfworld

For Alfworld we present the results for ReAct, AdaPlanner with and without code execution and StateAct (ours), which consists of `goal + state + thought + action`. We also show StateAct without each of the components (i.e. without goal, state and thought). Interestingly we find, contrary to previous findings, that 'thought' or 'reasoning' actually sometimes harms the performance.

In Table 2, we can see that StateAct with all `goal+state+thought` and the correction performs the best. It outperforms ReAct with correction by around 13 points (using the same GPT model for ReAct) and by around 9 points (using the better model for ReAct). StateAct also outperforms ReAct by 22 points when corrections are not used. Furthermore, StateAct even outperforms AdaPlanner by 2.48 points, an approach that uses regex for command mapping (similar to our correction) and code-execution.

Perhaps the most surprising finding is that the simple correction described in Section 4.1.1 leads to a 16 and 23 point jump for ReAct and a 27 point jump for StateAct. This indicates that the model generally performs very well, however, struggles with minute differences in *domain specific* syntax.

Method	GPT-3.5	SR %
Baselines w/o corrections		
ReAct	0301*	51.9
ReAct	1106	40.7
ReAct	0125	30.37
ReAct (joined***)	1106	36.30
Results w/o corrections		
StateAct (ours)	1106	50.37
- w/o thought	1106	**62.96**
- w/o goal	1106	42.96
- w/o state	1106	44.44
Baselines with corrections		
ReAct + corr.	0301*	**68.15**
ReAct + corr.	1106	63.70
AdaPlanner No-Exec**	0301*	46.66
AdaPlanner	0301*	**75.56**
Results with corrections		
StateAct + corr. (ours)	1106	**77.04**
- w/o thought	1106	64.44
- w/o goal	1106	74.04
- w/o state	1106	64.44

Table 2: Success Rate (SR) on the 135 test-set examples from Alfworld. *gpt-3.5-0301 is scheduled to be deprecated in June 2024. **No-Exec means AdaPlanner without code execution. ***joined means that `thought + action` are produced at every turn. AdaPlanner results are from (Sun et al., 2023). All other results are 'single run'.

5.2 Webshop

For Webshop we present results for ReAct and StateAct (ours). Similarly, to Alfworld we also present the results of StateAct without each of goal, state and thought. See Table 3. Interestingly, we see that removing thought produces the highest results and outperforms ReAct by 10 points. Our hypothesis is that *domain specific* syntax, which is more prevalent in Webshop than Alfworld, conflicts with using verbose thoughts.

5.3 Summary of results

In conclusion we found that our simple goal-reminding and state tracking approach that purely relies on in-context learning outperforms previous in-context learning approaches by more almost 10 points and even outperforms leading approaches that rely on code-execution. Interestingly, we found that the approaches are quite sensitive to domain specific syntax and that when this is the case 'thoughts' that are verbose can harm performance.

6 Analysis and Ablations

In the results section we discovered that our methods perform better than previous state-of-the-art. This answers the question that we can perform better with in-context learning without resorting to additional tools, data or bigger models. In this section we want to analyse our results further and particularly also answer if our second hypothesis that *goal 'reminding'* and *state tracking* help with long-range reasoning actually holds. For all ablation studies we focus on Alfworld as it has two favourable properties over Webshop. Firstly, Alfworld has a longer time horizon (50 steps vs. 15 in Webshop), with tasks taking an average of less than 10 steps in Webshop and around 20 to 30 steps in Alfworld. Secondly, Alfworld has much less domain specific syntax and is purely text based, while Webshop has a more specific syntax to follow.

Method	GPT-3.5	Success Rate %
Baselines		
Rule-based*	N/A	9.60
RL*	N/A	17.60
ReAct	1106	17.80
Results		
StateAct (ours)	1106	17.00
- w/o thought	1106	**27.80**
- w/o goal	1106	20.40
- w/o state	1106	21.00

Table 3: Success Rate (SR) on the 500 test-set examples from Webshop. *results taken from (Yao et al., 2023a). Results are 'single run', except for StateAct without thought and ReAct, where we ran the experiment twice and reported the average.

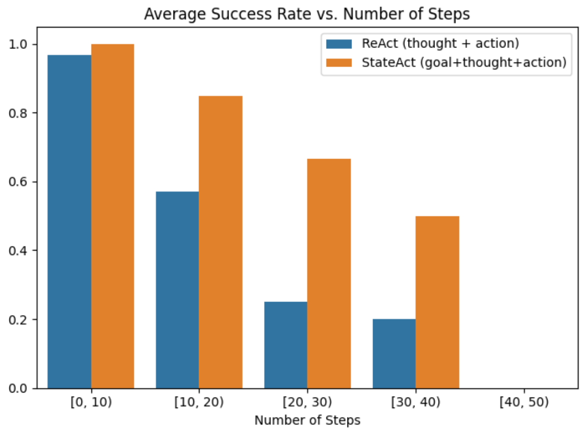

Figure 5: Goal vs. No Goal, on the 135 test examples from Alfworld, using gpt-3.5-turbo-1106 without correction.

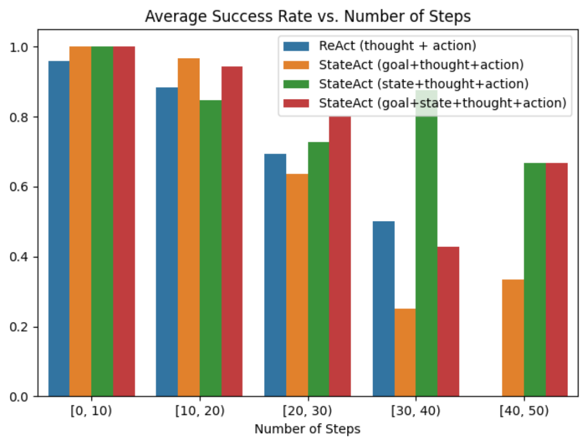

Figure 6: State vs. No State, on the 135 test examples from Alfworld, using gpt-3.5-turbo-1106 with correction.

6.1 Do goal reminders help with long range tasks?

For this purpose we compare the original ReAct (thought+action) with just adding the goal in, i.e. StateAct (goal + thought + action). In Figure 5 we can see that while the performance of both ReAct and StateAct goes down as there are more number of steps the goal tracking has a significantly better relative performance as the number of steps increase.

To verify that this actually means that goal tracking helps with performance, as opposed to just increasing the number of steps it takes to solve a task, we calculate the average number of steps for ReAct (ignoring empty 'thought' turns, as otherwise ReAct would have even more steps) and StateAct. Table 4 clearly show that ReAct with an average of 38.84 steps to solve an environemnt is the least efficient and StateAct with an average of 28.96 steps to solve an environment is the most efficient. This shows that not only does goal tracking help with longer range tasks, it also helped with efficiency by shortening the tasks.

6.2 What effects does state-tracking have?

We also analyse whether state tracking helps with long-range reasoning and efficiency. We compared the full StateAct against StateAct without state-tracking as well as ReAct (thought + action) against StateAct with state-tracking added (state + thought + action). In Figure 6 we see that state tracking also helps with long-range reasoning. In fact, we can see that reasoning alone is unable to solve tasks longer than 40 steps, while with state tracking even longer-range tasks can be solved than with goal-tracking alone. Also, looking at Table 4 we see that state-tracking makes the model the most efficient[5]. Therefore we find that explicit state-tracking even further helps with long-range tasks and helps the agent solve the tasks more efficiently than without.

6.3 Does the model perform actual state tracking?

We ask ourselves the question if the model is actually performing state-tracking. For that purpose we look at Alfworld and construct a self-verification algorithm that is able to track the state heuristically[6] based on the actions the agent takes. For example if the agent produces the action `go to fridge 1`

[5] In terms of cost we found that despite our method using a twice longer prompt, our cost remains similar at around $8 for the full Alfworld run, since we solve tasks more efficiently and use fewer number of steps.

[6] On our few-shot prompts it achieves 100% correctness.

Method	Avg. Steps ↓
ReAct	38.84
StateAct (goal+thought+action)	31.19
StateAct (full)	28.96
ReAct with corr.	31.49
StateAct with corr.	**19.11**
- w/o thought	23.76
- w/o goal	20.09

Table 4: Average number of Steps (Avg. Steps) [lower is better] on the 135 test-set examples from Alfworld. gpt-3.5-1106 for all methods.

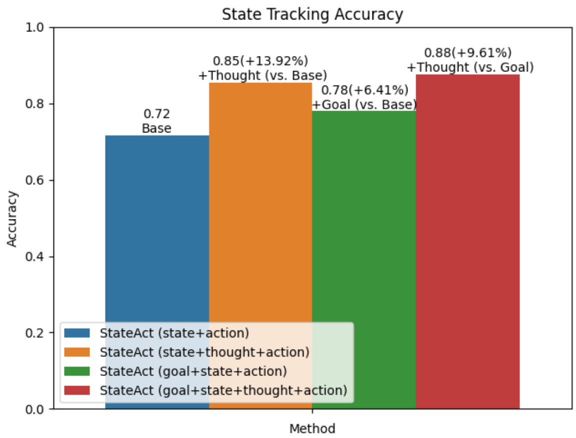

Figure 7: State tracking accuracy for StateAct on 135 test examples of Alfworld using gpt-3.5-1106.

and the environment accepts this action we update the state with `current location: fridge 1`. We compare the *'gold' state* against the predicted state. Figure 7 shows that StateAct in fact does correct state-tracking 88% of the time. We also observe that thoughts and goals help the state tracking.

6.4 Does json structure help with performance?

Since we found that domain specific syntax harms performance, we wondered whether adding a structured format like json would help. For this purpose, we re-ran StateAct on Alfworld, but translated the state into a json format, see Appendix D for more details. Surprisingly, we found that the json format harms performance significantly, see Table 5. However, we also see that *corrections* help the the json format less, indicating that json helps with syntax, but harms performance.

7 Conclusion and future work

We propose a novel method *StateAct*, using our 'chain-of-states', based on in-context-learning alone and establish a new state-of-the-art for agents that do not perform training, even against methods that use code-execution. The method outperforms the previous state-of-the-art, that uses in-context-learning alone, between 9% and 20% given different models and tasks and outperform in-context-learning with tools (code-execution) by 3%. We also show that explicit *state-tracking* and *goal reminders* make the model more efficient as well as significantly help with longer range tasks.

We found that 'thoughts' or explicit reasoning do not always help performance. It would be very interesting to systematise 'thought' and 'states' and understand what contributes positively and why. Also, inspired by the good results of StateAct, it is interesting to see what other improvements can be done without resorting to training, larger model or external tools. Finally, problems related to *domain specific* syntax are also an interesting avenue of future work.

Method	SR%	SR (+json)%
StateAct	50.37	45.19 (-5.2)
StateAct w/ corr.	**77.04**	58.52(-18.5)

Table 5: Success Rate (SR) on the 135 test examples from Alfworld. Showing no-json vs. json, gpt-3.5-1106.

8 Ethical Considerations

8.1 Computational footprint

Running many of the experiments presented in this paper can have a significant computational footprint. We should consider the environment and financial resources for reproduciblity of our work. We aimed to address this concern by using `gpt-3.5-turbo` level models, reporting costs and minimising the cost of our method.

8.2 Hallucinations in LLMs

As LLM-based agents become more powerful and therefore more pervasive in our daily lives 'hallucinations' of LLMs can be very harmful (Wei et al., 2024). We hope that explicit state-tracking presented in this work can also lead to future work that can reduce 'hallucinations.'

9 Limitations

9.1 Languages and evaluation benchmarks

We evaluated our method only in the English language and on two evaluation benchmarks. While we do not expect major changes in other languages, this is something that should be investigated. Furthermore, performance on other benchmarks should be evaluated as well.

9.2 Reasoning traces rely on human judgement

Our prompts require human annotations, as such there is a natural bias present. This can have both task-performance implications as well as ethical implications.

References

Gemini Team Google Rohan Anil, Sebastian Borgeaud, Jeffrey Dean, and Oriol Vinyals. 2023. Gemini: A family of highly capable multimodal models. *ArXiv*, abs/2312.11805.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Siwei Chen, Anxing Xiao, and David Hsu. 2024. Llm-state: Open world state representation for long-horizon task planning with large language model. *Preprint*, arXiv:2311.17406.

João Coelho, Bruno Martins, João Magalhães, Jamie Callan, and Chenyan Xiong. 2024. Dwell in the beginning: How language models embed long documents for dense retrieval. *Preprint*, arXiv:2404.04163.

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. 2019. Textworld: A learning environment for text-based games. *Preprint*, arXiv:1806.11532.

Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, Katsushi Ikeuchi, Hoi Vo, Li Fei-Fei, and Jianfeng Gao. 2024. Agent ai: Surveying the horizons of multimodal interaction. *Preprint*, arXiv:2401.03568.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of state-aware guidelines for large language model agents. *ArXiv*, abs/2403.08978.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *Preprint*, arXiv:2201.07207.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022b. Inner monologue: Embodied reasoning through planning with language models. *Preprint*, arXiv:2207.05608.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts. *Preprint*, arXiv:2401.04088.

Anne Lauscher, Olga Majewska, Leonardo F. R. Ribeiro, Iryna Gurevych, Nikolai Rozanov, and Goran Glavaš. 2020. Common sense or world knowledge? investigating adapter-based knowledge injection into pretrained transformers. In *Proceedings of Deep Learning Inside Out (DeeLIO): The First Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 43–49, Online. Association for Computational Linguistics.

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. 2023a. Chain of code: Reasoning with a language model-augmented code emulator. *Preprint*, arXiv:2312.04474.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. 2023b. Loogle: Can long-context language models understand long contexts? *ArXiv*, abs/2311.04939.

Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, Jacob Andreas, Igor Mordatch, Antonio Torralba, and Yuke Zhu. 2022. Pre-trained language models for interactive decision-making. *Preprint*, arXiv:2202.01771.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023. Agentbench: Evaluating llms as agents. *Preprint*, arXiv:2308.03688.

Kolby Nottingham, Yasaman Razeghi, Kyungmin Kim, JB Lanier, Pierre Baldi, Roy Fox, and Sameer Singh. 2023. Selective perception: Optimizing state descriptions with reinforcement learning for language model actors. *ArXiv*, abs/2307.11922.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. 2018. Virtualhome: Simulating household activities via programs. *Preprint*, arXiv:1806.07011.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. 2020. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. *Preprint*, arXiv:1912.01734.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. *Preprint*, arXiv:2010.03768.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2022. Progprompt: Generating situated robot task plans using large language models. *Preprint*, arXiv:2209.11302.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. *ArXiv*, abs/2305.16653.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*, second edition. The MIT Press.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Jiaheng Wei, Yuanshun Yao, Jean-Francois Ton, Hongyi Guo, Andrew Estornell, and Yang Liu. 2024. Measuring and reducing llm hallucination without goldstandard answers. *Preprint*, arXiv:2402.10412.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. 2024. Stateflow: Enhancing llm task-solving through state-driven workflows. *ArXiv*, abs/2403.11322.

Yijun Yang, Tianyi Zhou, Kanxue Li, Dapeng Tao, Lusong Li, Li Shen, Xiaodong He, Jing Jiang, and Yuhui Shi. 2024a. Embodied multi-modal agent trained by an llm from a parallel textworld. *Preprint*, arXiv:2311.16714.

Zonghan Yang, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. 2024b. React meets actre: When language agents enjoy training data autonomy. *Preprint*, arXiv:2403.14589.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2023a. Webshop: Towards scalable real-world web interaction with grounded language agents. *Preprint*, arXiv:2207.01206.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners. *Preprint*, arXiv:2308.10144.

Qinhao Zhou, Zihan Zhang, Xiang Xiang, Ke Wang, Yuchuan Wu, and Yongbin Li. 2024. Enhancing the general agent capabilities of low-parameter llms through tuning and multi-branch reasoning. *Preprint*, arXiv:2403.19962.

A Alfworld

A.1 Environment Types

Alfworld has six different environment types: 1. *clean*, 2. *heat*, 3. *cool*, 4. *examine*, 5. *put*, 6. *puttwo*.

The '*clean*' task, e.g. `Task: Put a clean apple on table`, requires the agent to first find the apple, then clean it (in the sinkbasin) and then put it on a table.

The '*heat*' task, e.g. `Task: Put a hot pie on table`, requires the agent to first find the pie, then heat it (on the stoveburner) and then put it on a table.

The '*cool*' task, e.g. `Task: Put a cool tomato on table`, requires the agent to first find the tomato, then cool it (with the fridge) and then put it on a table.

The '*examine*' task, e.g. `Task: Examine the mug with the desklamp`, requires the agent to first find the mug, then find the desklamp, and then use the desklamp.

The '*put*' task, e.g. `Task: Find some apple and put it in sidetable`, requires the agent to first find an apple, and then put it on the sidetable.

The '*puttwo*' task, e.g. `Task: Put two cellphone in sofa`, requires the agent to first find one cellphone, and then put it on the sofa, and then to find the second one and put it on the sofa.

A.2 Action Types

Alfworld has the following valid actions: 1. *go to*, 2. *open*, 3. *close*, 4. *put*, 5. *take*, 6. *cool*, 7. *heat*, 8. *use*.

```
go to <place>
Example: go to table 1

open <object>
Example: open door 1

close <object>
Example: close door 1

put <object> in/on <place>
Example: put apple 1 in/on table 1

take <object> from <place>
Example: take apple 1 from table 1

cool <object> with <place>
Example: cool apple 1 with fridge 1

heat <object> with <place>
Example: heat apple 1 with fire 1

use <object>
Example: use desklamp 1
```

A.3 License

Alfworld has the permissible MIT license, we used it in line with the license.

B Webshop

B.1 Commands and environment

Webshop has one environment type: '*search & buy*', as well as two commands: 1. *search*, 2. *click*.

```
click[<button>]
Example: click[< Back to Search]

search[<query>]
Example: search[interesting book]
```

B.2 Prodcuts and attributes

Webshop has over 1 million real-world products across 5 main categories (fashion, makeup, electronics, furniture, and food) and 113 sub-categories.

B.3 License

Webshop has the permissible Princeton license, we used it in line with the license.

C Code snippet to call OpenAI / GPT-3.5

```
client = openai.OpenAI(
        # Defaults to os.environ.get("
    OPENAI_API_KEY")
        # api_key=OPENAI_KEY,
)

full_prompt = [{
    "role": "user",
    "content": prompt
}]

chat_completion = client.chat.
    completions.create(
    model="gpt-3.5-turbo-1106",
    messages=full_prompt,
    temperature=0.0,
    stop = ["\n\n"]
)
```

A prompt is given in Appendix E.

D StateAct Json Format

We translate the text based StateAct prompt:

```
>goal: put a hot apple in fridge
current location: starting location
current inventory: None
thought: To solve the task, I need to
    find and take an apple, then heat it
     with microwave, then put it in
    fridge. First I need to find an
    apple. An apple is more likely to
    appear in fridge (1), diningtable
    (1), coffeetable (1), drawer (1),
    cabinet (1-13), garbagecan (1). I
    can check one by one, starting with
    fridge 1.
action: go to fridge 1
```

Into the following json format:

```
>{"goal": "put a hot apple in fridge",
"current_location": "starting location",
"current_inventory": None,
"thought": "To solve the task, I need to
     find and take an apple, then heat
    it with microwave, then put it in
    fridge. First I need to find an
    apple. An apple is more likely to
    appear in fridge (1), diningtable
    (1), coffeetable (1), drawer (1),
    cabinet (1-13), garbagecan (1). I
    can check one by one, starting with
    fridge 1."
"action": "go to fridge 1"
}
```

E StateAct Prompts

E.1 Example Alfworld prompt for the 'heat' task

```
Interact with a household to solve a
    task.


Here are 2 examples:

You are in the middle of a room. Looking
     quickly around you, you see a
    cabinet 13, a cabinet 12, a cabinet
    11, a cabinet 10, a cabinet 9, a
    cabinet 8, a cabinet 7, a cabinet 6,
     a cabinet 5, a cabinet 4, a cabinet
     3, a cabinet 2, a cabinet 1, a
    coffeemachine 1, a countertop 1, a
    diningtable 1, 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: put a hot apple in
    fridge.

>goal: put a hot apple in fridge
current location: starting location
current inventory: None
thought: To solve the task, I need to
    find and take an apple, then heat it
     with microwave, then put it in
    fridge. First I need to find an
    apple. An apple is more likely to
    appear in fridge (1), diningtable
    (1), coffeetable (1), drawer (1),
    cabinet (1-13), garbagecan (1). I
    can check one by one, starting with
    fridge 1.
action: go to fridge 1

The fridge 1 is closed.

>goal: put a hot apple in fridge
current location: fridge 1
current inventory: None
thought: None
action: open fridge 1

You open the fridge 1. The fridge 1 is
    open. In it, you see a cup 1, and a
    egg 1.

>goal: put a hot apple in fridge
current location: fridge 1
current inventory: None
thought: None
action: go to diningtable 1

On the diningtable 1, you see a apple 1,
     a bread 1, a fork 3, a fork 2, a
    fork 1, a mug 2, a peppershaker 3, a
     plate 2, a pot 1, a soapbottle 3, a
     spatula 1, a spoon 3, a spoon 2, a
    spoon 1, a winebottle 3, and a
    winebottle 2.

>goal: put a hot apple in fridge
current location: diningtable 1
current inventory: None
thought: Now I find an apple (1). Next,
    I need to take it.
action: take apple 1 from diningtable 1

You pick up the apple 1 from the
    diningtable 1.

>goal: put a hot apple in fridge
current location: diningtable 1
current inventory: apple 1
thought: Now I take an apple (1). Next,
    I need to go to a microwave (1) and
    heat it.
action: go to microwave 1

The microwave 1 is closed.

>goal: put a hot apple in fridge
current location: microwave 1
current inventory: apple 1
thought: None
action: heat apple 1 with microwave 1

You heat the apple 1 using the microwave
     1.

>goal: put a hot apple in fridge
current location: microwave 1
current inventory: apple 1
thought: Now I heat an apple (1). Next,
    I need to put it in/on fridge 1.
action: go to fridge 1

The fridge 1 is open. In it, you see a
```

```
cup 1, and a egg 1.

>goal: put a hot apple in fridge
current location: fridge 1
current inventory: apple 1
thought: None
action: put apple 1 in/on fridge 1



You are in the middle of a room. Looking quickly around you, you see a cabinet 10, a cabinet 9, a cabinet 8, a cabinet 7, 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 diningtable 1, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 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 egg and put it in diningtable.

>goal: heat some egg and put it in diningtable
current location: starting location
current inventory: None
thought: To solve the task, I need to find and take an egg, then heat it with microwave, then put it in diningtable. First I need to find an egg. An egg is more likely to appear in fridge (1), countertop (1-3), diningtable (1), stoveburner (1-4), toaster (1), garbagecan (1), cabinet (1-10). I can check one by one, starting with fridge 1.
action: go to fridge 1

The fridge 1 is closed.

>goal: heat some egg and put it in diningtable
current location: fridge 1
current inventory: None
thought: None
action: open fridge 1

You open the fridge 1. The fridge 1 is open. In it, you see a lettuce 2, a mug 2, and a potato 3.

>goal: heat some egg and put it in diningtable
current location: fridge 1
current inventory: None
thought: None
action: go to countertop 1

On the countertop 1, you see a bread 1, a fork 1, and a saltshaker 1.

>goal: heat some egg and put it in diningtable
current location: countertop 1
current inventory: None
thought: None
action: go to countertop 2

On the countertop 2, you see nothing.

>goal: heat some egg and put it in diningtable
current location: countertop 2
current inventory: None
thought: None
action: go to countertop 3

On the countertop 3, you see a bowl 1, a butterknife 1, a egg 2, a kettle 2, a plate 1, a sink 1, and a spatula 2.

>goal: heat some egg and put it in diningtable
current location: countertop 3
current inventory: None
thought: Now I find an egg (2). Next, I need to take it.
action: take egg 2 from countertop 3

You pick up the egg 2 from the countertop 3.

>goal: heat some egg and put it in diningtable
current location: countertop 3
current inventory: egg 2
thought: Now I take an egg (2). Next, I need go to a microwave (1) and heat it.
action: go to microwave 1

The microwave 1 is closed.

>goal: heat some egg and put it in diningtable
current location: microwave 1
current inventory: egg 2
thought: None
action: heat egg 2 with microwave 1

You heat the egg 2 using the microwave 1.

>goal: heat some egg and put it in diningtable
current location: microwave 1
current inventory: egg 2
thought: Now I heat an egg (2). Next, I need to put it in/on diningtable 1.
action: go to diningtable 1

On the diningtable 1, you see a apple 2, a bread 3, a egg 1, a kettle 1, a knife 1, a mug 1, a papertowelroll 1, a peppershaker 2, a potato 1, a soapbottle 1, and a spatula 1.

>goal: heat some egg and put it in diningtable
current location: diningtable 1
current inventory: egg 2
thought: None
action: put egg 2 in/on diningtable 1
```

```
Here is the task.
<CURRENT TASK>
```

E.2 Example Webshop prompt

```
Webshop
Instruction:
i would like a 3 ounce bottle of bright
    citrus deodorant for sensitive skin,
     and price lower than 50.00 dollars
[Search]

Goal: Buy a 3 ounce bottle of bright
    citrus deodorant for sensitive skin,
     and price lower than 50.00 dollars
Current Location: Search Home Page
Current Selection: None
Thought: None
Action: search[3 ounce bright citrus
    deodorant sensitive skin]

Observation:
[Back to Search]
Page 1 (Total results: 50)
[Next >]
[B078GWRC1J]
Bright Citrus Deodorant by Earth Mama |
    Natural and Safe for Sensitive Skin,
     Pregnancy and Breastfeeding,
    Contains Organic Calendula 3-Ounce
$10.99
[B078GTKVXY]
Ginger Fresh Deodorant by Earth Mama |
    Natural and Safe for Sensitive Skin,
     Pregnancy and Breastfeeding,
    Contains Organic Calendula 3-Ounce
$10.99
[B08KBVJ4XN]
Barrel and Oak - Aluminum-Free Deodorant
    , Deodorant for Men, Essential Oil-
    Based Scent, 24-Hour Odor Protection
    , Cedar & Patchouli Blend, Gentle on
     Sensitive Skin (Mountain Sage, 2.7
    oz, 2-Pack)
$15.95

Goal: Buy a 3 ounce bottle of bright
    citrus deodorant for sensitive skin,
     and price lower than 50.00 dollars
Current Location: Search Results Page,
    page 1
Current Selection: None
Thought: B078GWRC1J and B078GTKVXY are
    bright citrus deodorant less then 50
     dollars. I can check B078GWRC1J
    first.
Action: click[B078GWRC1J]

Observation:
[Back to Search]
[< Prev]
scent [assorted scents][bright citrus][
    calming lavender][ginger fresh][
    simply non-scents]
size [travel set (4-pack)][3 ounce (pack
     of 1)][3-ounce (2-pack)]
Bright Citrus Deodorant by Earth Mama |
    Natural and Safe for Sensitive Skin,
     Pregnancy and Breastfeeding,
    Contains Organic Calendula 3-Ounce
Price: $10.99
Rating: N.A.
[Description]
[Features]
[Reviews]
[Buy Now]

Goal: Buy a 3 ounce bottle of bright
    citrus deodorant for sensitive skin,
     and price lower than 50.00 dollars
Current Location: Item B078GWRC1J Page
Current Selection: None
Thought: For 3 ounce bottle of bright
    citrus deodorant for sensitive skin,
     the item has options 'bright citrus
    ' and '3 ounce (pack of 1)' and
    seems good to buy.
Action: click[bright citrus]

Observation: You have clicked bright
    citrus.

Goal: Buy a 3 ounce bottle of bright
    citrus deodorant for sensitive skin,
     and price lower than 50.00 dollars
Current Location: Item B078GWRC1J Page
Current Selection: bright citrus
Thought: None
Action: click[3 ounce (pack of 1)]

Observation: You have clicked 3 ounce (
    pack of 1).

Goal: Buy a 3 ounce bottle of bright
    citrus deodorant for sensitive skin,
     and price lower than 50.00 dollars
Current Location: Item B078GWRC1J Page
Current Selection: bright citrus, 3
    ounce (pack of 1)
Thought: None
Action: click[Buy Now]
```