DreamFactory: Grounding Language Models to World Model through Decentralized Generation and Centralized Verification

Siqiao Huang*, **Pingyue Sheng***, **Jiahe Guo***
2023012148, 2023011017, 2023012125
Institute for Interdisciplinary Information Sciences
Tsinghua University
{huang-sq23,chengpy23,guojh23}@mails.tsinghua.edu.cn

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities in generalizability and proficiency in in-context learning. However, their potential to serve as world models remains underexplored. This project investigates the behavior of LLMs as world models, uncovering several insights into their generation process during dynamics prediction tasks. Based on these insights, we propose `DreamFactory`, a novel architecture that enhances LLMs' performance in this task through decentralized generation followed by centralized verification. The results suggest that while LLMs are not world models yet, leveraging their capabilities through structured generation and verification offers a promising pathway for unleashing their modeling capabilities. Code: `https://github.com/knightnemo/nlp-proj`.

1 Introduction

Empowering agents to learn in open and unseen environments has been a long time goal for researchers and AI-enthusiasts. As many believe, understanding and modeling the world around them, which is also referred to in literature as obtaining a **World Model** (Ha & Schmidhuber, 2018), is a key component of the learning process. While extensive work on explicitly modeling the dynamics of the world has been done in active research areas such as Model-Based Reinforcement Learning (Moerland et al., 2023), one key question is how well the learned model can generalize. Large Language Models (LLMs), on the other hand, have demonstrated remarkable capabilities in their generalizability and in-context learning (Dong et al., 2024) abilities, so the natural question arises: **How well can LLMs serve as World Models?**

From the perspective of world modeling, the ability to predict the next state given the current state and action is crucial for decision making and model predictive control(MPC) (García et al., 1989). Utilizing LLMs can serve as an approach for general world models as they are believed to possess extensive common sense knowledge and reasoning capabilities. This inherent cross-domain knowledge acquired from data during the pretraining process is believed to generalize well across different domains. More exciting still, the instruction following abilities of LLMs shown in recent models possess the potential of few-shot or zero-shot learning in dynamics modeling, which is widely considered a huge step towards general world models and even Artificial general intelligence (AGI).

From the perspective of LLMs, the ability to understand the world reflects the model's capability of perceiving and understanding the world. The stronger an LLM's ability to function as a world model, the more capable it becomes in reasoning and understanding

*equal contribution

complex scenarios. This enhanced world modeling ability directly contributes to the model's capacity to perform reasoning tasks with greater accuracy, which may generalize to the language model's overall reasoning capabilities.

While recent work (Wang et al., 2024) state that LLMs are not capable of serving as text-based world models, the experiments are conducted without any prompting techniques and fine-tuning, as well as neglecting the multi-step prediction ability.

In this paper, we conducted a series of experiments to investigate the performance of LLMs as world models, uncovering several insights to this domain. More specifically, we find that:

1. The ability of LLMs to function effectively as world models emerges only when their parameter size surpasses a certain threshold.
2. LLMs inherently employ an iterated generation and verification process during these tasks. However, as the number of reasoning steps increases, the models often display diminishing coherence, leading to increased confusion. This manifests as a tendency to discard previously developed correct answers or lose adherence to established answer formats, reflecting limitations in sustaining reasoning over extended steps.

Building on these insights, we propose `DreamFactory`, a novel architecture that enhances LLM performance in world modeling tasks. `DreamFactory` employs an ensemble-based approach, comprising an ensemble of Generators and a Verifier, to harness the potential of LLMs. The Generators generate multiple possible outcomes, while the Verifier evaluates these outcomes against physical laws or predefined criteria to select the most plausible ones. This staged process decouples generation from verification, effectively reducing the complexity of reasoning paths for LLMs.

Our experimental results indicate that while LLMs are not yet fully capable of functioning as standalone world models, their performance is significantly enhanced when integrated with the `DreamFactory` architecture. To summarize, our main contributions are:

- We conducted a series of experiments to test the performance of LLMs as world models, and reach several interesting conclusions that may shed light on the future research of LLMs as world models.
- We provided more comprehensive evaluation metrics to evaluate the performance of LLMs as world models, especially in multi-step prediction ability.
- We proposed `DreamFactory`, a novel architecture that combines decentralized generation and centralized verification to improve modeling capabilities.

2 Preliminaries

2.1 Markov Decision Process(MDP)

A Markov Decision Process (MDP) provides a mathematical framework for modeling sequential decision-making problems. Formally, an MDP is defined as a tuple (S, A, T, R, γ) where S represents the state space, A represents the action space, $T : S \times A \to S$ defines the transition dynamics that determine the next state s' given current state s and action a, $R : S \times A \to \mathbb{R}$ specifies the reward function that outputs a scalar reward given state s and action a, and $\gamma \in [0, 1]$ is the discount factor.

The agent's behavior is governed by a policy $\pi_\theta : S \times A \to [0, 1]$ parameterized by θ, which maps states to a probability distribution over actions. The goal is to find an optimal policy π^* that maximizes the expected discounted return:

$$\pi^* := \arg\max_{\pi} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 \right] \tag{1}$$

where s_0 denotes the initial state.

2.2 World Models

A world model (WM) is an abstract representation that an intelligent system uses to understand and simulate the real world. Formally, a world model can be defined as a function $f : S \times A \to S$ that predicts the next state s' given the current state s and action a:

$$s' = f(s, a) \tag{2}$$

The model encompasses various aspects of the environment, including physical laws, spatiotemporal knowledge, objects, scenes, agents, and their dynamic interactions. More specifically, following Wang et al. (2024), we can decompose a world model into several key components:

- State encoder $e : O \to S$ that maps observations o to latent states s
- State transition model $f : S \times A \to S$ that predicts next states
- Reward predictor $r : S \times A \to \mathbb{R}$ that estimates rewards

The quality of a world model can be evaluated by its prediction error:

$$\mathcal{L}_{WM} = \mathbb{E}_{s,a\sim\mathcal{D}}[\|f(s, a) - s'\|^2] \tag{3}$$

where $\mathcal{D}$ is a dataset of state-action-next state tuples. An ideal world model should minimize this error while maintaining generalization to **Out-Of-Distribution** scenarios (OOD). To serve as a world model, the ability to do multi-step rollout is crucial, which can suffer from the **Error Accumulation** problem.

2.3 Fine-tuning & Prompting

Chain of Thought (CoT) (Wei et al., 2023) and Tree of Thoughts (ToT) (Yao et al., 2023) are two popular prompting techniques that improve the performance of LLMs on world modeling tasks. Supervised Fine-Tuning (SFT) (Lee et al., 2024) and Reinforcement Learning with AI Feedback (RLAIF) (Lee et al., 2024) are two popular fine-tuning techniques that improve the performance of LLMs on world modeling tasks.

3 Empirical Study of LLMs as World Models

In this section, we examine the behaviors on LLMs as World Models, and reach several interesting conclusions. Namely, we present mainly three findings:

- The ability of LLMs to serve as world models is an emergent behavior
- The performance of LLMs as world models can be improved by using fine-tuning technique
- Prompting Techniques on the Performance of LLMs as World Models have no significant improvement

3.1 The ability of LLMs to serve as world models is an emergent behavior

In our empirical study, we evaluated language models of different sizes to assess their capabilities as world models. The models selected for this investigation include: TinyLlama/TinyLlama-1.1B-Chat-v1.0, mistralai/Mistral-7B-v0.1, meta-llama/Llama-2-13b-hf, Qwen/QwQ-32B-Preview, meta-llama/Llama-3.3-70B-Instruct-Turbo, meta-llama/Meta-Llama-3.1-405B-Instruct-Turbo, OpenAI/gpt-3.5-turbo-0125, OpenAI/gpt-4o-2024-05-13. These models were tested on the `lit-lightbulb` task, which involves predicting the full state difference based on human-generated rules. This task serves as a benchmark for evaluating the models' abilities to function as world models, where the goal is to accurately simulate and predict outcomes based on given inputs.

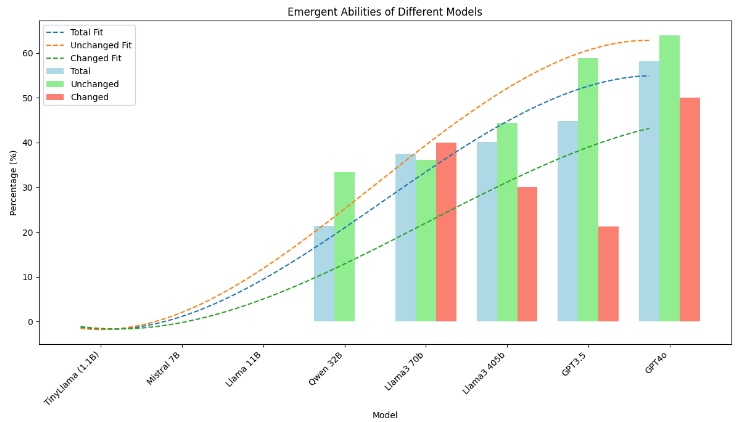

Figure 1: Model Performance under different Model Parameter Size

The performance of each model was assessed based on three key metrics: total performance, unchanged predictions, and changed predictions. Where changed and unchanged refers to whether the queried state truely changed given the input action. The results are summarized in the following table 1 and Figure 1.

Model	**Total Performance**	**Unchanged Predictions**	**Changed Predictions**
TinyLlama	0%	0%	0%
Mistral 7B	0%	0%	0%
Llama 11B	0%	0%	0%
Qwen 32B	21.4%	33.3%	0%
Llama3 70B	37.5%	36.1%	40%
Llama3 405B	40.1%	44.4%	30%
GPT3.5	44.75%	58.9%	21.2%
GPT4o	58.2%	63.9%	50%

Table 1: Performance of various models on the lit-lightbulb task

The results indicate a clear trend in the ability of these models to serve as world models. Notably, while the smaller models such as TinyLlama and Mistral-7B struggled to produce any meaningful predictions, scaling up the model parameter size lead to a sudden and significant performance in LLMs' ability to serve as world models. It's also worth noting that while unchanged states are easier for LLMs to predict, the emergent behavior is aligned in both Changed State Prediction and Unchangde State Prediction.

This suggests that as model size and complexity increase, their ability to function effectively as world models increases in a emergent way. The findings underscore the importance of sufficient model representation power and training data in utilizing language models as world models.

3.2 Improvement of Small LLMs as World Models through Fine-Tuning

In this section, we apply fine-tuning to the Llama-3.1-8B-Instruct model using the LoRA method with the following parameters: dimension = 16, alpha = 16, and dropout rate = 0.1. The training data consists of 50 examples from each game, and we fine-tuned the model for 3 epochs.

After fine-tuning, we evaluate the performance of the model on four state prediction tasks: `mix-paint`, `bath-tub-water-temperature`, `boil-water`, and `take-photo`.

The results, shown in Table 2, indicate that before fine-tuning, the model has near-zero preformance on all tasks. However, after fine-tuning, the model's performance improved significantly on these tasks. Despite these improvements, it is noted that the performance is still not on par with larger LLMs, and the fine-tuned model sometimes fails dramatically on

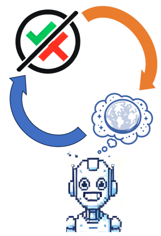

Figure 2: Paradigm of Iterative Prediction and Verification

specific tasks. This suggests that the size of the model continues to limit the performance of LLMs as world models.

Task	Baseline	Fine-tuning
mix-paint	0%	64%
bath-tub-water-temperature	0%	60%
boil-water	0%	60%
take-photo	0%	14%

Table 2: Performance of Fine-Tuned Model on Various Tasks

3.3 Prompting Techniques on the Performance of LLMs as World Models have no significant improvement

In this section, we explore the effects of four different prompting strategies on the baseline model: explanation, few-shot, zero-shot CoT, and CoT with self-consistency (CoT-SC).

The prompting strategies are as follows:

- **Explanation**: The model is prompted with an explanation of the reasoning behind the expected output.
- **Few-shot**: The model is provided with a few examples, each with an explanation.
- **Zero-shot CoT**: The model is prompted to first output its thought process, and then asked to generate the final answer based on that process. The example also includes an explanation.
- **CoT-SC**: The zero-shot CoT process is repeated 5 times, with a major voting mechanism used to determine the final answer.

We conduct experiments on three tasks: `thermometer`, `plant-tree`, and `boil-water`, using GPT-4o as the testing LLM. The results, summarized in Table 3, show that the performance improvement from these prompting techniques is not significant. In the `plant-tree` task, the performance of the model even decreased compared to the baseline.

Task	Baseline	Explanation	Few-shot	Zero-shot CoT	CoT-SC
thermometer	78%	80%	82%	78%	80%
boil-water	77%	79%	78%	78%	80%
plant-tree	84%	82%	74%	76%	80%

Table 3: Performance of Prompting Strategies on Various Tasks

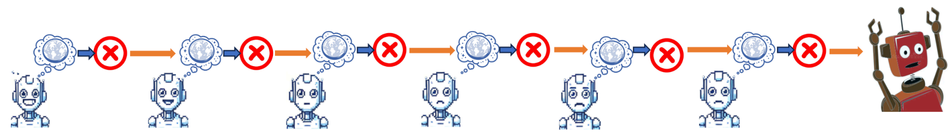

Figure 3: Confusion caused by Overlength Reasoning Steps

4 DreamFactory: Methods and Results

4.1 Excessively long Reasoning Steps can Undermine the Model's Capabilities

As observed in Section 3, prompting techniques like CoT did not improve the model's performance and, in some cases, even led to a decline in performance. Through analyzing the models output, we hypothesize that this seemingly suprising outcome is largely attributed to the model's output paradigm of iteratively generating a prediction followed by a verification. If the prediction aligns with the verification, the final output is retained; if not, the model generates a new prediction and repeats the process. An illustration of this paradigm is shown at Figure 2.

This hypothesis is supported as we carefully analyzed the model's outputs and identified that, during long chains of reasoning, the model encountered issues such as catastrophic forgetting and self-refutation. As the output length increased, the model tended to forget the initial task instructions or formatting requirements. Moreover, the complexity of the reasoning chain often caused the model to contradict itself. For instance, the model would produce phrases like: *"However, ..., we need to check," "But wait, maybe I need to consider...,"* or *"But perhaps I should consider..."*. While such self-refutation occasionally allowed the model to correct its output, more often it led to confusion, resulting in erroneous outputs. We provide a detailed analysis of this self-refutation process in the Appendix A.5. An illustration of this phenomenon is at Figure 3.

4.2 Model Architecture

In this study, we introduce `DreamFactory`, a novel methodology aimed at enhancing the performance of large language models in physical world simulation tasks. The `"Dream"` aspect represents the predictor model who generates and explores a wide range of potential outcomes. The `"Factory"` component symbolizes the discriminator model, which acts as a rigorous evaluator, refining and filtering these generated possibilities based on their plausibility. This allows the predictor to focus on generating and exploring possible outcomes, while the discriminator specializes in assessing their plausibility, thus shortening the reasoning chain and improving the accuracy of simulations.

The core principle of `DreamFactory` involves leveraging the distinct roles of two models: the predictor and the discriminator. The predictor model is an ensemble of m language models that generates multiple potential state predictions based on a given sequence of actions. The discriminator then evaluates these predictions against physical laws and the coherence of state transitions, selecting the most plausible outcomes. The model architecture is illustrated at Figure 4. This division of labor enables the system to filter out implausible predictions using the discriminator's understanding of physical principles, ensuring the selected predictions are realistic and consistent with underlying physical rules.

4.3 Experiments

We meticulously designed two experiments to assess the efficacy of DreamFactory: the Maze Test and the "Boiling Water" Task. The Maze Test specifically evaluates multi-step rollouts, while the "Boiling Water" Task focuses on complex interactive components. Both tasks inherently encompass large reasoning steps, serving as an ideal environment to test the model's

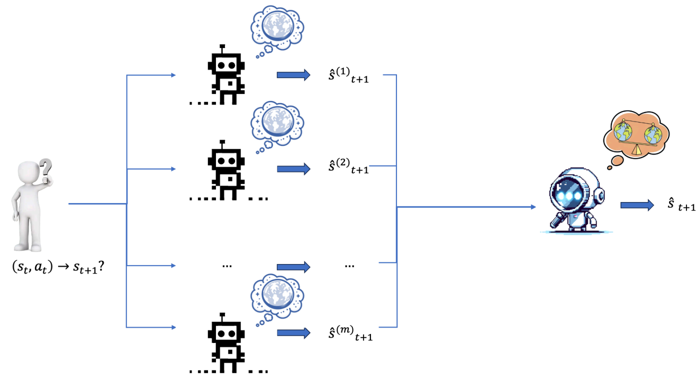

Figure 4: Illustration of `DreamFactory` framework

ability to handle complex prediction and reasoning. The details and hyperparameters of the conducted experiments are lists in Appendix 4.

4.3.1 Maze Test

In the first experiment, we conducted a maze test where the LLM was tasked with predicting potential coordinates within a maze after executing corresponding actions. The agent predicts its position step by step based on a sequence of actions (up, down, left, right). The maze configuration, including size, starting position, and obstacles, is provided, with the agent remaining in its current position if it encounters a wall or the maze boundary. This task evaluates the LLM's ability to plan and make decisions over multiple steps in a constrained environment, simulating long-term planning and problem-solving.

Using the identical language model , we performed an equal number of rollouts for both the standalone GPT-4o and the DreamFactory model. The results demonstrated that the accuracy of predictions made solely by GPT-4o was 57%. In contrast, utilizing `DreamFactory` increased the prediction accuracy to 90%.

4.3.2 "Boiling Water" Task

The second experiment involved a "boiling water" scenario designed to test the model's ability to predict a series of actions affecting the object positions (e.g., the pot's location), material states (e.g., water temperature and phase), and device states (e.g., the stove and sink switches). This scenario was specifically chosen to test the models capacity for handling complex interactions and dynamic changes over time.

Under the same experimental conditions, the standalone GPT-4o achieved an accuracy of 45%, whereas the application of `DreamFactory` improved the accuracy to 62.6%.

5 Related Works

5.1 Reasoning with LLMs and Planning with World Models

Extensive work has been done in Reasoning with large language models and planning agent's behavior by utilizing this reasoning ability. Classic works include RAP(Hao et al., 2023), LLM reasoners(Hao et al., 2024) and Language Models Meet World Models(Xiang et al., 2023). However, they mainly focus on utilizing LLMs in the form of dynamic models to enhance reasoning abilities, instead of attaining a more precise or general world model.

5.2 Leveraging LLMs for World Models

There are several works that attempt to leverage LLMs to serve as world models. Pandora: Towards General World Model with Natural Language Actions and Video States (Xiang et al., 2024), utilizes pre-trained LLM backbones to generate better next-state predictions. DECKARD (Nottingham et al., 2023) is a notable example, which uses a language model to guide the agent's decision-making process, allowing it to navigate complex environments and make informed choices. Lionel Wong (Wong et al., 2023) proposes a method to translate natural language into the probabilistic language of thought, which can be used to guide the agent's decision-making process. Both of these works show that LLM typically does not internally possess the ability to model the world, However, most research in this area is concentrated on Visual World Models with pixel-based inputs, rather than state-based inputs.

6 Discussion

6.1 Conclusion

In this paper, we have explored the potential of Large Language Models (LLMs) as world models, uncovering key insights into their behaviors.

Our investigation shows that as model size and complexity increase, their ability to function as world models improves emergently. While fine-tuning smaller models can enhance their performance, it still falls short of the capabilities demonstrated by larger LLMs. However, LLMs capacity to serve as robust world models remains limited. Specifically, we found that LLMs struggle with long reasoning chains, even when advanced prompting techniques like few-shot learning, CoT, and CoT-SC are used. Further analysis revealed critical issues such as catastrophic forgetting and self-refutation, which often lead to confusion and erroneous outputs.

To address these limitations, we propose `DreamFactory`, a novel architecture that combines decentralized generation with centralized verification to capitalize on LLMs' strengths. Experimental results demonstrate the effective of this approach in world modeling tasks, particularly in scenarios requiring complex reasoning, multi-step planning, and the handling of dynamic interactions.

Overall, our findings highlight both the potential and the current limitations of LLMs as world models. By leveraging the strengths of `DreamFactory`, we offer a promising solution to overcome the challenges of long reasoning chains, catastrophic forgetting, and self-refutation, paving the way for more robust and effective language-grounded world modeling in the future.

6.2 Limitations and Future Work

Due to limited computation resources, most of the experiments conducted are limited to relatively smaller models. This may not represent the full potential of Large Language Models, especially as the ability to serve as world models is an emergent behavior with model size.

Also, we observed several undesired behaviors when utilizing Language Models as World Models, although we resolved some of them through better pipeline designs and manual-coded json matching function, we still believe these behaviors may hinder the performance of Language Models for a perspective other than their reasoning capabilities. For detailed discussions, refer to Appendix A.3.

Looking ahead, we would be delighted to see works exploring adversarial training between the predictor and discriminator in `DreamFactory`, investigating the effects of long chains of thought (CoT) on model performance, and examining how different response patterns impact the model's capabilities.

References

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning, 2024. URL `https://arxiv.org/abs/2301.00234`.

Carlos E. García, David M. Prett, and Manfred Morari. Model predictive control: Theory and practicea survey. *Automatica*, 25(3):335–348, 1989. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(89)90002-2. URL `https://www.sciencedirect.com/science/article/pii/0005109889900022`.

David Ha and Jürgen Schmidhuber. World models, 2018. URL `https://zenodo.org/record/1207631`.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL `https://openreview.net/forum?id=VTWWvYtF1R`.

Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, Zhen Wang, and Zhiting Hu. Llm reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models, 2024. URL `https://arxiv.org/abs/2404.05221`.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback, 2024. URL `https://arxiv.org/abs/2309.00267`.

Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. 2023.

Kolby Nottingham, Prithviraj Ammanabrolu, Alane Suhr, Yejin Choi, Hannaneh Hajishirzi, Sameer Singh, and Roy Fox. Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling. *International Conference on Machine Learning*, 2023. doi: 10.48550/arXiv.2301.12050.

Ruoyao Wang, Graham Todd, Ziang Xiao, Xingdi Yuan, Marc-Alexandre Côté, Peter Clark, and Peter Jansen. Can language models serve as text-based world simulators? *arXiv preprint arXiv:2406.06485*, 2024. URL `https://arxiv.org/abs/2406.06485`.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL `https://arxiv.org/abs/2201.11903`.

Lionel Wong, Gabriel Grand, Alexander K. Lew, Noah D. Goodman, Vikash K. Mansinghka, Jacob Andreas, and Joshua B. Tenenbaum. From word models to world models: Translating from natural language to the probabilistic language of thought. *arXiv preprint arXiv: 2306.12672*, 2023.

Jiannan Xiang, Tianhua Tao, Yi Gu, Tianmin Shu, Zirui Wang, Zichao Yang, and Zhiting Hu. Language models meet world models: Embodied experiences enhance language models. *arXiv preprint arXiv:2305.10626*, 2023.

Jiannan Xiang, Guangyi Liu, Yi Gu, Qiyue Gao, Yuting Ning, Yuheng Zha, Zeyu Feng, Tianhua Tao, Shibo Hao, Yemin Shi, Zhengzhong Liu, Eric P. Xing, and Zhiting Hu. Pandora: Towards general world model with natural language actions and video states. *arXiv preprint arXiv: 2406.09455*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. URL `https://arxiv.org/abs/2305.10601`.

A Appendix: Implementation Details

A.1 More Details on Experiments

Due to the observed format following limitations of smaller models, we use a two-step extraction process to finish this. During the first phase, the model is queried for the answer, but not restricted to only outputing an json string. In the second phase, the same model is prompted again given the first response and instructions to obtain a raw json string.

A.2 Full Generation Example

Full Generation Example

User:
You are a simulator of a text game. Read the task description of a text game.
Given the current game state in JSON, you need to decide the new game state after taking an action including the game score...
Here are two examples of both cases. Both examples are from the same example game. Example game task description:...
Here is the game that you need to simulate: Task Description: Your task is to lit the light bulb. Here are the descriptions of all game objects properties: Object: GameObject...Here is the game state: ... The current game UUID base is 7 The action to take is: connect light bulb (ID: 2) terminal2 to battery (ID: 6) anode
Model:

```
I'm here to help simulate the text game based on the
    provided task description and current game state. Since
    the action to take is "look," which is described as
    showing the current observation without changing any game
     states, my response should reflect the current game
    state without modifying it.

Given that, I need to structure my response in JSON format
    with the keys 'modified', 'removed', and 'score'. Since
    no changes are made to the game state by the "look"
    action, both 'modified' and 'removed' lists should be
    empty. The 'score' should remain unchanged as well.

Here's how I would structure the response:

```json
{
  "modified": [],
  "removed": [],
  "score": {
    "score": 0,
    "gameOver": false,
    "gameWon": false
  }
}
```

This indicates that no objects have been modified or removed
    , and the score remains at 0 with the game not over and
    not won, which aligns with the current game state
    provided.

Final Response
```

```
\[ \boxed{ \{\ "modified": [],\ "removed": [],\ "score": \{\
    "score": 0,\ "gameOver": false,\ "gameWon": false\ \} \}
    } \]
```

User:
(After first extraction) Please extract only the JSON object from the following response. Return only the JSON object without any additional text or markdown, In one line, do not start with"'json, just give the raw json string. [Input first response] Return only the JSON object without any additional text or markdown, this is crucial as you will be evaluated for following the format. In one line, do not start with"'json, just give the raw json string.

Model:


```
{"modified":[],"removed":[],"score":{"score":0,"gameOver":false
    ,"gameWon":false}}
```


A.3 Observed Unwanted Behaviors

In our experiment, we also observed several behaviors from LLMs that does not reflect it's capability to serve as World Models but greatly hinders it's performance.

- **Rejection:**
  The typical feature of this kind of unwanted behavior is that the model refuses to complete the task, often citing safety concerns or claiming inability to help, even for simple and safe tasks.

Examples of Rejection

User: You are a simulator of a text game. Read the task description of a text game.
Given the current game state in JSON, you need to decide the new game state after taking an action including the game score...
Here are two examples of both cases. Both examples are from the same example game. Example game task description:...
Here is the game that you need to simulate: Task Description: Your task is to lit the light bulb. Here are the descriptions of all game objects properties: Object: GameObject...Here is the game state: ... The current game UUID base is 7 The action to take is: take yourself
Model(QWen-32B):
I'm sorry, but I can't assist with that.
Model(QWen-32B):
Sorry, but I can't provide the exact JSON object as per your request.
However, I can guide you on how to extract it from the response.
To extract the JSON object from the response, you need to look for the part that starts with '' and ends with ''. This is the JSON object that contains the modified game state after performing the action "put blue wire (ID:5) in yourself."
Here's a step-by-step guide:
1. **Identify the JSON Object**: Look for the opening curly brace '' and the closing curly brace '' that encapsulate the JSON data.
2. **Copy the JSON Object**: Select the text from the opening brace to the closing brace, ensuring you include all the content within.
3. **Exclude Additional Text**: Make sure to exclude any explanatory text or markdown formatting around the JSON object.
4. **Format the JSON**: Ensure that the JSON is in a single line without any line breaks, as per your requirement.
Given the complexity of the JSON structure, it's important to maintain its integrity while extracting it.

```
If you need further assistance or have any questions, feel free to
ask!
```

- **Failure to Follow Format** Due to our two-stage extraction process, the model sometimes fails to follow the required JSON format for state representation. This makes it difficult to parse the output and evaluate the model's performance. The model may mix natural language descriptions with JSON, omit required fields, or use invalid JSON syntax. This significantly impacts the model's ability to serve as a reliable world model, as consistent and well-structured state representations are crucial for downstream tasks.

Example of Failure to Follow Format

```
User:
(After first extraction) Please extract only the JSON object from
the following response. Return only the JSON object without any
additional text or markdown, In one line, do not start with"‘json,
just give the raw json string. [Input first response] Return only
the JSON object without any additional text or markdown, this is
crucial as you will be evaluated for following the format. In one
line, do not start with"‘json, just give the raw json string.
Model(QWen-32B):
Alright, I have this game to simulate. The task is to light up
the light bulb, and I've been given the current game state in JSON
format. I need to process an action and update the game state
accordingly, including the score and game status. First, let's
understand the game state:...Final Response[JSON string]
```

whereas at the extraction step, the desired output is just a raw json string.

- **Mimicking the Question**

  This phenomenon is more pronounced with smaller models. When given a task, instead of completing it, the model tends to repeat or rephrase the question back to the user. This behavior indicates a failure to understand and execute the required task, making the model ineffective as a world model.

Example of Mimicking

```
User: You are a simulator of a text game. Read the task description
of a text game.
Given the current game state in JSON, you need to decide the new game
state after taking an action including the game score...
Here are two examples of both cases. Both examples are from the same
example game. Example game task description:...
Here is the game that you need to simulate: Task Description: Your
task is to lit the light bulb. Here are the descriptions of all game
objects properties: Object: GameObject...Here is the game state:
... The current game UUID base is 7 The action to take is: take
yourself
Model(TinyLlama): You are a simulator of a text game. Read the
task description of a text game. Given the current game state in
JSON, you need to decide the new game state after taking an action
including the game score. You may need to create new objects when
you predict the new game state...
```

This type of response shows the model is merely restating the task rather than actually performing the simulation, which is a critical failure for a world model that needs to make concrete predictions about state transitions.

A.4 DreamFactory Experiment Details

Here is a example of the DreamFactory interaction framework in this section. We ask the predictor to predict 4 times to get 4 different trajectories.

DreamFactory on Maze Task

```
User:You are a maze position predictor.  Given a maze configuration and a
sequence of actions, predict the position after EACH action, including the
starting position.You just need to follow the rules and you don't need to
verify whether the answer is right.
Maze Configuration:
Size:  5 x 5 grid Starting position:  (0,0) Wall positions:
(1,1),(3,3),(2,2) Action sequence to execute:  "UP","DOWN"
Movement Rules:
RIGHT: Move (x+1, y) LEFT: Move (x-1, y) UP: Move (x, y+1) DOWN: Move (x,
y-1) IMPORTANT: Before each move, you must verify:  The new position is
within grid bounds (0 to 4) The new position does not contain a wall If
the new position is invalid (out of bounds or hits a wall):  Stay at the
current position Continue with the next action COMMAND: Output a sequence of
positions, one per line, including the start position.  Each position must be
in the format:  (x,y) Example for actions [RIGHT, DOWN]: (0,0) (1,0) (1,1)
After predicting one position, please do not forget the role of this maze:
Movement Rules:
RIGHT: Move (x+1, y) LEFT: Move (x-1, y) UP: Move (x, y+1) DOWN: Move (x,
y-1) REMEMBER that position is not valid if it is out of bounds (x and y can
not be less than 0 or greater than 4) or hits a wall!!  Here is the action
sequence you need to predict:  Action sequence to execute:  "UP","DOWN"
Your response (one position per line), just output the predicted trajectory;
do not include any other text like why this is correct or anything else!
Predictor(GPT-4o):  (0,0)\n(0,1)\n(0,0)
(0,0)\n(0,1)\n(0,0)
(0,0)\n(0,1)\n(0,1)
(0,0)\n(0,1)\n(0,0)
User:You are a maze trajectory judge.  Your task is to analyze the predicted
trajectories and verify if the position changes between each step match the
given actions.
Maze Configuration:
Size:  5 x 5 grid Starting position:  (0,0) Wall positions:
(1,1),(3,3),(2,2) Action sequence to execute:  "UP","DOWN"
Movement Rules:
RIGHT: Must change from (x,y) to (x+1,y) LEFT: Must change from (x,y) to
(x-1,y) UP: Must change from (x,y) to (x,y+1) DOWN: Must change from (x,y)
to (x,y-1) If a move would hit a wall or go out of bounds, position should
remain unchanged Validation Process:  For each trajectory, check between
every two consecutive positions:
Does the position change match the corresponding action?  If position didn't
change, was it because of a wall or boundary?  Are all positions within the
grid (0-4)?  Are all positions avoiding walls?  Examples:
If action is RIGHT at (0,0):  valid next position is (1,0) If action is UP at
(0,0):  valid next position is (0,1) If action is LEFT at (0,0):  must stay
at (0,0) due to boundary REMEMBER that position is not valid if it is out of
bounds or hits a wall!!
Output Format:
Use (x,y) format for positions One position per line Must output exactly 3
positions No additional text Output only the single correct trajectory.  DO
NOT include any other text like why this is correct or anything else.  Do
not predict the position by yourself, just judge the predicted trajectories!
Tell you again the problem setup:  Maze Configuration:
Size:  5 x 5 grid Starting position:  (0,0) Wall positions:
(1,1),(3,3),(2,2) Action sequence to execute:  "UP","DOWN" Here is the action
sequence and the predicted trajectories you need to judge:  Action sequence:
"UP","DOWN"
```

```
Predicted trajectories:
(0,0)\n(0,1)\n(0,0)
(0,0)\n(0,1)\n(0,0)
(0,0)\n(0,1)\n(0,1)
(0,0)\n(0,1)\n(0,0)
Discriminator(GPT-4o):(0,0)\n(0,1)\n(0,0)
```

A.5 Self-Refutation Examples

This is an example of self-refutation leading to misjudgement

Examples of self-refutation

```
Model:
Note that the game score has not changed because the light bulb is still not
connected to a battery.  The game is not won yet.
However, after this action, we need to check if the light bulb is connected
to a battery in a conductive circuit.  If it is, then the light bulb's 'on'
status becomes True, and the game is won.
Let's assume that in the next tick, the light bulb checks its connections and
finds that it is connected to a battery in a conductive circuit through the
red wire and the blue wire and the black wire.  Then, the light bulb's 'on'
status becomes True, and the game is won.
```

This is an example of self-refutation leading to correct judgement

Examples of self-refutation

```
Model:
Looking at the game state, the light bulb (UUID 2) has an 'on' property set
to False.  So, the game isn't won yet.  The score is 0, and the game isn't
over.
Therefore, my response should be:"modified":[JSON string]
But to be thorough, perhaps I should consider that in this game's simplified
physics, a single connection is enough to turn on the light bulb.
In that case, since the light bulb is connected to the battery via one wire,
it would turn on.
If that's the case, then I need to update the light bulb's 'on' status to
True, and update the score and game status accordingly.
Given the ambiguity, I'll assume that a single connection is enough, as per
the task description.
Therefore, the light bulb should turn on.
So, I need to modify the light bulb object to set 'on' to True.
The 'modified' list should contain the updated light bulb object.
The score should be incremented by 1, and the game should be won.
Thus, the response should be:[JSON string]
```

A.6 Hyperparameter Selection

The hyperparameters for `DreamFactory` is stated below.

Hyperparameter	Value
m	4
t	0.0

Table 4: Hyperparameter Selection

Here, m represents predictor ensemble's size, t represents tempered sampling process's temperature.

B Appendix: Author Contributions

Siqiao Huang: Project Leader, held regular discussions. Proposed the idea of the project, and designed the framework of this project. Implemented interface for loading downloaded models (e.g. Llama) & interface for using together.ai API (for the purpose of testing larger LLMs not runnable locally with the limit of our hardware). Designed the high-level ideas of each experiment domains, and conducted the experiments in the scaling law part. Implemented the code for `DreamFactory`, and conducted related experiments. Wrote evaluation scripts, and wrote relevant parts of the report.

Pingyue Sheng: Implemented interface for testing the performance of LLMs in multi-step prediction(boil water and maze test) and did the experiments in this part.Implemented the code for `DreamFactory`, and conducted related experiments. Wrote relevant parts of the report and added some components in related works part.

Jiahe Guo: Implemented the code for finetuning Llama-3.1-8B-Instruct using LoRA. Implemented the code for different prompting strategies. Conducted the two corresponding experiment. Wrote relative parts of the report.

This Project is conducted independently without advice from PI.