The University of Manchester

Staler: State-Manitaining Language Models for Embodied Reasoning

Takuma Yoneda[*,1], Jiading Fang[*,1], Peng Li [*,2], Huanyu Zhang [*,3], Tianchong Jiang [3], Shengjie Lin [1], Ben Picker [3], David Yunis [1], Hongyuan Mei [1], Matthew R. Walter [1]

[1]TTI-Chicago, [2]Fudan University, [3]University of Chicago, [*]Equal Contribution

Yiyu Jiang
Phd student
University of Manchester

MANCHESTER
1824
The University of Manchester

Introduction

Effective methods for difficult embodied reasoning tasks

- rely solely on the implicit in-context memory that is internal to the LLM
- augment LLMs with scene information extracted from an ego-centric image captured at the current time step

Struggle when faced with planning long time horizons tasks

- Due to limited context window of contemporary LLMs

Struggle to exploit information conveyed in long-term context

- Just improves prediction accuracy only <u>on a small number of tokens</u>
- The context beyond what can be directly copied

Prohibits LLMs from reasoning over aspects of the scene

(not directly observable)

- reliance on <u>the robot's current ego-centric </u>view prohibits the language model from reasoning over aspects of the scene that are not directly observable

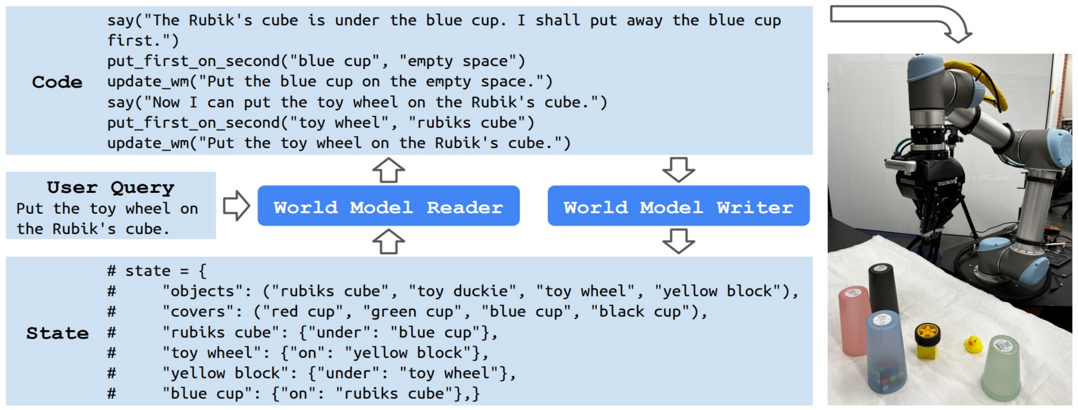
The University of Manchester

Introduction

In this paper, we propose **Statler (STATe-maintaining Language models for Embodied Reasoning)**, a framework that maintains an external world model as explicit memory to improve the long-term reasoning capabilities of LLMs for robot planning

- **World Model Reader**
  - interfaces with the world model to generate code that answers user queries
- **World Model Writer**
  - responsible for predicting the next world state based on the current world state and a query given by the reader

- **Structured representation** of the world state
- **Simulated and real-world robot manipulation** domains
  - improves the long-term embodied reasoning capabilities of LLMs
  - outperforms the current state-of-the-art



```
say("The Rubik's cube is under the blue cup. I shall put away the blue cup
first.")
put_first_on_second("blue cup", "empty space")
update_wm("Put the blue cup on the empty space.")
say("Now I can put the toy wheel on the Rubik's cube.")
put_first_on_second("toy wheel", "rubiks cube")
update_wm("Put the toy wheel on the Rubik's cube.")
```

```
# state = {
#     "objects": ("rubiks cube", "toy duckie", "toy wheel", "yellow block"),
#     "covers": ("red cup", "green cup", "blue cup", "black cup"),
#     "rubiks cube": {"under": "blue cup"},
#     "toy wheel": {"on": "yellow block"},
#     "yellow block": {"under": "toy wheel"},
#     "blue cup": {"on": "rubiks cube"},}
```

Figure 1: Our Statler framework enables robots to carry out complex tasks specified in natural language that require reasoning over long time horizons. Integral to our model are its world model writer and world model reader, two instances of general LLMs that are responsible for maintaining the explicit world state and generating code that enables the robot to carry out the task.

MANCHESTER 1824

The University of Manchester

Motivational Example

Three-cups-and-a-ball **version of the classic shell game**

```
# Initial state
cups = [False, True, False]
Swapping cup 1 with cup 2
Swapping cup 0 with cup 2
Swapping cup 1 with cup 2
cups = [True, False, False]
```

Prompt 1: The prompt and desired output of a vanilla LLM.

```
# Initial state
cups = [False, True, False]
Swapping cup 1 with cup 2
Swapping cup 0 with cup 2
Swapping cup 1 with cup 2
cups = [False, False, True]
cups = [True, False, False]
cups = [True, False, False]
```

Prompt 2: The prompt and desired output of an LLM w/ CoT.

```
# Initial state
cups = [False, True, False]
Swapping cup 1 with cup 2
cups = [False, False, True]
Swapping cup 0 with cup 2
cups = [True, False, False]
Swapping cup 1 with cup 2
cups = [True, False, False]
```

Prompt 3: The prompt and desired output of an LLM w/ state.

Evaluation

- 30 demonstrations swaps
- 100 episodes
- Result
  - Vanilla LLM: highlights the difficulty of maintaining the world sate implicitly in LLMs
  - LLM w/CoT: performs better, also experiences a pronounced decrease
  - LLM w/State:
    - Decreases far more gradually
    - Retaining more than 75%(absolute and relative accuracy after 5 rounds of swaps)

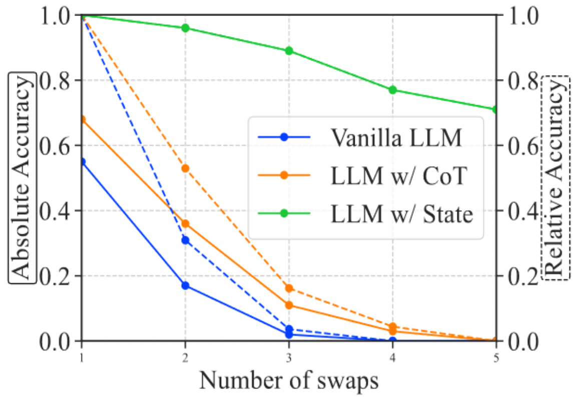
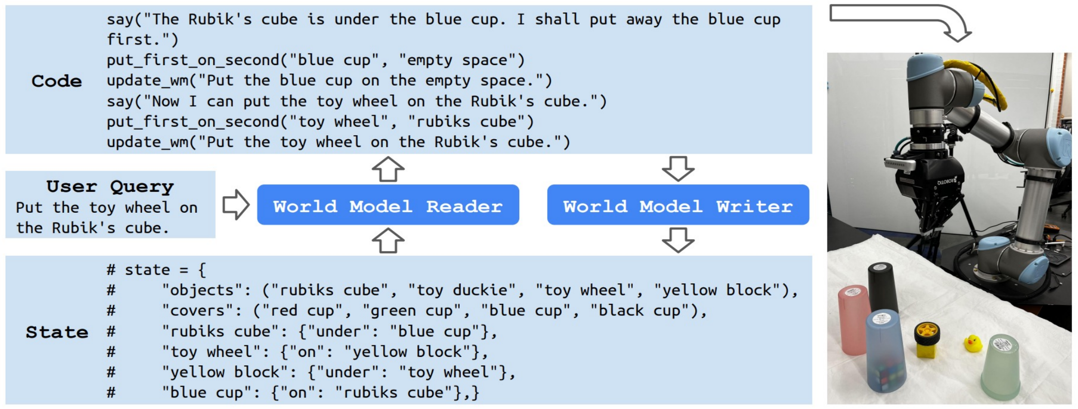


Figure 2: The accuracies of different methods for different numbers of swaps in the three-cups-and-a-ball shell game. LLM w/ State is a simplified version of our proposed Statler framework. For each method, the solid line shows how its accuracy $a(n)$ changes with the number of swaps n. The dashed line is the *relative* accuracy: $r(n) = a(n)/a(1)$. Intuitively, it measures how fast the performance decreases from a *hypothetically perfect* one-swap performance. Note that LLM w/ State indeed achieves $a(1) = 100\%$.

The University of Manchester

Method

Key: allow the LLM to describe the next state while responding to each user query



```
say("The Rubik's cube is under the blue cup. I shall put away the blue cup
first.")
put_first_on_second("blue cup", "empty space")
update_wm("Put the blue cup on the empty space.")
say("Now I can put the toy wheel on the Rubik's cube.")
put_first_on_second("toy wheel", "rubiks cube")
update_wm("Put the toy wheel on the Rubik's cube.")
```

```
# state = {
#     "objects": ("rubiks cube", "toy duckie", "toy wheel", "yellow block"),
#     "covers": ("red cup", "green cup", "blue cup", "black cup"),
#     "rubiks cube": {"under": "blue cup"},
#     "toy wheel": {"on": "yellow block"},
#     "yellow block": {"under": "toy wheel"},
#     "blue cup": {"on": "rubiks cube"},}
```

Figure 1: Our Statler framework enables robots to carry out complex tasks specified in natural language that require reasoning over long time horizons. Integral to our model are its world model writer and world model reader, two instances of general LLMs that are responsible for maintaining the explicit world state and generating code that enables the robot to carry out the task.

- Inspired by the concept of modularity, we propose to **split the burden across multiple different prompted LLMs**
    - **Maintain a separate prompt** that **includes instructions and demonstrations** for each subtask (state tracking or query responding) and then use the prompt to elicit an LLM to perform the particular subtask
    - Includes
        - World-model reader
        - World-model writer
    - **Not pose any limitation** on what **domain** it can be applied to, **or how many number of subtasks** there are

The University of Manchester

Method

```
# state = {
#     "objects": ["cyan block", "yellow block", "brown block", "purple block", "blue block", "green
bowl", "red bowl", "disinfector"],
#     "relations": [],
#     "disinfector": {"contains": []},
#     "cyan block": {"is": ["dirty"]},
#     "yellow block": {"is": ["clean"]},
#     "brown block": {"is": ["clean"]},
#     "purple block": {"is": ["dirty"]},
#     "blue block": {"is": ["clean"]},
#     "green bowl": {},
#     "red bowl": {}
# }
# query: Put the cyan block on the yellow block
put_first_on_second("cyan block", "yellow block")
update_wm("Put the cyan block on the yellow block")
```

Prompt 4: world-model reader. The text highlighted in green represents the part that the model is expected to generate.

```
# state = {
#     "objects": ["cyan block", "yellow block", "brown block", "purple block", "blue block", "green
bowl", "red bowl", "disinfector"],
#     "relations": [],
#     "disinfector": {"contains": []},
#     "cyan block": {"is": ["dirty"]},
#     "yellow block": {"is": ["clean"]},
#     "brown block": {"is": ["clean"]},
#     "purple block": {"is": ["dirty"]},
#     "blue block": {"is": ["clean"]},
#     "green bowl": {},
#     "red bowl": {}
# }
# query: Put the cyan block on the yellow block.
# state = {
#     "objects": ["cyan block", "yellow block", "brown block", "purple block", "blue block", "green
bowl", "red bowl", "disinfector"],
#     "relations": [["cyan block is on yellow block"]],
#     "disinfector": {"contains": []},
#     "cyan block": {"is": ["dirty"]},
#     "yellow block": {"is": ["dirty"]},
#     "brown block": {"is": ["clean"]},
#     "purple block": {"is": ["dirty"]},
#     "blue block": {"is": ["clean"]},
#     "green bowl": {},
#     "red bowl": {},
# }
```

Prompt 5: world-model writer. The text rendered in blue highlights the updated part of the state.

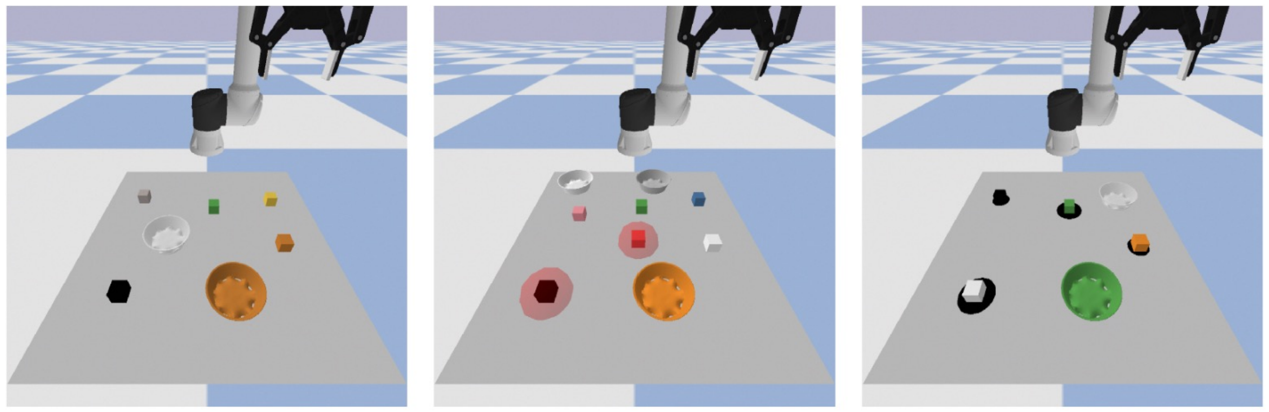
MANCHESTER 1824

The University of Manchester

Experiments

Table-top Manipulation Domains

- 20 evaluation episodes
- Each episode consists of between 5 and 16 consecutive steps of user queries



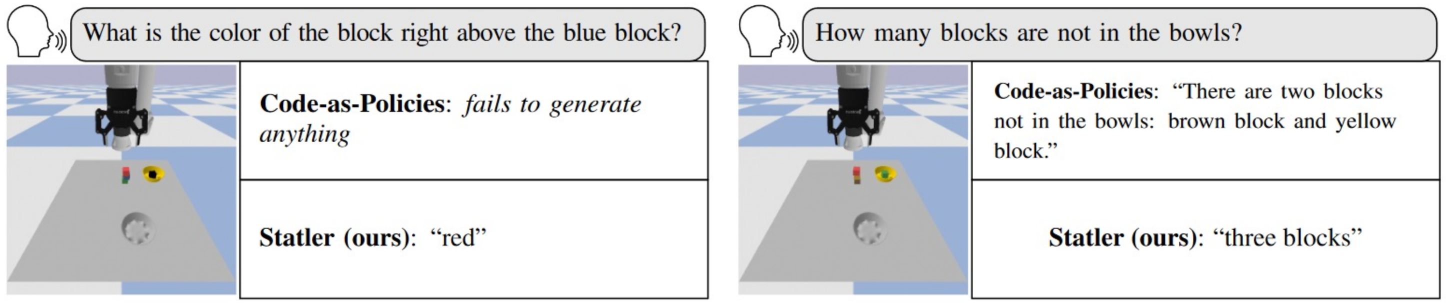
(a) Pick-and-place (b) Disinfection (c) Weight reasoning

Figure 4: The simulated domains we consider include (a) pick-and-place; (b) block disinfection, where the translucent sphere around a block represents its dirtiness (this is not visible to the robot); and (c) relative weight reasoning, where the radius of the disk under each block provides an indication of its weight. These disks are rendered there only for visual aids.

Experiments

Table 1: Number of successful steps until failure (normalized by episode length) and the success rate for each domain.

	Simple Pick-and-Place		Block Disinfection		Rel. Weight Reasoning	
	successful steps	success rate	successful steps	success rate	successful steps	success rate
Code-as-Policies	0.54	0.00 (0/20)	0.68	0.00 (0/20)	0.84	0.00 (0/20)
Statler (ours)	**0.88**	**0.50** (10/20)	**0.82**	**0.40** (8/20)	**0.93**	**0.55** (11/20)


What is the color of the block right above the blue block?

Code-as-Policies: *fails to generate anything*

Statler (ours): "red"

How many blocks are not in the bowls?

Code-as-Policies: "There are two blocks not in the bowls: brown block and yellow block."

Statler (ours): "three blocks"

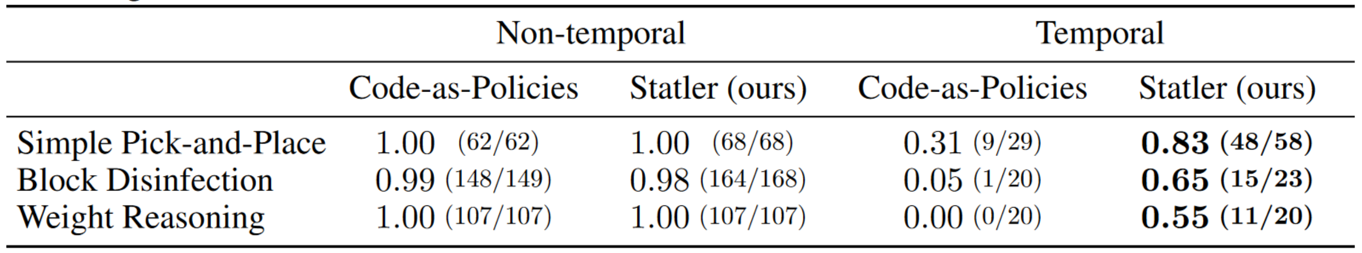
Figure 5: Examples that show the result of querying language models with and without state maintenance for the environment depicted in the image. In the scenario depicted on the left, a standard language model fails to produce an answer, while our state-maintaining language model produces the correct response. On the right, one of the blocks is currently not visible and so a standard language model (Code-as-Policies) incorrectly identifies two blocks as not being in the bowls. By maintaining a persistent model of the world, our method is aware of the third block and correctly answers the query.

Experiments

Code generation based on the type of textual utterance

- Aligning the set of queries evaluated by both of the models

Table 2: Success rates of Code-as-Policies and Statler for non-temporal and temporal queries, truncating at the first failure of each model.



	Non-temporal		Temporal	
	Code-as-Policies	Statler (ours)	Code-as-Policies	Statler (ours)
Simple Pick-and-Place	1.00 (62/62)	1.00 (68/68)	0.31 (9/29)	**0.83 (48/58)**
Block Disinfection	0.99 (148/149)	0.98 (164/168)	0.05 (1/20)	**0.65 (15/23)**
Weight Reasoning	1.00 (107/107)	1.00 (107/107)	0.00 (0/20)	**0.55 (11/20)**

Model is not without errors:

- Hallucinates **block conditions or locations**
- Model's reasoning strategy **predominantly focus on evaluating the weight relationships between blocks**
- **Struggle to Comprehend ambiguous terms** like "other" in queries

Real Robot Experiments

- We use **MDETR, an open-vocabulary segmentation model**, to obtain segmentation masks for objects from an RGB camera on the gripper
- The **difficulty** is in **recognizing that the black cup must be removed** in order to move the yellow block, which Statler correctly spots.

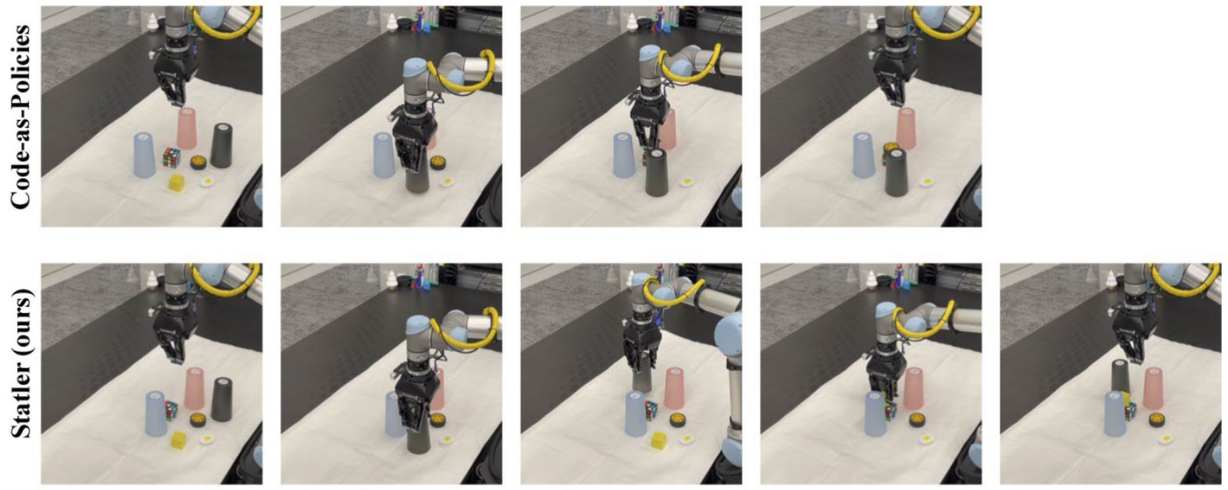


Figure 6: A comparison of the resulting behavior for (top) Code-as-Policies and (bottom) our Statler model for the real robot experiments given the multi-sentence instruction "Put the black cup on the yellow block. Put the yellow block on the Rubik's cube." Frames are arranged with time increasing from left to right, and correspond to instances when the robot has placed a (possibly imaginary) object. In order to successfully carry out the instruction, the robot must remove the black cup after placing it above the yellow block in order to place the block on the Rubik's cube. However, the the baseline Code-as-Policies (top row, third frame) fails to move the black cup aside, leaving the yellow block covered, and instead places an imaginary object on top of the Rubik's cube.

MANCHESTER 1824
The University of Manchester

Limitations

Limitation 1: Hand individually world models for each task

- Ideally there should be **an automatic way of generating** it, maybe from the LLMs themselves

Limitation 2: Current world models are still purely text-based

- It does **not directly reason about visual information**
- How it will work out when multi-modal models are accessible

Limitation 3: Issues in execution the updated state will be incorrect

- Assume that the generated code executes successfully
- This could be alleviated by **providing some feedback from external modules** such as image captioning models

MANCHESTER
1824
The University of Manchester

Conclusion

- We presented **Statler, a state-maintaining language model that consists of a world-model reader and a writer**
- Our model does **not pose any limitations in how the state representation** should be formatted, as long as it is represented in the form of a string, leaving some space for flexibility in its design
- We **evaluated our approach on various simulated and real tasks**. The experimental results suggest that our approach **effectively maintains state representation and handles non-trivial reasoning over the past steps**, whereas the baseline approach (Code-as-Policies) fails to generate correct code on such queries
- having separate models suggests that it may be possible to **use a lightweight language model for some components**