StateAct: Enhancing LLM Base Agents via Self-prompting and State-tracking

Nikolai Rozanov
Imperial College London
Department of Computing
nikolai.rozanov13@imperial.ac.uk

Marek Rei
Imperial College London
Department of Computing
marek.rei@imperial.ac.uk

Abstract

Large language models (LLMs) are increasingly used as autonomous agents, tackling tasks from robotics to web navigation. Their performance depends on the underlying *base agent*. Existing methods, however, struggle with long-context reasoning and goal adherence. We introduce **StateAct**, a novel and efficient base agent that enhances decision-making through (1) *self-prompting*, which reinforces task goals at every step, and (2) *chain-of-states*, an extension of chain-of-thought that tracks state information over time. StateAct outperforms ReAct, the previous best *base agent*, by over 10% on Alfworld, 30% on Textcraft, and 7% on Webshop across multiple frontier LLMs. We also demonstrate that StateAct can be used as a drop-in replacement for ReAct with advanced LLM agent methods such as test-time scaling, yielding an additional 12% gain on Textcraft. By improving efficiency and long-range reasoning without requiring additional training or retrieval, StateAct provides a scalable foundation for LLM agents. We open source our code to support further research at `https://github.com/ai-nikolai/stateact`.

1 Introduction

Leveraging the in-built world and commonsense knowledge[1] of large language models (LLMs), such as GPT, Gemini, DeepSeek,and Mixtral (Brown et al., 2020; Anil et al., 2023; DeepSeek-AI et al., 2025; Jiang et al., 2024) to perform interactive reasoning tasks has become a frontier in AI research. "AI Agents" are now able to solve a range of multi-modal complex tasks (Durante et al., 2024). These include simulated robotics tasks (Puig et al., 2018; Shridhar et al., 2021) and digital tasks, such as online shopping (Yao et al., 2023a), navigating operating systems (Liu et al., 2023), and playing a variety of games (Côté et al., 2019; Liu et al., 2023; Prasad et al., 2024).

At the core of an LLM agent is the *base agent*[2], such as Act (Huang et al., 2022a), ReAct (Yao et al., 2023c), and AdaPlanner (Sun et al., 2023). Existing efforts to improve LLM agents build on top of base agents and are usually quite resource-intensive: Wu et al. (2024) require human expert annotations of rules; Sun et al. (2023) require 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; Yang et al. (2024) use additional training data to fine-tune the LLM; Shinn et al. (2023) and Prasad et al. (2024) use test-time computation to produce better results. Notably, most of the current state-of-the-art methods use ReAct as the base agent.

In our work, we propose a new base agent, called StateAct. Starting with the observation that: i) LLM agents fail to follow the original instruction and goal in longer interactions; and ii) LLMs struggle with long context despite longer available contexts (Li et al., 2023b; Coelho et al., 2024), we propose two main contributions for improving the base agent. To address the first issue, we propose a mechanism for the agent to 'self-prompt' at every turn of the interaction to improve staying on track with the main goal. Concretely, our agent 'reminds' itself of the final goal at every turn. To address the second issue, we propose 'chain-of-states', an extension of chain-of-thought based on state tracking to help the agent stay on track with the current interaction and context. Using this, the agent keeps track of its state in the environment (such as location and inventory).

Our experiments show that StateAct achieves

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

[2] Base agents are the core building block of LLM agents. These are LLM agnostic and usually consists of prompts and communication between components such as LLMs and other systems.

Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025), pages 367–385
July 31, 2025 ©2025 Association for Computational Linguistics

near state-of-the-art performance without using additional training data or additional tools. StatAct also significantly outperforms ReAct across multiple tasks and eight frontier LLMs of varying sizes. Specifically, StateAct improves performance over ReAct by more than 10% on Alfworld (Shridhar et al., 2021), 30% on Textcraft (Prasad et al., 2024), and 7% on Webshop (Yao et al., 2023a).

Additionally, we validate that StateAct can serve as a drop-in replacement for existing extension methods. Using test-time computation (Prasad et al., 2024), we achieve a further 12% performance gain with StateAct on Textcraft using ADaPT.

2 Background

Due to recent advances, LLMs are now being used as autonomous agents in interactive environments as an alternative to traditional reinforcement learning (RL) (Sutton and Barto, 2018; Yao et al., 2023c; Li et al., 2022; Nottingham et al., 2023). LLM agents now tackle tasks in simulated environments such as Alfworld, Webshop, and Textcraft. LLM agents consist of the *base agent* and a possible extensions, such as fine-tuning (FT), retrieval augmented generation (RAG), test-time scaling (TTS), repeated attempts (REPS) or tools; see Table 1. **StateAct is a base agent that can be used in combination with extensions**.

2.1 Base agents

Huang et al. (2022a,b) 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., 2023c) takes this work further by combining *acting* (Huang et al., 2022a) and *chain-of-thought* (Wei et al., 2023). Progprompt and AdaPlanner (Singh et al., 2022; Sun et al., 2023) use code-based prompts to interact with environments. Notably, **ReAct is the base agent of modern state-of-the-art approaches**. Our method, StateAct, falls in the category of base agents and is therefore most comparable to ReAct.

2.2 Extensions of base agents: RAG, fine-tuning, test-time-scaling

ExpeL (Zhao et al., 2023) extend ReAct by using additional training data to generate success trajectories during training. At inference time, they look up the closest success trajectories as few-shot examples to the agent. Follow-on work, AutoGuide (Fu et al., 2024), uses ReAct as the base agent with additional training data to create state-aware text-based guidelines. This 'knowledge' of the environment is then used with retrieval augmented generation (RAG) to guide the decision-making process. While AutoGuide achieves the best result among RAG approaches, the complexity of the setup makes it less scalable in practice.

Chen et al. (2023); Yao et al. (2023c) introduce fine-tuning of ReAct with marginal improvements. KnowAgent (Zhu et al., 2025) compiles knowledge of the environment and distills this into the LLM to produce better results. The best approach, ActRe (Yang et al., 2024), achieves successful fine-tuning of ReAct by annotating successful trajectories with CoT before fine-tuning. While fine-tuning yields the best performance for a specific task, it also requires additional training, which is costly, and does not allow for generalisation.

ADaPT (Prasad et al., 2024) and THREAD (Schroeder et al., 2024) use test-time computation to achieve better results. This approach is in line with works such as tree-of-thought (Yao et al., 2023b), where the AI agent proposes several thoughts or actions in one go during test-time inference and another model (often the same LLM) evaluates these; the top k promising thoughts or actions are then expanded upon in a beam-search manner. These methods produce strong results and are very easy to setup, albeit they often require more compute budget compared to base agents. In our work we combine ADaPT with StateAct.

2.3 Alternative methods: tools, hand-crafted rules, multi-agent

ProgPrompt (Singh et al., 2022) and follow-on work AdaPlanner (Sun et al., 2023) introduce code-based prompts (Li et al., 2023a). They use code-execution as an additional tool, by executing LLM-generated code and feeding the results into the next LLM generation. The shortcoming of such code-based prompts is that they require human experts to annotate very long prompts[3]. This can be hard to scale to new environments and requires an additional step of code-execution.

StateFlow (Wu et al., 2024) uses Finite State Machines (FSMs) combined with LLMs to solve Alfworld. These FSMs are human expert-crafted

[3]For example, in the case of ReAct or StateAct the prompt for Alfworld can be annotated by most humans quite easily; while annotating it using the AdaPlanner paradigm would require the human to know python programming.

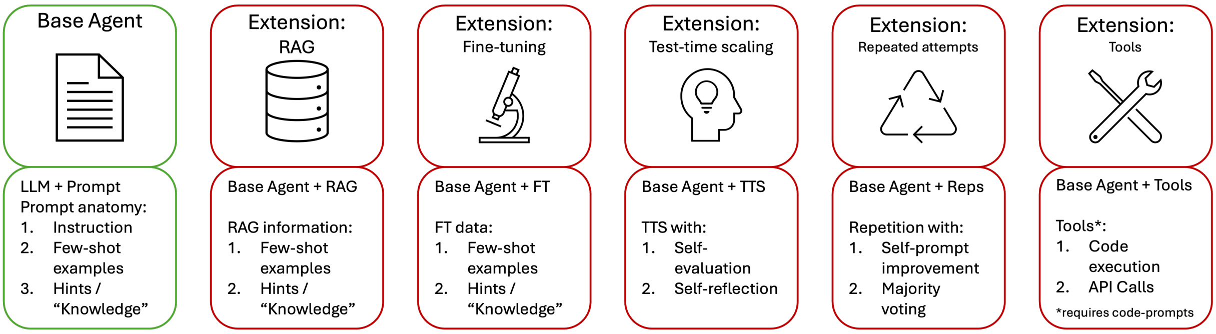

Figure 1: Overview of LLM-agent methods and their components.

states, transitions and rule-based heuristics, where the LLM is asked to perform limited tasks in each of the given states. While this method can achieve a very high score, it is limited, as it requires a human expert to design the FSM.

Another approach is to use multiple LLMs to 'chat' to one another to produce a result. So far, multi-agent frameworks Wu et al. (2023) only achieve minor improvements over using a single agent.

2.4 State tracking in LLM-based agents

Chen et al. (2024) propose state-tracking as a way to help the agent solve the task without training data. The difference of their method to StateAct is two-fold. Firstly, they employ a complex sequence of components working together, which are an LLM-based attention over the observations, an LLM-based compilation of a complex state and a prediction of a program. Secondly, their system involves execution of actual programs. StateAct, on the other hand, requires a straightforward extension of CoT and uses a single LLM call to produce the state, thought and action. Additionally, it does not require program execution. Statler (Yoneda et al., 2024) also introduce state-tracking for LLM agents. While the state has similarity to StateAct, there are notable differences. Firstly, Statler is aimed at lower level robotics execution and works with domain specific functions. Secondly, Statler produces and requires code to update and read from the state. This complex construction of the state is difficult to scale to new environments.

3 Method

StateAct is an LLM-based AI agent that works on top of pre-trained LLMs. It 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.

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. 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. This renders StateAct similar to ReAct and therefore an easy replacement for extension methods.

3.1 Self prompting

We found that with long input sequences and multiple turns LLMs can get distracted and lose track of their main goal. One of the key ideas of StateAct to overcome this 'haystack' challenge for long prompts and therefore long horizon problems (Coelho et al., 2024) is to introduce a mechanism for the LLM to pass an instruction to itself. By having the language model remind itself the goal at every turn, this objective is brought into recent context and reinforced through repetition.

To make it work in practice, instead of copying the goal manually, we teach the model to repeat the goal (or summarise it) by showing it few shot examples.

This approach can be applied in many settings and to alleviate various problems (including goal reminding and formatting). In our setting, we focus on keeping the Agent on track with the main goal and so the LLM reminds itself of this goal at every turn.

3.2 Chain of states as state-tracking

The idea of state-tracking is to introduce 'structured thoughts' into the reasoning part of the LLM Agent, specifically by giving the agent small intermediate predictions that can be inferred from the environment and actions. This method is different to existing methods such as ReAct (Yao et al., 2023c), where CoT is taken to mean verbal 'thoughts'. The inspiration comes from the original CoT paper (Wei et al., 2023) where the LLM is tasked with producing intermediate calculations. For StateAct, the LLM agent is tasked with predicting very specific intermediate steps, such as the current location or the inventory of the agent.

3.3 Formalising StateAct

Let us denote by π 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 $[o_t, a_{t-1}, ..., a_0, o_0]$.

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

Where a_t is the action produced by the agent at time step t and o_t is the observation produced by the environment at time 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., 2023c), we introduce the *context* vector, c_t. The context vector contains the action, as well as the additional predictions of the agent , i.e. $c_t = (g_0, s_t, r_t, a_t)$. Where g_0 is the goal and it 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:

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

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

3.4 An example trace

An example of the trace for step *n* in Alfworld is:

Observation n:
```
On the countertop 2, you see a tomato 1.
```

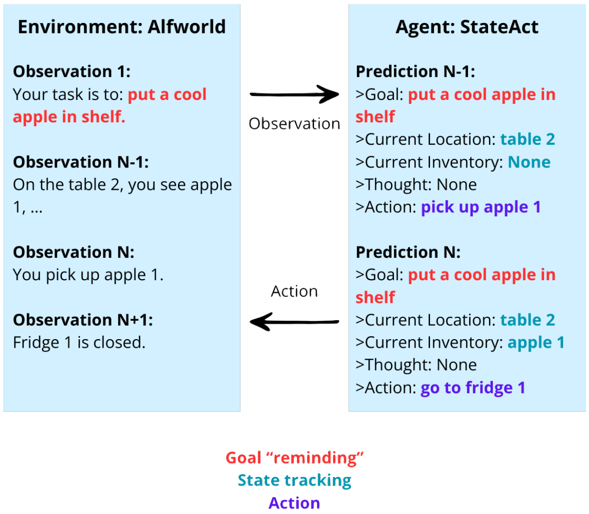

Figure 2: Example of StateAct outputs in Alfworld.

LLM Output n:
```
>Goal: Put a clean tomato in fridge
Current Location: countertop 2
Current Inventory: None
Locations Visited: table 1, countertop 1
Thought: I find the tomato, now I need to
take it.
Action: take tomato 1
```

In this example, the agent produces the action '`take tomato 1`' in response to the observation '`On the countertop 2, you see a tomato 1`'.

4 Experimental setup

Our aim is to study reasoning capabilities of LLM-based agents. In line with previous work, we use Alfworld (Shridhar et al., 2021), a household robotics environment; Webshop (Yao et al., 2023a), an online shopping environment; and Texcraft (Prasad et al., 2024), a text-based tool crafting game, as our evaluation environments. We mainly evaluate StateAct as the base agent. However, we also show that StateAct can be used as a **drop-in replacement for ReAct** and validate it on test-time scaling. RAG (AutoGuide) and fine-tuning (ActRe)-based extensions are left for future work, as these are expensive and complex to run due to training on additional data and use of additional tools.

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

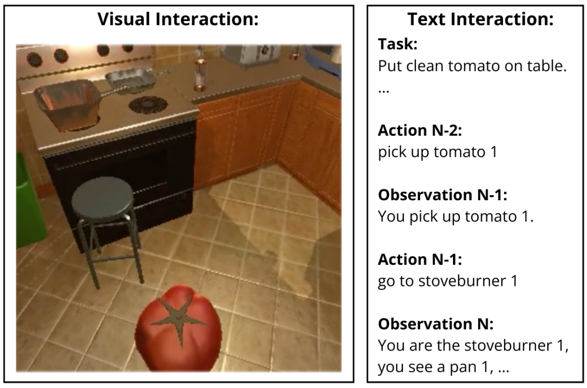

Figure 3: An example textual interaction in Alfworld (right) and corresponding 3D rendering (left).

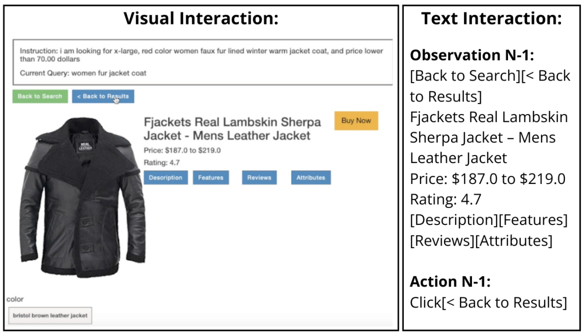

Figure 4: An textual interaction in Webshop (right) and correspondingwebsite rendering (left).

language-based AI models, see Figure 3. Alfworld has a total of 134 test-set examples and six environment types. It features long-time horizons, partial observability, an out-of-distribution evaluation set 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 either a textual observation or the 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 the commands and details of the environments.

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, 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 100 examples and each one is of the type “search and buy a product”. Overall, Webshop has a maximum of fifteen steps and two commands: 1. `search[<query>]`, 2. `click[<button>]`.

4.3 Textcraft

Textcraft (Prasad et al., 2024) is an environment based on the popular game Minecraft, where the task of the agent is to craft items. The environment is fully text-based. We use prompts and implementations based on the ADaPT paper (Prasad et al., 2024) that introduced this environment. We also use this environment to analyse whether StateAct performs well in combination with other methods such as ADaPT. See Appendix C for more details.

4.4 In-context learning

Since ReAct (Yao et al., 2023c) forms the underlying agent for many current (Zhao et al., 2023) and state-of-the-art approaches (Fu et al., 2024; Yang et al., 2024; Prasad et al., 2024), we use the same few-shot interaction traces as ReAct. The main reason is to have a fair comparison and to isolate additional effects, such as performance change, from different in-context examples. Alfworld, for example, 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 tokens to 935 tokens). For our study, we reuse the observations, thoughts and actions, and annotate these examples further with the goal and state, 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. 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, the one-shot example from ReAct for Webshop and the few-shot prompts of ReAct from ADaPT for Textcraft in all our experiments.

4.5 Models

In our work, we compare our method using newer state-of-the-art models, architectures and sizes to show that our method generalises. Specifically, we use Mistral-Small-24B-Instruct (Jiang et al., 2023), Qwen-2.5-7B,14B,32B-Instruct (Qwen et al., 2025) and Gemma2-27B-Instruct (Team et al., 2024). We note that api-based models such as OpenAI's models are generally very expensive[4] and lack rigorous reproducibility standards[5]. Nonetheless, we include experiments using `gpt-3.5` and `gpt-4o-mini`. We use temperature 0 for all experiments and sample only the top 1 response; we use vllm for inference (Kwon et al., 2023), see Appendix D & E for details.

Method	Score
State-of-the-art methods	
AdaPlanner (Code-prompt + exec.)[1]	0.75
AutoGuide (ReAct + RAG)[2]	0.79
ActRe (ReAct + fine-tuning)[3]	**0.83**
ADaPT (ReAct + test time scaling)[4]	0.72
Base agents	
Act (few-shot only)	0.41
ReAct (few-shot only)	0.64
AdaPlanner (Code-prompt only)[1]	0.45
StateAct (**ours**, few-shot only)	**0.77**

Table 1: Results on the 134 test samples of Alfworld using gpt-3.5. ReAct and StateAct scores are single run with greedy decoding and gpt-3.5-1106. [1]=code-execution (Sun et al., 2023), [2]=(Fu et al., 2024), [3]=(Yang et al., 2024), [4]=(Prasad et al., 2024).

5 Results

We present results of state-of-the-art methods such as ActRe and ADaPT, as well as the underlying base agent ReAct. In Table 1 we see that while methods that rely on ReAct + extensions, such as ActRe and AutoGuide outperform StateAct overall, they also rely on additional training data and computation. Furthermore, StateAct achieves the best result among base agents, outperforming ReAct between 7% and 30%. Additionally, StateAct achieves comparable result with the state-of-the-art methods, while not using any additional tools or data. For example, AutoGuide, a best-in-class method, uses ReAct and RAG and gets 0.79, while StateAct gets 0.77.

5.1 Base agent comparison

Since ReAct is the previous best base agent and forms the basis of state-of-the-art approaches, we compare against it in detail. The results in Table 2 show that StateAct outperforms ReAct across three different benchmarks and five different models. Sometimes the difference is substantial, with StateAct outperforming ReAct by more than 10 points. Across all 15 experiments only in two ReAct performs better than StateAct: in Webshop, where the 7B model is likely overwhelmed by the amount of textual input it receives, as Webshop has a lot of verbal input; and in the case of Gemma2-27B ReAct slightly outperforms StateAct on Alfworld; our hypothesis is that Gemma has a limited context length of 8192, while Alfworld requires long traces due to the longer step length.

We further validate our results with additional LLMs that are too large to fit on a single GPU or are closed source. In Table 3 we see a significant performance increase. Using gpt-3.5, gpt-4o-mini and Mixtral-8x22B[6] (Jiang et al., 2024) on Alfworld, ReAct achieves 63.7, 68.15 and 72.59, while StateAct achieves 77.04 (+13.3), 71.85(+3.7) and 83.70(+11.2) respectively.

5.2 Base agents + test time scaling

An important contribution is to validate that StateAct can be used as a drop-in replacement with advanced methods. To this end we validate StateAct using test-time scaling using the ADaPT method. Starting with ADaPT, which is based on ReAct, as the starting point(Prasad et al., 2024), we enhance their method using StateAct. In Table 4 we can clearly see that StateAct scales well with test time scaling jumping in performance by 39% and surpassing ADaPT+ReAct by 12%.

5.3 Summary of results

StateAct establishes itself as the best-performing base agent, surpassing ReAct by 7–30% across multiple benchmarks while requiring no additional training data or external tools. Although advanced methods like ActRe and AutoGuide achieve higher scores, they rely on costly training and retrieval. We also validate StateAct as a drop-in replacement

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

[5]Since we do not have access to weights and inference settings and models become regularly deprecated.

[6]Mixtral-8x22b-instruct-v0.1 was queried using Nvidia's NIM API `https://developer.nvidia.com/nim` [Last Accessed March 2025].

Agent Name	Mistral-24B	Qwen-7B	Qwen-14B	Qwen-32B	Gemma-27B	Average ↑
Alfworld*						
ReAct	0.44	0.10	0.75	0.89	**0.71**	0.58
StateAct	**0.49**	**0.46**	**0.78**	**0.90**	0.68	**0.66**
Webshop**						
ReAct	0.34	**0.19**	0.22	0.27	0.26	0.26
StateAct	**0.35**	0.12	**0.33**	**0.32**	**0.29**	**0.28**
Textcraft***						
ReAct	0.33	0.02	0.31	0.31	0.18	0.23
StateAct	**0.40**	**0.04**	**0.37**	**0.40**	**0.34**	**0.31**

Table 2: Base agent performance across different models and environments. *=134 Test Environments from Alfworld. **=100 Test Environments from Webshop. ***=100 Test Environments from Textcraft. M=Mistral-Instruct-2501, Q=Qwen2.5-Instruct, G=Gemma 2-Instruct. We use greedy decoding (temperature=0).

Method	Model	Success Rate %
ReAct	Gpt-3.5	0.64
StateAct	Gpt-3.5	**0.77**
ReAct	Gpt-4o-mini	0.68
StateAct	Gpt-4o-mini	**0.72**
ReAct	Mixtral-8x22B	0.73
StateAct	Mixtral-8x22B	**0.84**

Table 3: Results on the 134 test examples from Alfworld. Results are single run and greedy. Models used: gpt-3.5-1106, gpt-4o-2024-07-18, mixtral-8x22b-instruct-v0.1.

Model/Agent Name	Normal	+ADaPT
Mistral-24B		
ReAct	0.33	0.53
StateAct	**0.40**	**0.64**
Qwen2.5-7B		
ReAct	0.02	0.09
StateAct	**0.04**	**0.11**
Qwen2.5-14B		
ReAct	0.31	**0.55**
StateAct	**0.37**	0.53
Qwen2.5-32B		
ReAct	0.31	**0.64**
StateAct	**0.40**	0.62
Gemma-2-27B		
ReAct	0.18	0.19
StateAct	**0.34**	**0.35**
Average		
ReAct	0.23	0.4
StateAct	**0.31**	**0.45**

Table 4: Comparison of test-time scaling performance on 100 test samples from Textcraft. Normal refers to using just the base agent. +ADaPT means running the respective base agent with test time scaling using the ADaPT method. ADaPT code is adapted to run ReAct and StateAct. We use greedy decoding and d_{max} = 2.

for ReAct in test-time scaling. These results highlight the performance and usability of StateAct as a robust foundation for LLM-based agents.

6 Analysis and Ablations

In the results section, we discovered that our methods outperform the previous state-of-the-art base agent, ReAct. StateAct shows strong performance with in-context learning without resorting to additional tools or data. In this section, we analyse our results further and also show that *self prompting* and *state-tracking* help with long-range reasoning. For most ablation studies, we focus on Alfworld as it has two favourable properties over Webshop and Textcraft. Firstly, Alfworld has a longer time horizon (50 steps vs. 15 in Webshop, 40 in Textcraft). Secondly, Alfworld is more realistic than Textcraft, as Alfworld is a robotics environment, while Textcraft is based on a game. For completeness, we include ablation results in Table 6 and full results in Appendix K,L,M.

6.1 Does self-prompting help with long-range tasks?

For this purpose, we compare the original ReAct (thought + action) with the self-prompting included, 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 steps, the goal tracking has better relative performance as the number of steps increases and is able

Method	Avg. Steps ↓
ReAct	31.49
StateAct (goal, state, thought)	**19.11**
- w/o goal (self-prompt)	20.09
- w/o state	22.50
- w/o thought	23.76

Table 5: Average number of steps (Avg. Steps) [lower is better] on the test set of Alfworld, using gpt-3.5-1106.

to solve longer tasks of 40 to 50 steps. This finding is in line with our original motivation, that LLM agents deteriorate in performance as the prompts and interactions get longer.

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[7] and StateAct. Table 5 clearly shows that ReAct, with an average of 31.49 steps to solve an environment, is the least efficient whilst StateAct, with an average of 19.11 steps to solve an environment, is the most efficient. This shows that not only does self-prompting help with longer range tasks, it also helps with efficiency, by shortening the tasks. See Appendix G for more discussion.

6.2 What effects does state-tracking have?

We also analyse whether state-tracking helps with long-range reasoning and efficiency. We compare the full StateAct against StateAct without state-tracking, as well as comparing ReAct (thought + action) against StateAct with state-tracking (state + thought + action). In Figure 5 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 both state tracking and goal-tracking longer-range tasks can be solved. Concretely, ReAct has 24/134 examples that are in the bucket '40-50' and solves 0, while StateAct has 6/134 examples and solves 4. Also, looking at Table 5 we see that state-tracking makes the model the most efficient[8] with StateAct being almost twice as fast to solve problems as ReAct. Therefore, we find that explicit state-tracking helps with long-range tasks and to solve the tasks more efficiently.

[7]We ignore 'thought' turns for ReAct as otherwise ReAct would have even more steps.

[8]Despite StateAct using a twice-longer prompt, our cost remains similar to ReAct, at around $8 for the full Alfworld run, since we solve tasks more efficiently and use fewer steps.

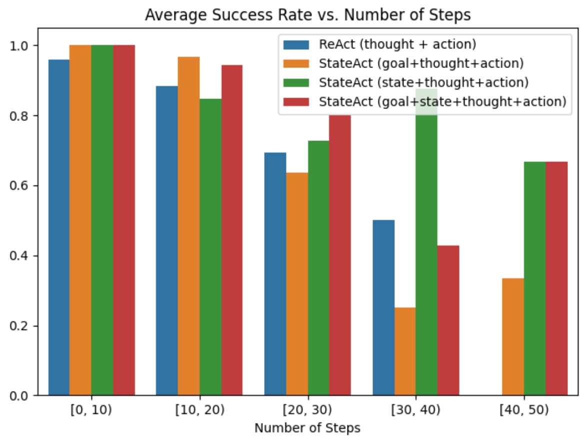

Figure 5: State vs. No State, on the 134 test examples from Alfworld, using gpt-3.5-turbo-1106

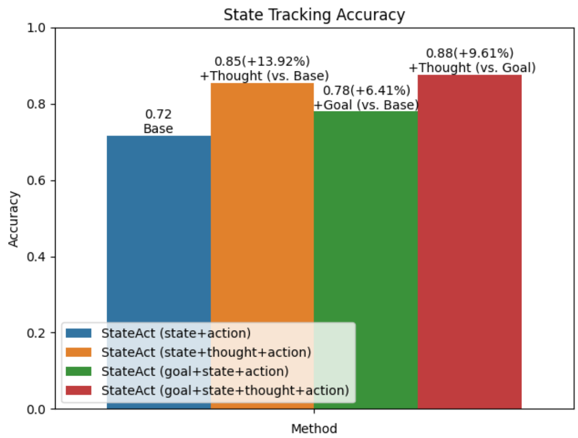

Figure 6: State-tracking accuracy for StateAct on 134 test examples of Alfworld, using gpt-3.5-1106.

6.3 Does the model perform actual state-tracking?

We investigate whether 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[9] based on the actions the agent takes. For example, if the agent produces the action `go to fridge 1` 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 6 shows that StateAct achieves a state-tracking accuracy of 88%. We also observe that thoughts and goals help the state-tracking.

6.4 Do 'thoughts' help?

While 'verbal' 'thoughts' (i.e. thoughts from ReAct) are mainly helpful in Alfworld and Textcraft.

[9]See Appendix F for details.

Surprisingly, we see that in Webshop thoughts actually harm overall performance across different agent and model types, see Table 6. Our hypothesis is the verbosity of the Webshop environment is confusing for the model if prompted with verbal thoughts.

Agent Name	**AW**	**WS**	**TC**
Baselines			
Act	0.51	0.19	0.27
Thought+Act (ReAct)	0.58	0.26	0.23
Our Methods			
State+Act	0.51	**0.31**	0.22
State+Thought + Act	0.58	0.18	**0.34**
Goal+Act	0.63	0.24	0.26
Goal+Thought + Act	0.65	0.18	0.30
Goal+State+Act	0.56	0.29	0.21
Goal+State+Thought+Act	**0.66**	0.28	0.31

Table 6: Ablation table. Averaged results across Mistral, Qwen, Gemma on AW=Alfworld, WS=Webshop and TC=Textcraft. Goal=Self-prompt.

7 Conclusion

Our work is driven by the fundamental challenge that LLM agents struggle with long context and keeping on track with instructions. The current state-of-the-art to overcome such challenges propose extensions on top of the base agent, ReAct. In contrast, we introduced a novel base agent, StateAct, an in-context learning method that leverages *chain-of-states* and *self-prompting* to significantly enhance the capabilities of LLM agents. By rethinking how agents track and utilize state information, StateAct establishes a new state-of-the-art for base agents, surpassing ReAct by 9% to 20% across different models and tasks. Furthermore, we demonstrate that StateAct scales effectively when combined with advanced techniques such as ADaPT, reinforcing its usability.

Beyond raw performance, our analysis uncovers a crucial insight: StateAct not only improves reasoning but also enhances efficiency, allowing agents to achieve better results with fewer steps. This suggests that integrating structured state-tracking and self-prompting cues helps mitigate the well-documented long-context issue in LLMs.

Our findings suggest that methods like StateAct can serve as a practical and efficient way to improve LLM-based agents without the need for additional data, tools or other extensions. By enabling agents to manage their own state explicitly, StateAct provides a scalable approach to improving reasoning and decision-making across a range of tasks. This makes StateAct a viable drop-in solution for current systems, offering both better performance and greater efficiency in LLM agent tasks.

8 Acknowledgement

We want to thank Imperial College London Research Computing Services (RCS)[10] for the generous contribution of computational resources to this project. We also thank Kyle Richardson, Joe Stacey and Lisa Alazraki for the careful review and thoughts on the work.

9 Ethical Considerations

9.1 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 models that are less computationally demanding such as `gpt-3.5-turbo` level models or open source models that fit on a single GPU (A100, 80GB), reporting costs and minimising the cost of our method.

9.2 Hallucinations in LLMs

As LLM-based agents become more powerful and therefore more pervasive in our daily lives, 'hallucinations' in 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 to reduce 'hallucinations.'

10 Limitations

10.1 Languages and evaluation benchmarks

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

10.2 Reasoning traces that 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.

[10] https://doi.org/10.14469/hpc/2232

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.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *Preprint*, arXiv:2310.05915.

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.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

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, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

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.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

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.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. Adapt: As-needed decomposition and planning with language models. *Preprint*, arXiv:2311.05772.

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.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Philip Schroeder, Nathaniel Morgan, Hongyin Luo, and James Glass. 2024. Thread: Thinking deeper with recursive spawning. *Preprint*, arXiv:2405.17402.

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.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam

Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. 2024. Gemma: Open models based on gemini research and technology. *Preprint*, arXiv:2403.08295.

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 gold-standard 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.

Zonghan Yang, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. 2024. 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, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023b. Tree of thoughts: Deliberate problem solving with large language models. *ArXiv*, abs/2305.10601.

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

Takuma Yoneda, Jiading Fang, Peng Li, Huanyu Zhang, Tianchong Jiang, Shengjie Lin, Ben Picker, David Yunis, Hongyuan Mei, and Matthew R. Walter. 2024. Statler: State-maintaining language models for embodied reasoning. *Preprint*, arXiv:2306.17840.

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.

Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, Huajun Chen, and Ningyu Zhang. 2025. Knowagent: Knowledge-augmented planning for llm-based agents. *Preprint*, arXiv:2403.03101.

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 sink/basin) 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 stove/burner) 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 desk lamp, and then use the desk lamp.

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 side table.

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.2.1 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. Using only "in" or only "on" produces a failed command. We observed this issue with LLMs in 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>".

Note: In the latest release of Alfworld (December 2024) this was fixed by replaceing the put <obj> in/on <place> command with move <obj> to <place> command. In our work we report results on the latest version of alfworld.

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 Textcraft

C.1 Commands and environment

Textcraft has one environment type: '*craft*', as well as three commands: 1. *inventory*, 2. *craft*, 3. *get*

C.2 Crafting Recipes

Textcraft has crafting recipes that range from easy to hard. Where hardness is measured by the 'depth' of the crating recipe. Specifically, depths of 2, 3 and 4 are present in the dataset.

C.3 License

Textcraft is published under the permissible MIT license.

D Compute Requirements for local LLMs

The exact code will be released upon publication. However, to help reproducibility we ran all experiments on single A100 80GB GPUs. In terms of software we used: vLLM for inference. The hyperparameters were set to: max model length 16000 (except for Gemini, where we used 8192), temperature = 0, datatype="auto" (which results in bfloat16).

E Code

E.1 Code Snippet to call Local LLMs

```
from vllm import LLM, SamplingParams
self.llm = LLM(
    model=model,
    tensor_parallel_size=
    tensor_parallel_size,
    gpu_memory_utilization=0.95,
    max_model_len=max_model_len,
    dtype="auto"
)

messages = [
    # {"role": "system", "content": "You
     are a helpful assistant."},
    {"role": "user", "content": prompt[-
    self.max_model_len:]}
]
text = self.tokenizer.
    apply_chat_template(
    messages,
    tokenize=False,
    add_generation_prompt=True
)

sampling_params = SamplingParams(
    temperature=self.temperature,
    top_p=1.0,
    repetition_penalty=1.00,
    max_tokens=min(2000,self.
    max_model_len),
    stop = self.stop_sequences,
    seed = self.seed
)
outputs = self.llm.generate([text],
    sampling_params)
return outputs[0].outputs[0].text
```

E.2 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 J.

F Heuristic State-tracking Explained

Heuristic state tracking is based on the idea that the state can be inferred automatically if one follows the actions of the agent and observations of the environments. Specifically, the state at time t for StateAct depends on the state at time `t-1` and the action a_{t-1} and observation o_t. For example, if the state at time t the 'current location' of the state is set to `table 1` and the action is `go to fridge 1` and the observation is successful, then the 'current location' can be updated to be `fridge 1` automatically. This rule based 'state-tracking' is how the heuristics work.

G 'Step Length Analysis' Discussion

An alternative to calculating and comparing step length could be 'gold solutions' to measure optimal step length and optimality of an agent. We see two issues. Firstly, the annotation cost of creating gold solutions. Secondly, it is not clear what the gold solution should be. Concretely in Alfworld, an 'oracle' solution could have very few steps as it would immediately go to the location of the 'hidden' object, while a 'non-oracle' expert solution would have more steps as more locations would be searched. Thus a 'difficulty' measure would be needed instead, but it is ambiguous.

H Potential Future Work Directions

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

I Does JSON structure help StateAct performance?

We also investigated 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 section I.1 for more details. Surprisingly, we found that the json format hinders performance, see Table 7.

Method	SR%	SR (+json)%
ReAct	**63.70**	62.96(-0.74)
StateAct (complete)	**77.04**	58.52(-18.5)

Table 7: No-json vs. json. Success Rate (SR) on the test set of Alfworld, using gpt-3.5-1106.

I.1 JSON prompt

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"
}
```

J StateAct Prompts

J.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>
```

J.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]
```

K Additional results for Alfworld

For Alfworld we can see that the full StateAct performs best. Interstingly, adding state in does not always help model performance. A potential explanation for this can be that state-tracking itself is not the challenge in this dataset. Rather self-prompting is critical in Alfworld. Which indicates that long-range reasoning depends on self-prompting.

L Additional results Webshop

For Webshop, we present the results for ReAct and StateAct. Similarly to Alfworld, we also present the results of StateAct in different forms, see Table 9.

We can see that our method again outperform the base-agents. For most model we see big jumps in improvement using 'state' in the Webshop environment, indicating that the environment benefits from additional structured prediction.

Interestingly in Webshop the results of Act-only are very strong for many models. A hypothesis might be that strong instruction tuning introduces these kind of reasoning techniques into the models directly without the need for explicit prompting from the user. Additionally, we can see that thoughts often harm performance on Webshop. A hypothesis for this is that Webshop already has a lot of textual content so verbose thoughts can be harmful by confusing the model.

M Additional results for Textcraft

We can see that in Textcraft similar to Alfworld. The results improve when thoughts are used. Interestingly, for Textcraft self-prompting does not yield the highest result. A hypothesis for is because the crafting recipe needs to followed very closesly for successful completion therefore additional reminders are not necessary.

N Use of AI

We used coding assistants in small parts using continue.dev [11] and claude-sonnet-3.5. Small use of ChatGPT was used for Latex advise and particular phrasing of parts of the text.

[11] https://www.continue.dev/, last accessed March 2025.

Agent Name	M-24B	Q-7B	Q-14B	Q-32B	G-27B	Average
Baselines						
Act-only (action)	0.29	0.31	0.66	0.87	0.41	0.51
ReAct (thought + action)	0.44	0.10	0.75	0.89	0.71	0.58
Our Methods						
StateAct (state+action)	0.09	0.57	0.75	0.76	0.40	0.51
StateAct (self-prompt+action)	0.36	0.52	0.77	0.90	0.62	0.63
StateAct (state+react)	0.21	0.44	0.76	0.77	0.73	0.58
StateAct (self-prompt+react)	**0.53**	0.39	**0.80**	**0.91**	0.64	0.65
StateAct (self-prompt+state+action)	0.14	**0.60**	0.71	0.86	0.51	0.56
StateAct (self-prompt+state+react)	0.49	0.46	0.78	0.90	**0.76**	**0.68**

Table 8: 135 Test Environments from Alfworld. Different Columns represent different models. In **Bold**: Best & 2nd Best Solution per Model. Light Green Background: Best Solution per Model. Dark Green Background: Best Solution Overall. Decoding Strategy: Greedy (temperature=0). M=Mistral-Instruct-2501, Q=Qwen2.5; G=Gemma 2-Instruct.

Agent Name	M-24B	Q-7B	Q-14B	Q-32B	G-27B	Average
Baselines						
act-only (action)	0.32	0.26	0.16	0.11	0.11	0.19
react (thought + action)	0.34	0.19	0.22	0.27	0.26	0.26
Our Methods						
StateAct (state+action)	**0.38**	0.25	**0.37**	0.29	0.26	**0.31**
StateAct (self-prompt+action)	0.00	**0.27**	0.36	0.23	**0.35**	0.24
StateAct (state+react)	0.23	0.14	0.25	0.26	0.04	0.18
StateAct (self-prompt+react)	0.24	0.14	0.15	0.25	0.14	0.18
StateAct (self-prompt+state+action)	0.37	0.26	0.36	0.30	0.15	0.29
StateAct (self-prompt+state+react)	0.35	0.12	0.33	**0.32**	0.29	0.28

Table 9: 100 Test Environments from Webshop. Different columns represent different models. Average is the average of all models. Decoding Strategy: Greedy (temperature=0). M=Mistral-Instruct-2501, Q=Qwen2.5; G=Gemma 2-Instruct.

Agent Name	M-24B	Q-7B	Q-14B	Q-32B	G-27B	Average
Baselines						
act-only (action)	0.37	0.05	0.26	**0.42**	0.25	0.27
react (thought + action)	0.33	0.02	0.31	0.31	0.18	0.23
Our Methods						
StateAct (state+action)	0.34	0.06	0.09	0.36	0.25	0.22
StateAct (self-prompt+action)	0.42	0.01	0.25	0.41	0.23	0.26
StateAct (state+react)	**0.45**	**0.12**	**0.38**	**0.42**	**0.34**	**0.34**
StateAct (self-prompt+react)	0.41	0.09	0.35	0.37	0.29	0.30
StateAct (self-prompt+state+action)	0.29	0.01	0.16	0.35	0.23	0.21
StateAct (self-prompt+state+react)	0.40	0.04	0.37	0.40	**0.34**	0.31

Table 10: Performance across 100 Textcraft test environments with different models. Decoding Strategy: Greedy (temperature=0). M=Mistral-Instruct-2501, Q=Qwen2.5; G=Gemma 2-Instruct.