Hello everyone, the paper I'm sharing today is "WALL-E: World Alignment by Rule Learning Improves World Model-Based LLM Agents". It was submitted to arXiv in October of this year. The authors are from Tencent, the University of Technology Sydney, and the University of Maryland.

Simply put, the goal of this paper is to enhance the ability of agents to complete tasks in Minecraft or Alfworld scenarios. Because previous LLM agent methods were model-free, there were gaps between the prior knowledge of LLMs and the dynamics of the specified environment. Therefore, they used LLM and rules to serve as a world model, and trained this world model to align it with the Minecraft world or alfworld.

Specifically, their approach is built upon model-predictive control (MPC). They train the world model on the training set using LLMs to learn rules in a gradient-free manner, which involves inducing, updating, and pruning rules. Then, the agent plans the action with the maximum reward based on the predictions of the world model. Finally, this method achieves higher success rates, with lower costs.

I will explain this process in detail in the methods section.

The entire method consists of three parts: the overall framework of model-predictive control (MPC), the method of world alignment, and inference on LLM Agents with learned rules.

The part about Model-Predictive Control is similar to reinforcement learning, where the world model is used to predict the next state after the selected action is executed at the current state. The best action the agent should choose is the one that maximizes the reward function. The rewards further in the future are diminished by a discount factor. Therefore, the accuracy of the world model is crucial for selecting the best action.

The process of world alignment consists of five steps, which are Comparing Predicted and Real Trajectories, Learning New Rules, Refining Learned Rules, Translating Rules to Code and Rule set Pruning. Steps (1) and (5) are executed by programs, and others are llm.

The state specifically consists of the action result and state info. The action result is binary, indicating success or failure. Based on the results of the action result, all transitions can be classified into correct and incorrect sets. Afterwards, they can highlight areas needing correction.

Next, they generate new natural language rules using the LLM based on the previous natural language rules and the Real Transition Set. The Transition Set also contains some informative hints in the form of feedback and suggestions.

The union of the new rules and the old rules has some redundancies. We further refine these using the LLM to obtain the final set of natural language rules, Rnl.

Then, the natural language rules are used to generate a code-based rule set with the help of the LLM. The inputs are the state and action, and the return values are feedback, success, and suggestions. Therefore, a single rule can contain many transitions. And if the rule involves the need to use llm’s knowledge to make a judgement about an item or action, it will generate the function 'llm request'.

To address the inherent uncertainty and variability in the LLM-driven rule-learning process, they prune the rule set through a combination of rule-based and hard-coded approaches, selecting certain rules to be discarded. Ensure that any rule that fails to predict a transition correctly is discarded.
Here is the objective function, and here is the meaning of the variables: where the constraint represents that each incorrect transition should be covered by at least one selected rule. This term represents the number of rules selected, with the aim to avoid choosing too many rules, in order to keep the rule set succinct. This term represents the number of incorrect transitions, with the aim to cover as many incorrect transitions as possible. They are balanced through a coefficient.
In math, this Maximum Coverage problem is NP-hard, meaning it's computationally difficult to solve exactly. And they use greedy algorithm for an approximate solution.

the agent will predict the next state based on both Natural Language rules and Code-based rules. If these predictions contradict each other, the function ApplyRules acts as a verification layer, overriding the LLM's prediction. This helps in achieving accurate reward evaluations of the world model and high-quality feedback from the agents. When an action is predicted to fail, the feedback information about the failure and the provided suggestions assist the agent in making better decisions.

Then the experiment part.

For the benchmark setup, they selected the techtree task in Minecraft's Minedojo, which involves crafting a series of tools. In Alfworld, this is a virtual environment designed as a text-based simulation where agents perform tasks by interacting with a simulated household environment.

They used three metrics. In the success rate, failure refers to the agent dies in Minecraft or not solving the task within the time limit. The replanning rounds represent the number of times the agent revisits the same task to revise its plan for recovering from the failed task planning. And the token cost refers to the number of tokens consumed by the LLM agent/world models during task completion.
Also, they need to train the world model by completing tasks in the training set before the agent performs actions in the test set tasks.

As the main results, they achieve higher success rate among other methods. In ALFWorld, it achieves a record of 95% success rate only after 6 iterations and in Minecraft, WALL-E surpasses baselines by 15–30%.

They also achieve lower replanning rounds. Such as in Minecraft, WALL-E costs 8–20 fewer replanning rounds.

They also achieve lower Token cost. Such as in Minecraft, WALL-E costs only 60–80% of tokens. Note that the tokens mentioned here do not include those used during the phase of obtaining the rule set through model training.

This method requires additional token costs during the world model learning phase compared to model-free methods. However, they claim that when facing complex tasks, WALL-E demonstrates remarkably high sample

efficiency, which is sufficient to offset the additional consumption caused by the world modeling. But for simpler tasks, model-free methods may achieve comparatively high sample efficiency.
Additionally, WALL-E is a general and environment-agnostic method. It can also be used in other environments to enhance an agent's capabilities of exploration, planning, and reflection in general, complex scenarios.

They also further calculated the cover rate of the LLM's failed predictions during training. The cover rate represents the probability that the LLM’s failed predictions are addressed by the rules obtained during the rule learning process. This further verifies that the learned rules enable a more accurate world model.
In addition, they compared their method with GITM, a method that employs buffered trajectories as in-context examples to align LLM agents with the environment dynamics. They found that WALL-E’s success rate hits the upper bound after 4 iterations, which proves that WALL-E’s improvement mainly benefits from the learning of new rules.

In the Ablation Study section, they conducted separate ablations on the world model and the rules. The results showed that the highest success rates were achieved by the LLM agent and LLM+rules. While the success rate of LLM+rules was also significantly improved, it was slightly lower, possibly due to the fact that the learned rules are highly related to the state information. Additionally, if the world model is solely based on the LLM, it barely improves the success rate.

That’s it for my paper sharing, thanks for your attention.